

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

Samsung Electronics Co., Ltd. and Samsung Electronics America, Inc.,
Petitioner,

v.

RADIAN MEMORY SYSTEMS, LLC,
Patent Owner.

Case IPR2025-01350

U.S. Patent No. 11,740,801

**PETITION FOR *INTER PARTES* REVIEW
UNDER 35 U.S.C. §§ 311-319 AND 37 C.F.R. §§ 42.100 *et seq.***

TABLE OF CONTENTS

I.	INTRODUCTION.....	9
II.	STANDING.....	9
III.	FOUNDATIONS.....	9
IV.	LEVEL OF ORDINARY SKILL IN THE ART.....	9
V.	CLAIM CONSTRUCTION	10
VI.	TECHNICAL BACKGROUND	10
VII.	OVERVIEW OF THE '801 PATENT	11
A.	Priority	11
B.	Specification	11
VIII.	GROUND I: CLAIMS 1-4, 6, 9-16, AND 21, 24 OBVIOUS IN VIEW OF SINCLAIR AND YAMADA.....	12
A.	Overview of the Prior Art	12
1.	Sinclair (EX1004).....	12
2.	Yamada (EX1005).....	12
B.	Claim-by-Claim Analysis	12
1.	Claim 1	12
2.	Claim 2	38
3.	Claim 3	40
4.	Claim 4	43
5.	Claim 6	46
6.	Claim 9	47
7.	Claim 10	53
8.	Claim 11	54
9.	Claim 12	55
10.	Claim 13	56
11.	Claim 14	57
12.	Claim 15	57
13.	Claim 16	58
14.	Claim 21	59
15.	Claim 24	65
IX.	GROUND II: CLAIMS 1-4, 6, 9-16, AND 21, 24 OBVIOUS IN VIEW OF TERASAKI AND YAMADA.....	66
A.	Overview of the Prior Art	66

1. Terasaki (EX1018)	66
B. Claim-by-Claim Analysis	66
1. Claim 1	66
2. Claim 9	85
3. Claim 13	88
4. Claim 14	88
5. Claim 15	90
X. GROUND III: CLAIM 22 IS OBVIOUS IN VIEW OF SINCLAIR, YAMADA AND OLBRICH.....	91
A. Overview of the Prior Art	91
1. Olbrich (EX1010).....	91
B. Claim-by-Claim Analysis	92
1. Claim 22	92
XI. GROUND IV: CLAIM 23 IS OBVIOUS IN VIEW OF SINCLAIR, YAMADA, SINCLAIR-II, AND YU.....	93
A. Overview of the Prior Art	93
1. Sinclair-II (EX1015).....	93
2. Yu (EX1016)	93
B. Claim-by-Claim Analysis	94
1. Claim 23	94
XV. MANDATORY NOTICES AND FEES	95
A. Real Party-In-Interest.....	95
B. Related Matters	96
C. Counsel and Service Information.....	96
D. Payment of Fees.....	96
XVI. CONCLUSION	97
CERTIFICATE OF WORD COUNT.....	98
CERTIFICATE OF SERVICE	1

TABLE OF EXHIBITS

Ex.	Description
1001	U.S. Patent No. 11,740,801 (“’801 patent”)
1002	Prosecution History of U.S. Patent No. 11,740,801
1003	Expert Declaration of Dr. Heiner Litz
1004	U.S. Publication No. 2009/0271,562 (“Sinclair”)
1005	U.S. Publication No. 2011/0271,032 (“Yamada”)
1006	U.S. Pub. No. 2012/0096217 (“Son”)
1007	U.S. Pub. No. 2008/0189485 (“Jung”)
1008	U.S. Patent No. 9,665,291 (“Lin”)
1009	<i>Intentionally Omitted</i>
1010	U.S. Patent 8,621,137 (“Olbrich”)
1011	Yang Hu et al., Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity (“Hu”)
1012	U.S. Patent 9,239,781 (“Ellis”)
1013	Certified Translation of JP2008176606A (“Takeo”)
1014	JP2008176606A
1015	International Publication No. WO 2008/082996 (“Sinclair-II”)
1016	U.S. Pub. No. 2012/0284587 (“Yu”)
1017	U.S. Patent 9,423,970 (“Darragh”)
1018	Certified Translation of JP2005018490A (“Terasaki”)
1019	JP2005018490A

CHALLENGED CLAIMS

Claim	Limitation
1[pre]	A storage device comprising:
1[a]	NAND flash memory; and
1[b]	a memory controller having logic to cause the memory controller to
1[c]	map non-overlapping ranges of logical addresses to respective subdivisions of physical storage in the NAND flash memory, each of the respective subdivisions comprising erase blocks that must each be physically erased as a unit,
1[d]	receive write requests, accompanying addresses and accompanying data from a host and, with respect to each one of the write requests,
1[d.i]	identify a specific one of the respective subdivisions dependent on which one of the non-overlapping ranges encompasses the address accompanying the one of the write requests,
1[d.ii]	program the data accompanying with the one of the write requests into a next available storage location within the specific one of the respective subdivisions, and
1[d.iii]	in association with the programming of the data accompanying the one of the write requests, update information, for the specific one of the respective subdivisions, representing an extent to which the erase blocks of the specific one of the respective subdivisions are full,
1[e]	store a value representing a size of the NAND flash memory that is to be physically erased in fulfilling a maintenance request issued by the host,
1[f]	provide to the host, responsive to a query received from the host, the information, as updated, and the stored value,
1[g]	update metadata for the specific one of the respective subdivisions, wherein the metadata represents a characteristic of the data in the specific one of the respective subdivisions,
1[h]	with respect to the metadata, as updated, perform a comparison based on the metadata to detect a condition, and
1[i]	store information, responsive to detection of the condition, wherein the information is accessible to the host and indicates that the condition was detected;

Claim	Limitation
1[j]	wherein each said logic comprises at least one of circuitry or instructions stored on physical storage media that, when executed, are to control operation of circuitry.
2[a]	The storage device of claim 1 wherein: the metadata comprises information representing an age-since-write of data stored in the specific one of the respective subdivisions; and
2[b]	the memory controller also comprises logic to, responsive to the identification of the condition, control copying of data associated with an age-since-write greater than a threshold to a new physical storage location in the NAND flash memory.
3[a]	The storage device of claim 2 wherein the maintenance request is an erase request;
3[b]	the logic is to further cause the memory controller to receive a copy request from the host, the copy request directed to the specific one of the respective subdivisions;
3[c]	the copy request is to cause the memory controller to control the copying in a manner responsive to the copy request;
3[d]	the erase request is to cause the memory controller to erase a source location of the data being copied; and
3[e]	the copy request and the erase request are respective requests which are received by the memory controller at different times.
4[a]	The storage device of claim 1 wherein: the memory controller further comprises logic to receive a query from the host;
4[b]	the comparison is between the metadata and a threshold; and
4[c]	the information that indicates that the condition was detected is sent to the host responsive to the query and is to provide to the host with at least one address associated with the specific one of the respective subdivisions.
6	The storage device of claim 4 wherein the at least one address comprises a logical address corresponding to the one of the non-overlapping ranges which has been mapped to the specific one of the subdivisions.
9[a]	The storage device of claim 1 wherein: the metadata is a function of data error in the data being characterized; and

Claim	Limitation
9[b]	the condition corresponds to a determination by logic of the memory controller that data should be moved to a new physical location in the NAND flash memory.
10[a]	The storage device of claim 1 wherein the metadata represents a data read frequency associated with the data being characterized; and
10[b]	the condition corresponds to a determination by logic of the memory controller that data should be moved to a new physical location in the NAND flash memory.
11[a]	The storage device of claim 1 wherein: the metadata represents a data rewrite frequency of the data being characterized; and
11[b]	the condition corresponds to a determination by the memory controller that data should be moved to a new physical location in the NAND flash memory.
12[a]	The storage device of claim 1 wherein: the metadata comprises information representing an age-since-write of data stored in the specific one of the respective subdivisions; and
12[b]	the memory controller also comprises logic to, responsive to the identification of the condition, copy data associated with the specific one of the respective subdivisions to a new physical storage location in the NAND flash memory.
13	The storage device of claim 1 wherein the logic comprises at least one processor and instructions stored on at least one physical storage medium, wherein said instructions when executed, are to control operations of the at least one processor.
14	The storage device of claim 1 wherein the storage device further comprises circuitry to detect a failure condition of one of the erase blocks in the specific one of the respective subdivisions and to transmit information to the host that identifies that a storage location in the specific one of the respective subdivisions has experienced the failure condition.
15[a]	The storage device of claim 1 wherein: the storage device comprises circuitry to detect a failure condition of one of the erase blocks of the flash memory; and

Claim	Limitation
15[b]	the logic is to cause the memory controller to, for one of the write requests which is addressed to a given one of the respective subdivisions corresponding to the one of the erase blocks for which a failure condition has been detected, transmit to the host a notification of a write failure and logical address information identifying the given one of the respective subdivisions corresponding to the one of the erase blocks for which the failure condition has been detected.
16[a]	The storage device of claim 1 wherein: for each of the write requests, the logic is to cause the memory controller to generate an address offset that designates a storage location within an addressed one of the respective subdivisions; and
16[b]	the logic is to further cause the memory controller to transmit the address offset to the host.
21[a]	The storage device of claim 1 wherein: the NAND flash memory comprises physical planes, wherein the specific one of the respective subdivisions comprises erase blocks that are in respective ones of the physical planes; and
21[b]	the logic is further to cause the memory controller to identify a need for maintenance for the specific one of the respective subdivisions dependent on the respective metadata, and an associated logical address, and
21[c]	transmit the associated logical address to the host, wherein an erase request from the host is dependent on the associated logical address which was transmitted to the host, and
21[d]	responsive to the erase request, control erasure of two or more of the erase blocks corresponding to the specific one of the respective subdivisions, including erase blocks in each of at least two different ones of the physical planes.
22[a]	The storage device of claim 1 wherein: the NAND flash memory comprises physical planes, wherein the specific one of the respective subdivisions comprises erase blocks that are in respective ones of the physical planes; and
22[b]	the logic is to cause the memory controller to, in response to the one of the write requests, control writing of the data into the erase blocks that are in the respective ones of the physical planes using a common address offset for each of the respective ones of the physical planes.

Claim	Limitation
23[a]	The storage device of claim 1 wherein the logic is further to cause the memory controller to: in connection with physical erasure of the one or more erase blocks, detect a defect in a given erase block of the one or more erase blocks being erased and, responsive to detecting the defect, to remap each logical address which corresponds to the given erase block to a different erase block in the nonvolatile memory; and
23[b]	direct ones of the write requests from the host, for which the accompanying address designates a logical address which has been remapped responsive to detection of the defect, to a storage location associated with the different erase block
24	24. The storage device of claim 1 wherein the logic is further to cause the memory controller to receive an erase request from the host which is directed to the specific one of the respective subdivisions, responsively control the physical erasure of the one or more of the erase blocks in the specific one of the respective subdivisions and, in association with the physical erasure, modify the information representing the extent to which the one or more of the erase blocks of the specific one of the respective subdivisions are full.

I. INTRODUCTION

Samsung Electronics Co., Ltd. and Samsung Electronics America, Inc., (collectively, “Samsung” or “Petitioner”) requests IPR review of claims 1-4, 6, 9-16, and 21-24 of U.S. Patent No. 11,740,801 (“’801 patent”). A reasonable likelihood exists that Petitioner will prevail on at least one challenged claim for the reasons discussed below.

II. STANDING

Petitioner certifies the ’801 patent is available for *inter partes* review and is not estopped under 35 U.S.C. § 325.

III. GROUNDS

Ground	Basis	Reference(s)	Claims
I	§ 103	Sinclair and Yamada	1-4, 6, 9-16, 21, 24
II	§ 103	Terasaki and Yamada	1, 9, 13-15
III	§ 103	Sinclair, Yamada and Olbrich	22
IV	§ 103	Sinclair, Yamada, Sinclair-II and Yu	23

IV. LEVEL OF ORDINARY SKILL IN THE ART

A POSITA at the time of the ’801 patent would have had at least a Bachelor of Science degree in electrical engineering, or an equivalent field, and at least three years of experience with SSDs or other flash memory devices. EX1003, ¶43. More education can supplement practical experience and vice versa. *Id.* Petitioner’s expert, Dr. Litz, exceeded this skill level by the priority date. *Id.*, ¶¶4-11.

V. CLAIM CONSTRUCTION

No express constructions are required to find the '801 patent invalid; however, Petitioner addresses the plain meaning of certain terms in the analysis for the presented Grounds.

VI. TECHNICAL BACKGROUND

Flash memory is organized into erase units or blocks. EX1003, ¶¶28-29, 32-33. Blocks are comprised of pages, which represent the smallest unit that may be written or programmed. *Id.* To write the pages of a block with new data, the block (i.e. all pages within the block) must first be erased. EX1003, ¶33. A block may only be erased a certain number of times prior to inoperability. EX1003, ¶¶35-36.

To increase the efficiency of flash memory, SSDs may employ a larger unit of memory. For example, SSDs may be organized into groupings of blocks operated as a single unit, termed, *e.g.*, “super blocks,” EX1012, 6:43-65, “metablocks,” EX1004, ¶5, EX1017, 6:1-7, or “zones,” EX1013, ¶¶41-42; EX1003, ¶41. Moreover, to increase parallelism, blocks of data in a grouping may be written or erased together, including blocks that are on different planes. EX1011, §2.2 (“Multi-plane command activates multiple read, program or erase operations in all planes of the same die”), Fig. 4. EX1003, ¶41.

To increase the longevity of flash memory, SSDs perform maintenance operations. EX1003, ¶¶35-36. Memory maintenance functions may be carried out

by the controller or via shared responsibility with the host. EX1003, ¶¶38-39. This is “a cooperative memory management system.” EX1007, ¶¶11-12.

VII. OVERVIEW OF THE '801 PATENT

A. Priority

The '801 patent claims priority to a provisional application filed on January 28, 2013. The prior art predates this date.

B. Specification

The specification describes a memory with subdivisions where “each subdivision is an EU or page of NAND flash memory.” EX1001, 26:45-47. “[T]he storage retains information [or metadata] for each EU or physical page of the flash memory.” EX1001, 27:29-33.

The controller provides metadata to the host in various forms: (i) “a map to the host indicating relative information (e.g., page utilization) for all subdivisions” or “derived information based... on some threshold,” *id.*, 26:47-54; (ii) “wear metric (e.g., EU erase count) exceeds a programmably-defined value,” *id.*, 34:44-35:21; and (iii) an “age field” metric which tracks “how long it has been since data was written to the erase unit,” *id.*, 39:2-35. The information may also be shared with the host in response to “a query” or “asynchronous command,” based on satisfaction of a condition or threshold. *Id.*, 28:10-59.

VIII. GROUND I: CLAIMS 1-4, 6, 9-16, AND 21, 24 OBVIOUS IN VIEW OF SINCLAIR AND YAMADA.

A. Overview of the Prior Art

1. Sinclair (EX1004)

Sinclair is prior art under at least AIA 35 U.S.C. § 102(a). *See* EX1003, ¶¶45-56.

2. Yamada (EX1005)

Yamada is prior art under at least AIA 35 U.S.C. § 102(a). *See* EX1003, ¶¶57-59.

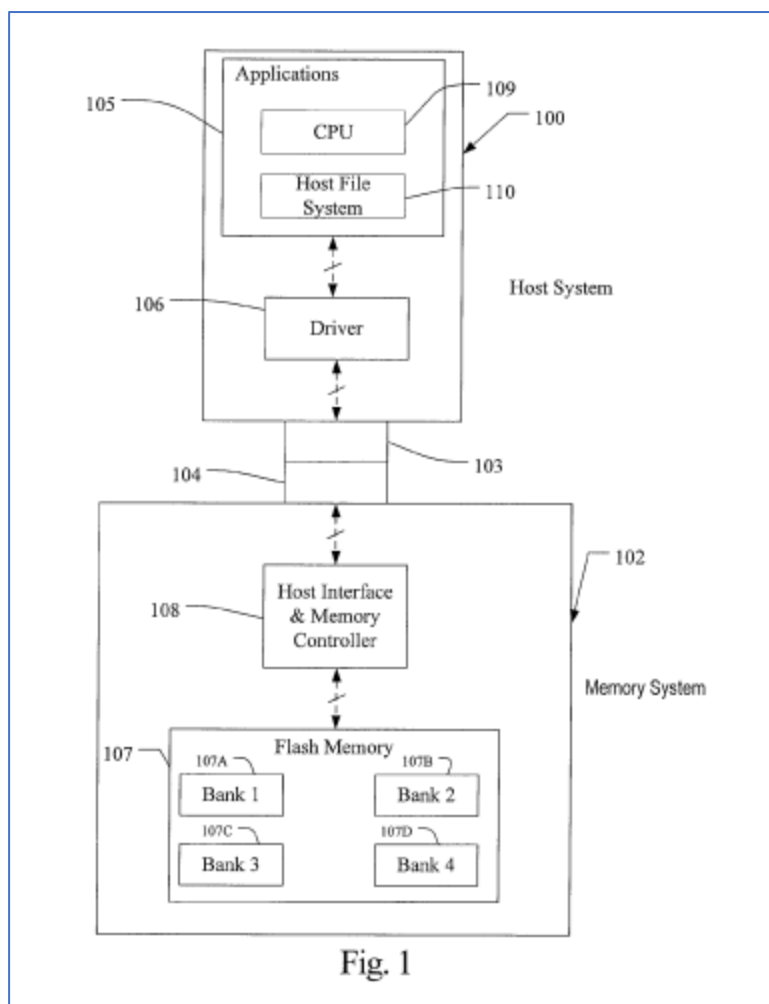
B. Claim-by-Claim Analysis¹

1. Claim 1

1[pre]

To the extent the preamble is limiting, Sinclair discloses a storage device. EX1003, ¶88. For instance, Figure 1 discloses a memory system 102 comprises a host interface and memory controller 108 and a flash memory 107, such as “in the form of a solid state disk (SSD) drive” installed in a personal computer:

¹ *See* Table of Claims, *supra*, for full text of limitations.



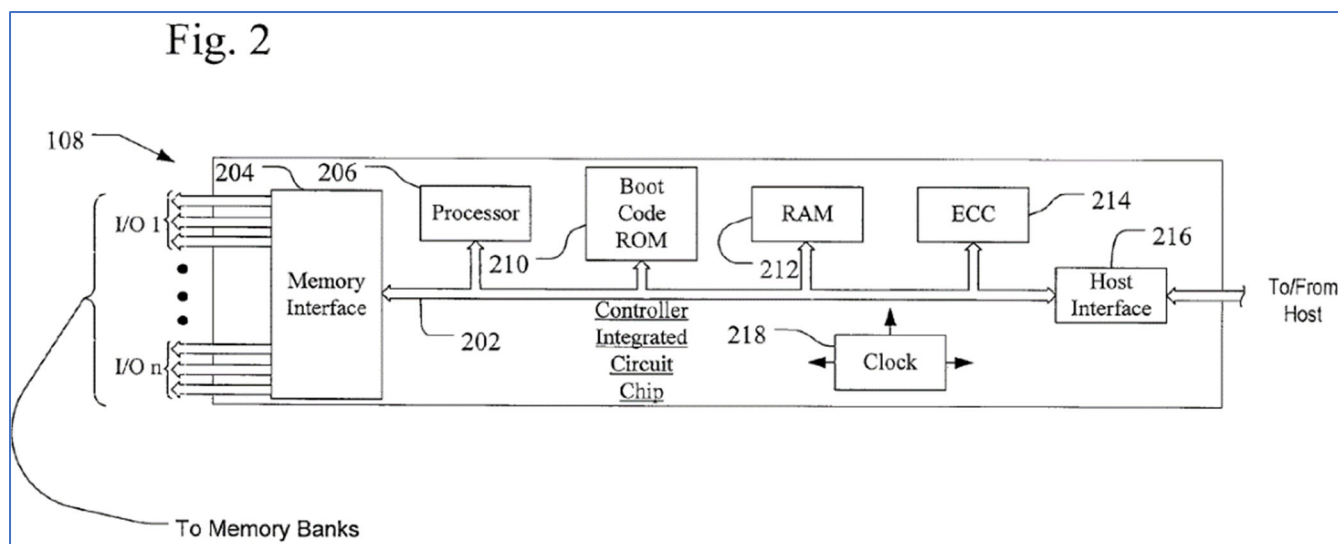
EX1004, Fig. 1, ¶¶40, 43-44; EX1003, ¶88.

1[a]

Sinclair discloses that its memory system employs a “NAND architecture” and includes “NAND flash memories” or a “NAND array” as part of its memory system. EX1004, ¶49, Figs. 1, 3-4; EX1003, ¶90.

1[b]

Sinclair discloses a “memory controller 108 in the memory system 102” as shown in Figures 1 and 2:

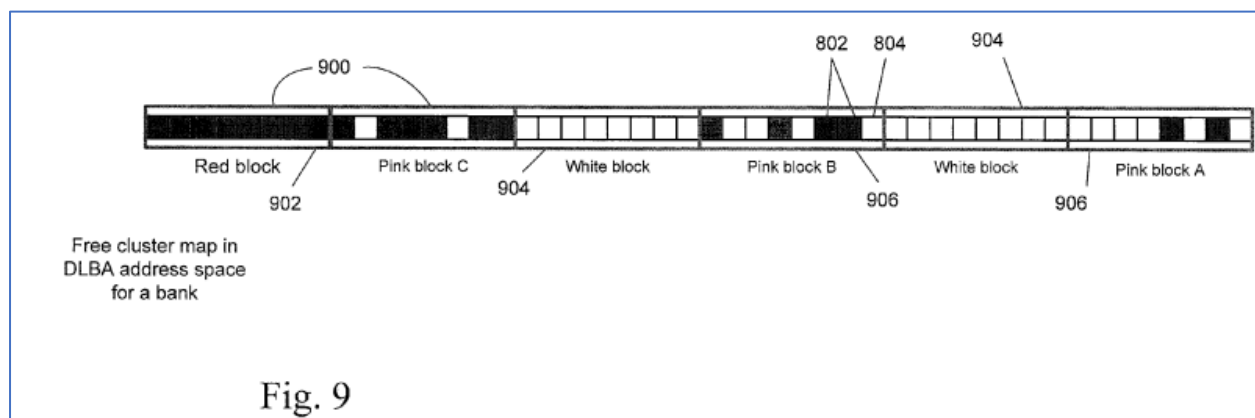


EX1004, ¶44, Figs. 1-2; EX1003, ¶92. The memory controller has logic to cause the controller to perform the claimed limitations, as discussed below, by interfacing with the host to which the memory system is connected for passing data back and forth and by controlling the memory 107. EX1004, ¶44; EX1003, ¶93.

1[c]

Sinclair discloses that its NAND flash memory is organized into “megablocks” [subdivisions of physical storage in the NAND flash memory]. EX1004, ¶¶53-56, 63; Figs. 5-7; EX1003, ¶95. The megablocks are comprised of metablocks, which are a grouping of blocks that are “logically linked together” and which can be “erased together” [each of the respective subdivisions comprising erase blocks that must each be physically erased as a unit]. EX1004, ¶54 (“One block from each plane is logically linked together to form a metablock.... All of the cells within a metablock are typically erased together.”); EX1003, ¶95.

Sinclair also discloses that the memory system maps a non-overlapping range of “continuous logical address space 800” to megablocks and metablocks. EX1004, ¶¶5 (“allocating a megablock of contiguous storage LBA addresses for addressing the data associated with the host LBA addresses, the megablock of contiguous storage LBA addresses comprising at least one block of memory cells in each of a plurality of banks of memory cells in the mass storage system.”), ¶¶57-58 (storage system maps “LBA addresses in terms of metablocks of clusters 900, referred to generally as ‘blocks’ in the discussion below.”); EX1003, ¶96. For instance, Figure 9 illustrates a contiguous [non-overlapping range] logical address space mapped to a megablock comprising metablocks (*i.e.*, red blocks, pink blocks, and white blocks):



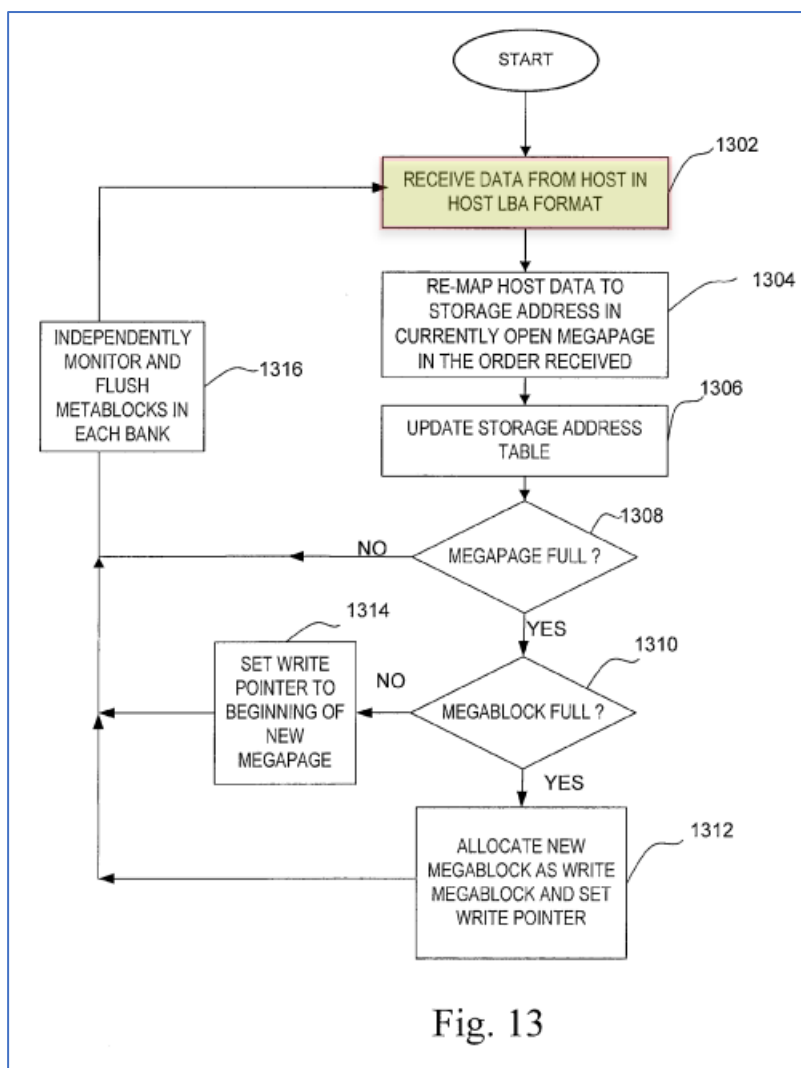
EX1004, Fig. 8, ¶¶57-58, Figs. 8-9; EX1003, ¶96.

1[d]

Sinclair discloses its controller receives write requests. EX1003, ¶98. For instance, Sinclair discloses a “megablock write algorithm” in which its storage device receives “a host data write operation” [receive write requests]. Ex. 1004,

¶63; EX1003, ¶98. Sinclair explains that, as part of the write request, the storage device “receiv[es] host data from a host file system 10” [**accompanying data from a host**] and a “host LBA [logical block address]” [**accompanying addresses**]. Ex.

1004, ¶63 (“original host LBA addresses”), 5 (“host LBA addresses”). The host data is then remapped and written to a megapage, which refers to a metapage from each of the metablocks in a megablock. EX1004, ¶¶63, 56; EX1003, ¶98. More specifically, “the host data is re-mapped to a storage address by writing the host data to the currently open megapage in the currently open write megablock in the order it is received regardless of host LBA order.” Ex. 1004, ¶63, ¶5 (“receiving data associated with host logical block address (LBA) addresses assigned by the host system....”); EX1003, ¶98. This is illustrated in steps 1302-1304 below:



Ex. 1004, Fig. 13.; EX1003, ¶98.

1[d.i]

Sinclair discloses **identifying a specific one of the respective subdivisions** during a write request from the host by identifying a currently open write megablock [a **subdivision**]. EX1003, ¶100. As illustrated in step 1304 of Figure 13, Sinclair discloses “receiving data associated with a host logical block address” and “[a]s the host data is received, the data is re-mapped to a storage address by writing the host

data to the currently open megapage in the currently open write megablock.”

EX1004, ¶¶5, 63, Fig. 13; EX1003, ¶100.

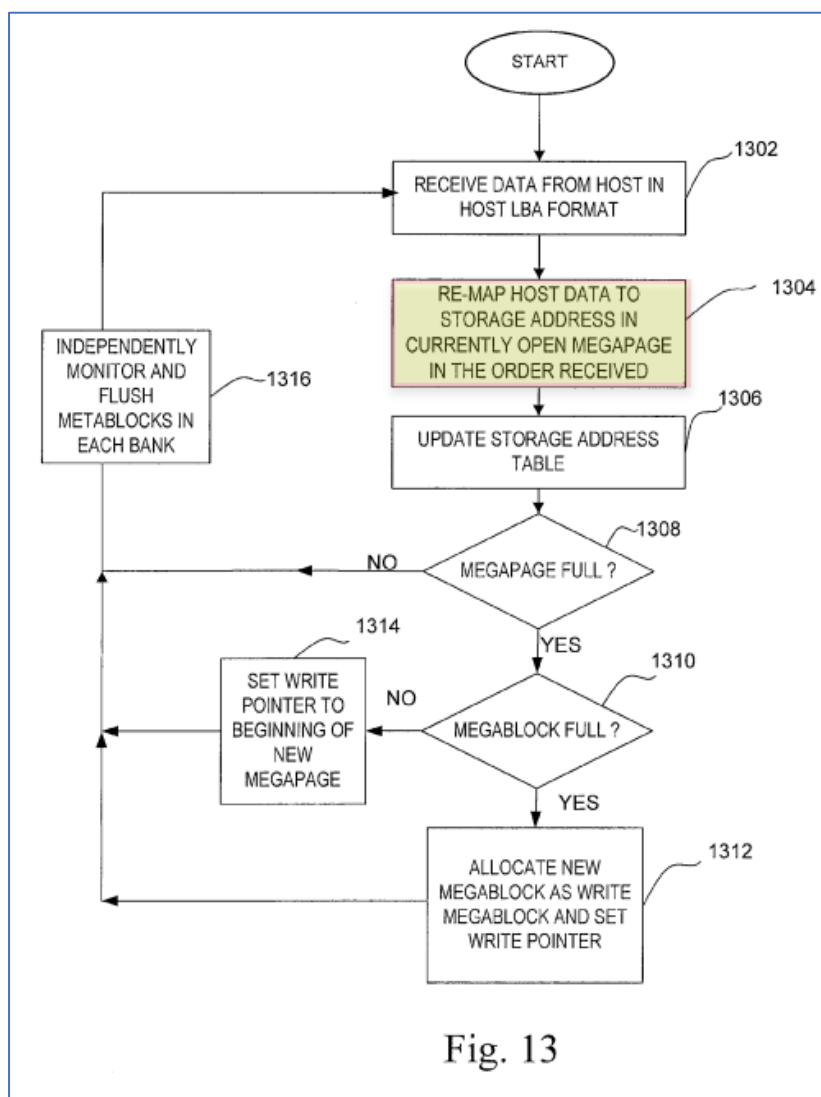
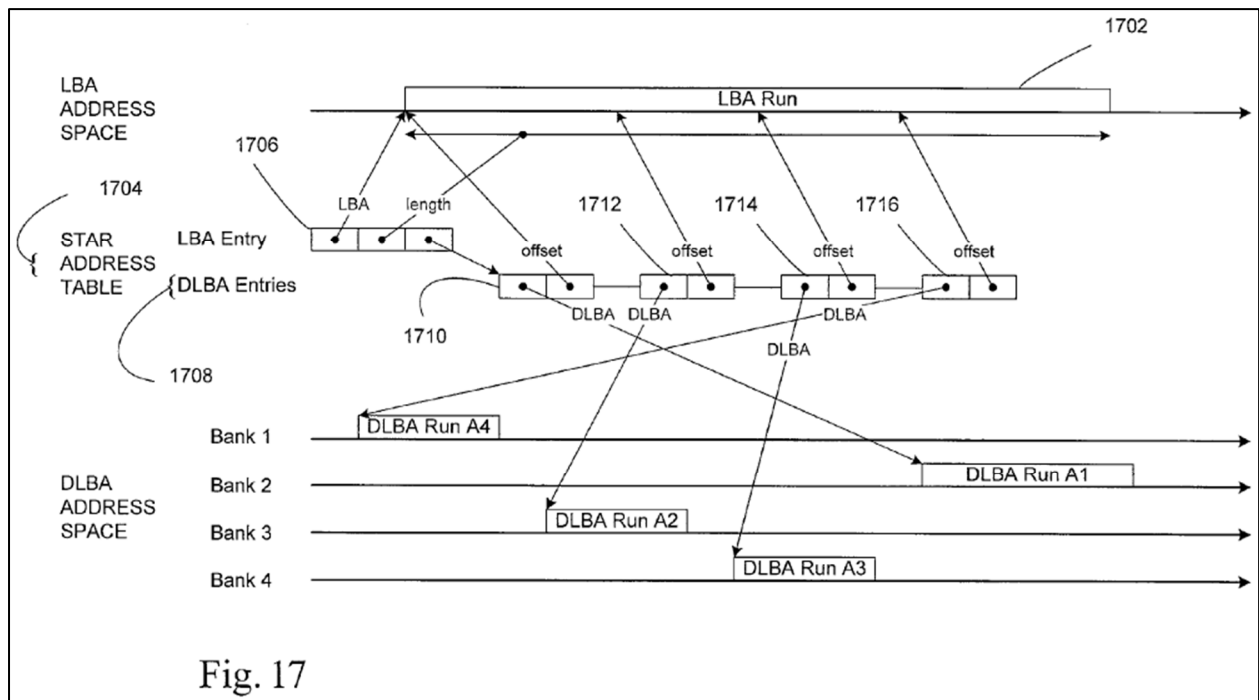
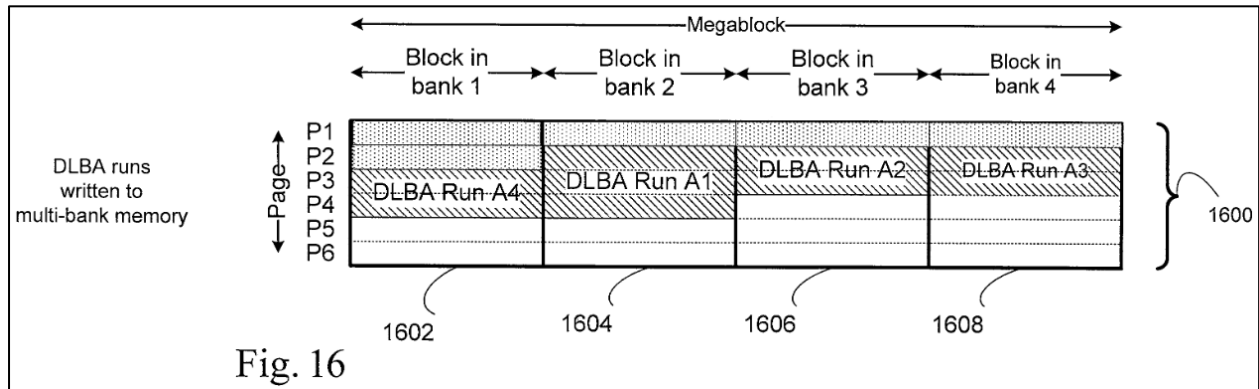


Fig. 13

Sinclair also discloses that its controller identifies the open write megablock [respective subdivision] dependent on which one of the intermediate storage LBA addresses [which one of the non-overlapping ranges] encompasses the host LBA addresses [address accompanying the one of the write requests]. EX1003, ¶101.

For instance, with reference to Figures 16 and 17, Sinclair discloses that “the host

LBA addresses are translated to an intermediate storage LBA address, also referred to as a DLBA address, in an application run by the controller 108 in the memory system 102.... As shown in FIG. 16, the open write megablock 1600 in a four bank memory with metablocks 1602 each having six metapages (P1-P6) is associated with the LBA addresses for the LBA run 1702 shown in FIG. 17.” EX1004, ¶70.



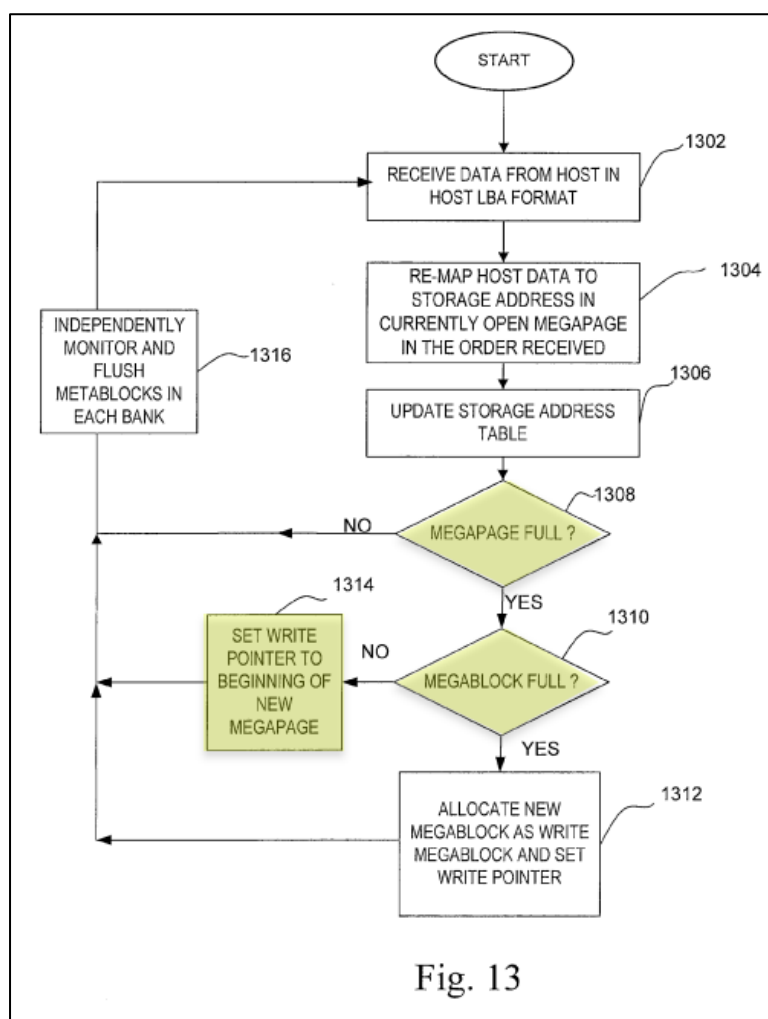
EX1004, Figs. 16-17; ¶¶23, 70-71; EX1003, ¶101. The SAT 1704 maps each LBA run containing valid data to the associated DLBA runs where the LBA entry 1706 in the SAT 1704 includes the first LBA address in the run, the length of the run and the DLBA address and bank identifier of the first DLBA run mapped to the LBA run 1702. EX1004, ¶71, EX1003, ¶101. The corresponding DLBA entries 1708 include a first DLBA entry 1710 that has the first DLBA address and bank number of the DLBA run and the LBA address offset in the LBA run 1702 the first DLBA address is mapped to which, in the case of the first DLBA entry 1710 will be zero and in all subsequent DLBA entries for a given LBA run 1702 will be non-zero values. *Id.*

The mapping from host LBA addresses is tracked in a “storage address table” (SAT) 1704 that maps each LBA run 1702 containing valid data to the associated DLBA runs and the host data is written to “address locations in the megablock 1600” until the “megablock is fully programmed.” EX1004, ¶72; *id.*, ¶5 (“allocating a megablock of contiguous storage LBA addresses for addressing the data associated with the host LBA addresses.”); EX1003, ¶102.

1[d.ii]

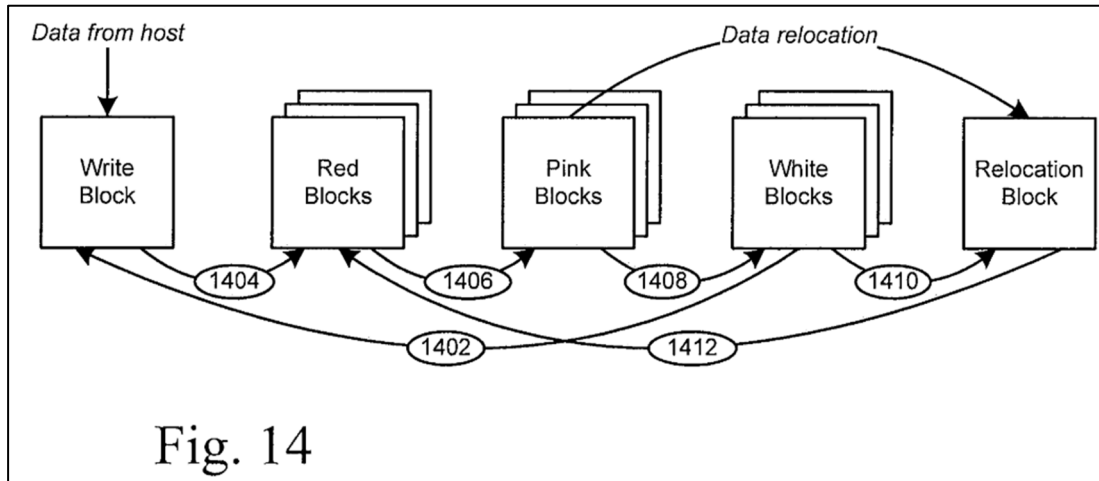
Sinclair discloses that “as the host data is received” [**the data accompanying with the one of the write requests**], it is programmed “by writing the host data to the currently open megapage [**a next available storage location**] in the currently open write megablock [**specific one of the respective subdivisions**].” Ex. 1004,

¶63; EX1003, ¶104. The “host data continues to be re-mapped to contiguous storage addresses in each metapage of the megapage, bank-by-bank, in the order received” until the “current write megablock is fully written.” *Id.* For instance, Figure 13 illustrates the process by which the data is written into the current write megablock with the write pointer being updated to the next available metapage (step 1314) until the megablock is fully programmed (at 1308, 1310, 1312):



Ex. 1004, Fig. 13; *id.*, ¶¶63-64, Fig. 14 (“When the last page in the current write block is filled with valid data, the current write block becomes a red block (at

step 1404) **and a new write block is allocated from a white block list** (at step 1404) to be part of the next megablock 704.”) (emphasis added); EX1003, ¶104.



EX1004, Fig. 14.

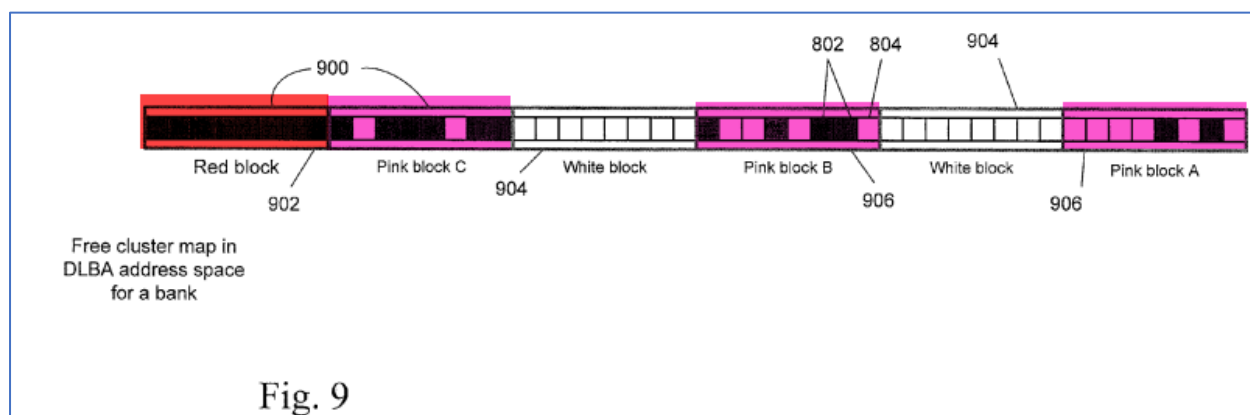
1[d.iii]

Sinclair tracks the extent to which metablocks [**erase blocks**] of a specific megablock [**specific one of the respective subdivisions**] are full, by tracking whether metablocks are “white,” “red,” or “pink” as follows: EX1004, ¶¶57-58.

- Red block (filled with valid data, i.e. completely full): “blocks 900 completely filled with valid data are referred to as red blocks 902.”
- White block (no valid data, i.e., not at all full): “blocks with no valid data, and thus containing only unwritten capacity, are referred to as white blocks 904.”
- Pink block (valid and invalid data, i.e., partially full): “Blocks that have been fully programmed and have both valid 802 and invalid (also

referred to as obsolete) 804 clusters of data are referred to as pink blocks 906.”

Id., Figs. 8-9; Ex. 1003, ¶106;



EX1004, Fig. 9 (red, pink, and white blocks highlighted).

In addition, a “valid page count value” (or cluster count value) is built to represent the count of valid pages (or clusters) of each block, and is “incremented accordingly” for each identified valid data page located in a pink block set. EX1004, ¶812. Thus, the “valid page count value” **information** represents the count of valid pages within a pink block [**representing an extent to which the erase blocks of the specific one of the respective subdivisions are full**]. EX1003, ¶107. This list of red, pink and white blocks and their “valid page count” is maintained by Sinclair, for instance in a block information table (BIT). EX1004, ¶¶82, 67. In association with the write algorithm [**in association with the programming of the data accompanying the one of the write request**], the memory controller increments the red, white, and pink status of the metablocks and the valid page count accordingly.

Id., ¶¶812; EX1003, ¶¶107. The write request also causes the entries of the BIT table to be updated to reflect the data programmed in the white, pink, and red status of the metablocks. EX1004, ¶¶127 (“A BIT page 3002 is the minimum updatable unit of block list information in the BIT”), 136 (“A BIT page 3002 is updated to add or remove entries from the WBL 3008, PBL 3010 and SBL 3012”), Fig. 30.

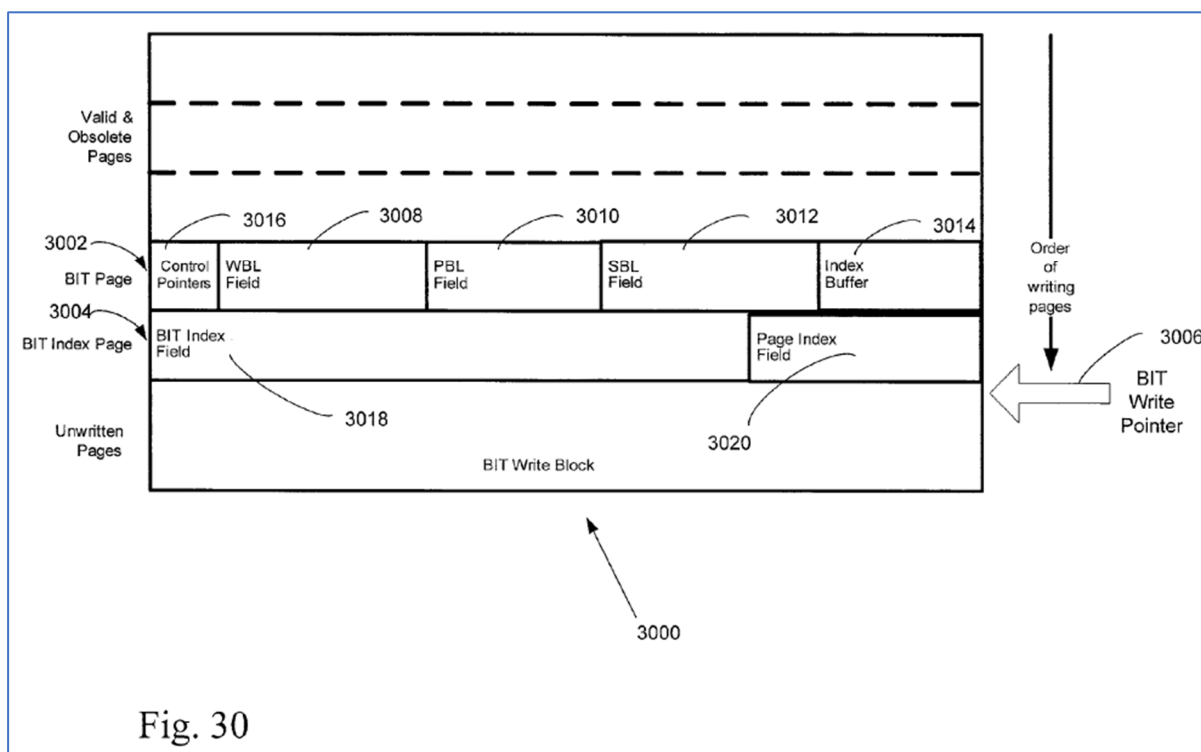


Fig. 30

EX1004, Fig. 30; EX1003, ¶¶107-08.

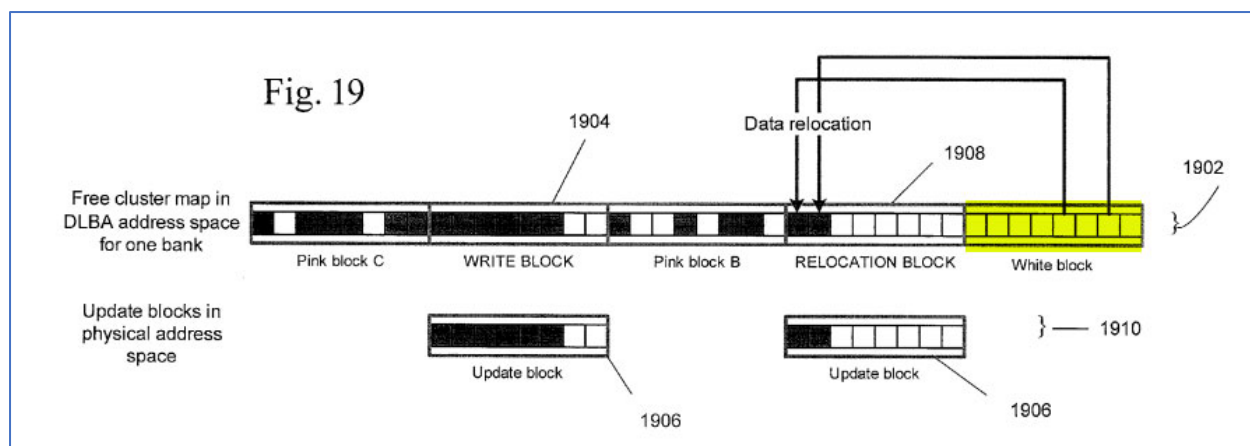
Sinclair also discloses that a flush maintenance operation may be performed during a write operation if it is determined that there are not sufficient white blocks to accommodate the write request. EX1004, ¶67, Fig. 15. Sinclair explains that the flush operation transforms pink blocks into white blocks. EX1004, ¶67, Fig. 15. A POSITA would thus understand Sinclair to disclose that in association with a write

command, the valid page count value [**information**] is updated and if the page count value is equal to the total number of blocks in the metablock, the metablock is set to the full (red) state [**representing an extent to which the erase blocks of the specific one of the respective subdivisions are full**]. EX1003, ¶109.

1[e]

Sinclair discloses that when the number of white blocks in the system is running low, a maintenance flush operation is initiated [**a maintenance request**] to transform pink blocks into white blocks. EX1004, ¶67 (“When the storage address re-mapping algorithm detects a need for more white blocks in the bank, the algorithm initiates a flush operation...”), Fig. 15; EX1003, ¶111.

To fulfill this maintenance request, the memory controller determines and stores the number of pink blocks that need to be relocated [**physically erased**]. EX1004, ¶67 (“In order to flush a pink block, the valid data of a pink block is sequentially relocated in an order of occurrence to a white block that has been designated as a relocation block.”), ¶111 (“in this example, pink blocks selected for flush operations will have 52.4% white capacity”), Fig. 15 (step 1506: “select pink block from pink block list for bank”); EX1003, ¶112. As illustrated in Figure 19, at step 1908 the flush operation creates a white block (highlighted in yellow) by performing data relocation and subsequently physically erasing data from pink blocks:



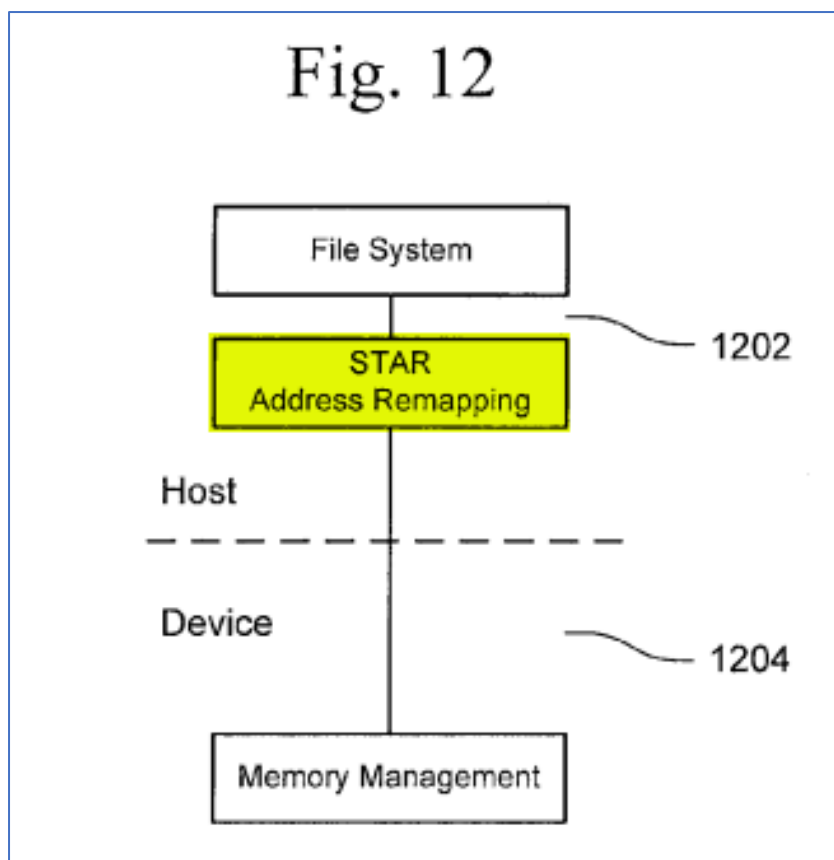
EX1004, ¶76 (“The storage address re-mapping algorithm manages the creation of white blocks by flushing data from pink blocks to a special write block known as the relocation block 1908”); EX1003, ¶112.

For instance, Sinclair discloses that the memory controller may maintain a list of pink blocks where the “amount of valid data” is below a “threshold,” and those pink blocks would then be erased to fulfill the flush operation. EX1004, ¶67 (“a dynamic list of a defined number (e.g. sixteen) or percentage (e.g. 30 percent) of pink blocks currently satisfying the threshold requirement may be maintained and any pink block may be selected from that list for flushing”). A POSITA would understand that each pink block is associated with a memory size such that the storing a listing of pink blocks that are to be flushed discloses **storing a value representing a size of the NAND flash memory that is to be physically erased**. EX1003, ¶113. Indeed, Sinclair discloses that the memory system may provide the

host with “information on the block size of physical blocks in the flash memory.” EX1004, ¶61.

Sinclair also discloses that the flush operation may be initiated by the host [maintenance request issued by the host]. EX1003, ¶114. For instance, Sinclair discloses that its flush operation performed in fulfilling a maintenance request may be initiated by a host activity, when a host “flush cache” command occurs, or in response to host commands. EX1004, ¶112. In some embodiments, “the flush operation may be utilized in a foreground operation that is active when the host is writing data.” *Id.* A POSITA would understand these disclosures to teach that during foreground operation, the flush operation [maintenance request] is initiated by a “host activity” such as an activity which requires more white blocks to be created, or at minimum suggest that the maintenance request can be issued by the host. EX1003, ¶114.

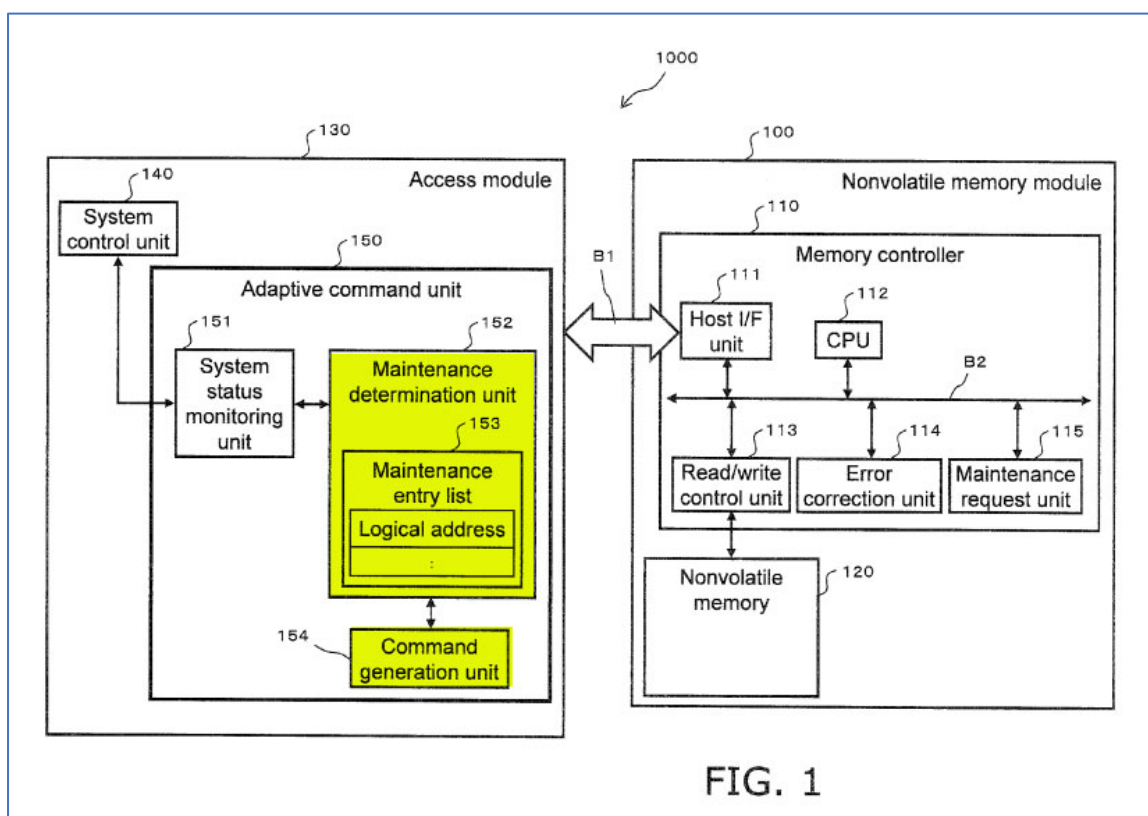
In addition, Sinclair explains that “when the storage address re-mapping algorithm detects a need for more white blocks in the bank, the algorithm initiates a flush operation.” EX1004, ¶65. In some embodiments, “re-mapping (STAR) functionality” may be contained “within the host” (EX1004, ¶68), as shown below:



EX1004, Fig. 12; EX1003, ¶115. Sinclair discloses that in the host-issued flush operation embodiment, the host and memory system may “exchange information on the block size of physical blocks” and their status (EX1004, ¶61), such that the “**host** [] may select pink blocks” for the flush operation (EX1004, ¶68). A POSITA would understand these disclosures to teach that the host may issue the flush operation [**maintenance request**], or suggest that the maintenance request can be issued by the host to the memory controller in various embodiments. EX1003, ¶116.

To the extent Sinclair does not teach the maintenance flush operation may be initiated by the host, it would have been obvious in further view of Yamada. EX1003, ¶117. Yamada discloses moving maintenance operations from the storage

controller to the host. EX1005, ¶10 (desirable to perform “maintenance” on the memory controller so that “device conditions” may be considered). For instance, Yamada discloses that a maintenance determination unit in the access module 130 may include a “command generation unit” 154 that “generated a maintenance command for performing maintenance for the address of the nonvolatile memory device selected by the maintenance determination unit” 152 *Id.*, ¶22; EX1003, ¶117. Figure 1 illustrates the access module [host] and its “maintenance determination unit” and “command generation unit” that “generates a maintenance command for causing maintenance for data retention, such as refresh, to be performed for an address stored in the maintenance entry list 153”:



EX1005, Fig. 1 (annotated), ¶¶89, 90 (“The maintenance command includes a logical address of a block that needs maintenance (hereafter referred to as a “maintenance-needed block”) stored in the maintenance entry list 153.”); EX1003, ¶117. In the combined Sinclair-Yamada system, the host would issue the flush operation [**maintenance request issued by the host**] for the pink blocks that need to be relocated [**physically erased**]. EX1003, ¶118.

Motivation to Combine. The prior art includes express teachings, suggestions and motivations to combine the teachings of Sinclair and Yamada such that the flush operation is issued by the host. EX1003, ¶¶79-86. For instance, as discussed above, Sinclair at least suggests that the host may select pink blocks and issue a maintenance operation to the storage device. EX1004, ¶¶61, 68, Fig. 12; EX1003, ¶116. Further, Yamada discloses that “maintenance” requests “require[] a certain amount of power consumption” and that “conventional techniques fail to consider the device conditions” when maintenance is performed. EX1005, ¶10; EX1003, ¶81. Yamada teaches that to “solve this problem” it is desirable to have the host issue management request, such that the host may consider device conditions when scheduling those commands. EX1005, ¶¶11; EX1003, ¶81.

Such cooperation of memory maintenance between a controller and a host was known in the art at the time, and other references also teach that a storage system may be improved by having the host issue maintenance requests. EX1003, ¶83. For

instance, Jung discloses a system for “cooperative memory management” and teaches that moving some or all management functions to the host can increase “execution speed” and avoid the controller becoming “overloaded.” EX1007, ¶¶3-5, 7, 41; EX1003, ¶83. Similarly, Son explains that a storage device that performs “management operations,” such as “address remapping” or “garbage collection,” can be improved by having the host issue the management command so that it may “incorporate file system information” and “relieve the SSD controller and firmware of these responsibilities.” EX1006, ¶¶13, 63; EX1003, ¶83. Thus, Son discloses that a device like Sinclair can be improved by configuring a host to issue maintenance requests, thereby “improv[ing] the performance of the SSD array.” EX1006, ¶¶17, 21; EX1003, ¶83. Thus, *Sinclair*, *Yamada*, *Jung*, and *Son* all provide an express teaching, suggestion, and motivation for modifying Sinclair so that its flush operation is issued by the host. EX1003, ¶83.

A POSITA would also have had a reasonable expectation of success in combining Sinclair and Yamada because it represents a working implementation of issuing maintenance requests, such as Sinclair’s flush operation, from a host device. EX1003, ¶¶84-86. Indeed, Sinclair already teaches and suggests that that its disclosures encompass performing maintenance operations on the host or the memory controller. EX1004, Fig. 11 (address remapping on the memory controller), Fig. 12 (moved to the host); EX1003, ¶86.

1[f]

As discussed, Sinclair discloses maintaining and updating a list of red, pink and white blocks for a megablock and their valid page count, which would be **the information representing an extent to which the erase blocks of the specific one of the respective subdivisions are full**. 1[d.iii]. Further, Sinclair stores a listing of pink blocks that are to be relocated [**storing a value representing a size of the NAND flash memory that is to be physically erased**]. *see* 1[e]. The only additional limitation in this element is that the updated “information” and the “stored value” are provided to the host.

Sinclair discloses that in the host-issued flush operation embodiment, the host and memory system “exchange information on the block size of physical blocks” and their status (EX1004, ¶61), such that the “host [] may select pink blocks” for the flush operation (EX1004, ¶68); *see* 1[e]. A POSITA would understand these disclosures to teach that for the host to perform the flush operation, it must be provided with the list of red, pink, and white blocks for a megablock and their valid page count [**information**] and the list of pink blocks that are to be relocated [**stored value**]. EX1003, ¶120.

To the extent Sinclair does not teach or suggest providing the host the information and the stored value in response to a host query, the limitation is at least obvious in view of Yamada. EX1003, ¶121. As discussed in 1[e], Yamada discloses

a maintenance determination unit with a maintenance entry list to perform maintenance operations. EX1005, Fig. 1, ¶¶22, 89-90. Yamada teaches that the controller maintains information about the valid block count and that this information can be retrieved via host queries. EX1005, ¶¶32, 96; EX1003, ¶122. Yamada also teaches that the host can issue maintenance operations. EX1005, Fig. 1, ¶¶22, 89-90; EX1003, ¶122. Thus, in the combined Sinclair-Yamada system, it would be obvious for a POSITA to combine these techniques, e.g. allowing the host to send a maintenance command to which the controller responds with the number of valid blocks moved by the maintenance command. EX1003, ¶122.

1[g]

Sinclair discloses that “the selection [of] a pink block for flushing” may be made by selecting a group of pink blocks that are “associated with less than some threshold amount of valid data.” EX1004, ¶69. A POSITA would understand that the “amount of valid data” associated with a pink block is **metadata** and also that pink blocks are associated with a megablock [**one of the respective subdivisions**]. EX1003, ¶124. The amount of valid data also represents a **characteristic of the data**, for example that it is valid. EX1003, ¶124.

In addition, Sinclair discloses that the “selection of pink blocks” for the flushing operation “could be based on an assumption that **data that has survived the longest** in the memory is least likely to be deleted.” EX1004, ¶69. A POSITA

would understand this disclosure to teach that Sinclair maintains and updates how long the data in a megablock has survived [**metadata**]. EX1003, ¶125. The time the data has survived represents the age of the data since it was last written [**a characteristic of the data**]. EX1003, ¶125.

As another example, Sinclair discloses that the controller determines and maintains whether the data in the megablock is “cold” or “hot” data [**metadata**]. EX1004, ¶69. In this manner, “[t]he flush operation relocates relatively ‘cold’ data from a block from which ‘hot’ data has been made obsolete to a relocation block containing similar relatively cold data.” *Id.*, ¶80 A POSITA would understand that the term “hot” and “cold” data refers to how recently or frequently the data has been accessed [**represents a characteristic of the data**], with hot data being more recently or frequently accessed than cold data. EX1003, ¶126.

1[h]

Sinclair determines that a list of pink blocks should be flushed based on the amount of valid data in the pink blocks [**condition**]. EX1004, ¶68; EX1003, ¶128. For instance, the amount of valid data may be compared to the amount of data in other pink blocks to determine which pink block has the “least amount of valid data” [**comparison based on the metadata**]. *Id.* Sinclair also determines the list of pink blocks by identifying “pink blocks that fall within the threshold requirement” (*e.g.*, “less than some threshold amount of valid data”) [**comparison based on the**

metadata to detect a condition]. *Id*; EX1003, ¶128. Thus, Sinclair discloses performing a comparison based on the amount of valid data [**metadata**] to detect the pink blocks to be flushed [**condition**]. *Id*.

As another example, Sinclair discloses that the storage controller may create “separate populations of relatively hot and relatively cold blocks” and then selects “hot block[s] containing the least amount of data” or “‘cold’ data from a block from which “hot” data has been made obsolete” to be flushed. EX1004, ¶80. A POSITA would understand that determining whether data is hot or cold involves comparing how frequently the data is accessed to some threshold [**comparison based on the metadata**]. EX1003, ¶129; EX1008 at 5:13-23 (“If the data access frequency of the migration unit is greater than the threshold, it indicates that the migration unit is frequently accessed and has become ‘hot data’; if the data access frequency of the migration unit is less than the threshold, it indicates that the migration unit is infrequently accessed and is ‘cold data’.”). Thus, Sinclair also discloses performing a comparison based on access frequency [**metadata**] to detect data that should be flushed [a **condition**]. EX1003, ¶129.

1[i]

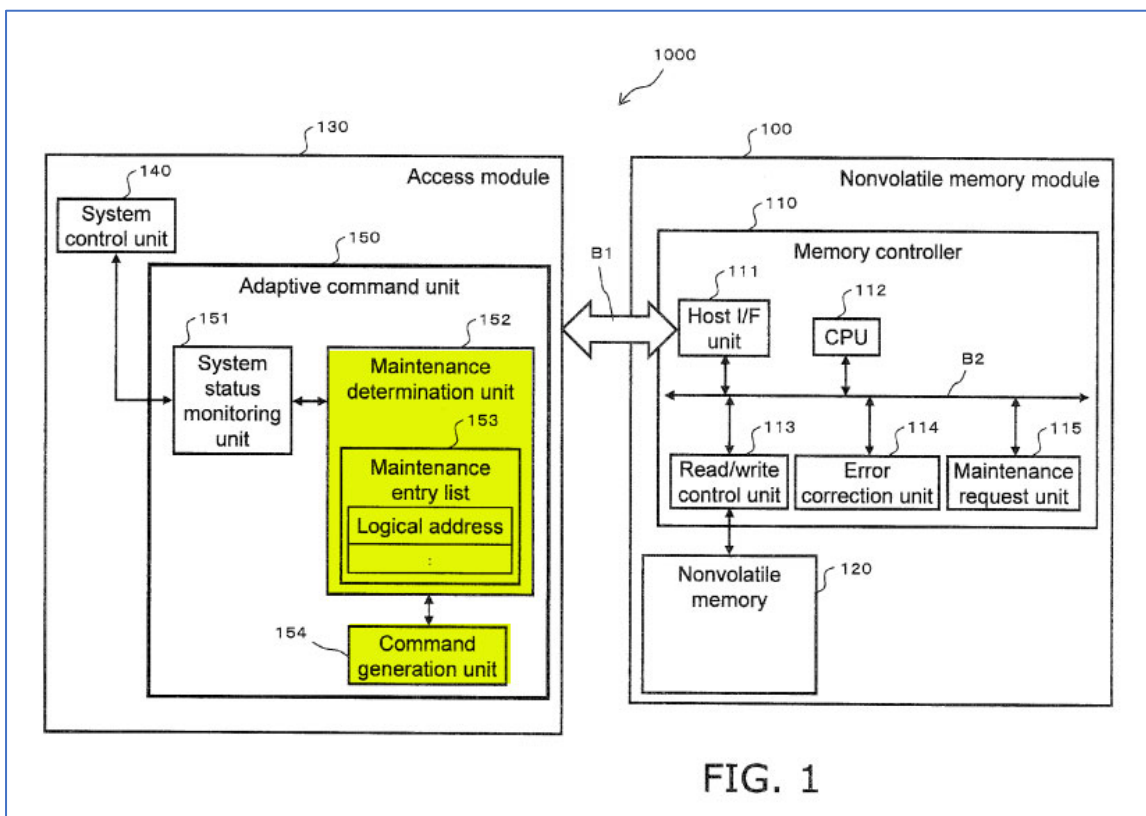
Sinclair discloses **detection** that a pink block should be added to a list of blocks that should be flushed based on the amount of valid data in the pink block [**condition**]. *see* 1[h]. Sinclair also discloses detecting “relatively hot and relatively

cold blocks” [**condition**] to be added to a list of blocks to be flushed. *see* 1[h]. The list of blocks is **store[d] information responsive to the detection of the condition** because blocks are added to the stored list based on the detection of the aforementioned conditions. EX1003, ¶131.

Sinclair also discloses that this **information** is accessible to the host. EX1003, ¶132. For instance, Sinclair discloses that “[a] subset of the pink blocks at or below the threshold amount of valid data may be **maintained in a list from which the host or memory system may select pink blocks**” [**stored information...accessible to the host**]. EX1004, ¶68 (emphasis added). A POSITA would understand that this detection of pink blocks that need to be flushed [**information**] is accessible to the host and indicates that the **condition** (*e.g.*, a pink block falling below a threshold amount) was **detected**. EX1003, ¶132. At minimum, a POSITA would understand that this disclosure suggests that the host may access the list of pink blocks that are below the threshold so that it may then select a pink block to flush. EX1003, ¶132.

That the information is accessible to the host is also obvious in view of Yamada. EX1003, ¶133. Sinclair already discloses that in the host-issued flush operation embodiment, the host and memory system may “exchange information on the block size of physical blocks” and their status (EX1004, ¶61), such that the “host [] may select pink blocks” for the flush operation (EX1004, ¶68). *See also* 1[e];

EX1003, ¶133. Yamada discloses moving maintenance operations to a maintenance determination unit on the host:

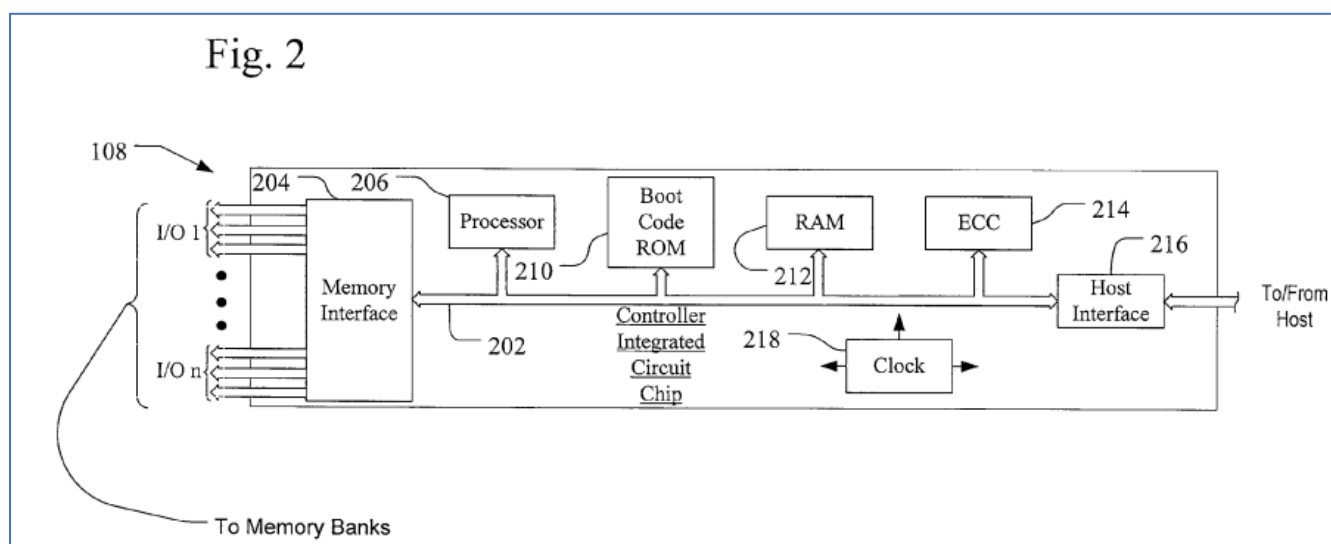


EX1005, Fig. 1, ¶¶10, 89-90; EX1003, ¶133. Yamada’s “maintenance determination unit determines that the maintenance is to be performed when determining that the data stored in the nonvolatile memory needs maintenance.” EX1005, ¶18; EX1003, ¶134. Once the maintenance condition is detected, the “the maintenance determination unit **stores** at least one address of the nonvolatile memory that needs maintenance” [**stored information**] in a maintenance entry list. EX1005, ¶¶23, 89-91 (emphasis added); EX1003, ¶134. A POSITA would understand these disclosures to teach that in the combined Sinclair-Yamada system,

the host issues the flush operation for the pink blocks that need maintenance, and would thus need to access the list of pink blocks that need to be flushed [**stored information**] in order for the host to successfully perform this task. EX1003, ¶134.

1[j]

Sinclair discloses a “memory controller” having logic. See 1[b]; EX1004, ¶44, Figs. 1-2; EX1003, ¶136. Figure 2 illustrates the memory controller 108, which a POSITA would understand includes processing circuitry (e.g. processor) and software instructions that are executed by the processor to perform the claimed functionality:



EX1004, Fig. 2; EX1003, ¶136.

2. Claim 2

2[a]

Sinclair discloses that the “selection of pink blocks” for the flushing operation “could be based on an assumption that data that has survived the longest in the

memory is least likely to be deleted.” EX1004, ¶69; *see* 1[g]. A POSITA would understand this disclosure to teach that Sinclair maintains and updates how long the data in a megablock has survived [**information representing an age-since-write of data**]. EX1003, ¶138.

As another example, Sinclair discloses that the controller determines and maintains whether the data in the megablock is “cold” or “hot” [**metadata**]. EX1004, ¶69; EX1003, ¶139. The flush operation then relocates and groups cold data together. *Id.* A POSITA would understand that “hot” and “cold” data refers to how recently or frequently the data has been accessed [**information representing an age-since-write**], with hot data being more recently written to than cold data which has not been written in a long time. EX1003, ¶139.

To the extent Sinclair does not teach age-since-write information, the limitation is at least obvious in view of Yamada. EX1003, ¶140. As discussed in 1[g], Sinclair discloses storing metadata representing a characteristic of the data such as “hot” and “cold” data contained in megablocks and metablocks, to be used by its maintenance algorithm. EX1004, ¶69. A POSITA would understand these disclosures to teach that the “hot” and “cold” data information may be used during maintenance to perform the “wear leveling of the nonvolatile memory” disclosed in Yamada, which a POSITA understands utilizes the stored age-since-write information. EX1003, ¶140; EX1005, ¶30, 33.

2[b]

Sinclair discloses the identification of a condition where “relatively ‘cold’ data” needs to be flushed. EX1004, ¶80; *see* 1[h]. This limitation further requires that in response to this condition, the controller controls copying the data associated with an age-since-write greater than a threshold, to a new physical storage location in the NAND flash memory. Sinclair discloses this limitation. EX1003, ¶142.

Sinclair discloses that “[t]he flush operation relocates relatively ‘cold’ data from a block from which ‘hot’ data has been made obsolete to a relocation block containing similar relatively cold data.” EX1004, ¶80. Thus, Sinclair relocates data **[copying of data]** from a selected pink block that has been marked to be flushed due to being “cold” **[associated with an age-since-write greater than a threshold]** to the relocation block **[to a new physical storage location in the NAND flash memory]**. EX1003, ¶143; EX1004, ¶80; EX1008 at 5:13-23 (cold data is infrequently accessed data).

3. Claim 3

3[a]

Sinclair discloses fulfilling maintenance requests by the host, such as in order to perform a flush operation and its address re-mapping. *see* 1[e]. This limitation further requires that the maintenance request is an erase request. Sinclair discloses this limitation. EX1003, ¶144.

For instance, Sinclair discloses to fulfill this maintenance request, the memory controller relocates [erase] valid data from a pink block to a relocation white block. EX1004, ¶67 (“In order to flush a pink block, the valid data of a pink block is sequentially relocated in an order of occurrence to a white block that has been designated as a relocation block.”), ¶111 (“in this example, pink blocks selected for flush operations will have 52.4% white capacity”), Fig. 15 (step 1506: “select pink block from pink block list for bank”); EX1003, ¶145. After relocation, the pink block is erased to become a white block. EX1004, ¶58 (“Blocks ... with no valid data, and thus containing only unwritten capacity... will need to be erased...”). A POSITA would therefore understand that Sinclair’s maintenance request is an erase request. EX1003, ¶145.

3[b]

Sinclair discloses this limitation. EX1003, ¶147. For instance, Sinclair discloses that “[i]n order to flush a pink block, the valid data of a pink block is sequentially relocated in an order of occurrence to a white block that has been designated as a relocation block.” EX1004, ¶67. To perform this request, Sinclair’s re-mapping algorithm relocates data by copying the data [copy request] from a selected pink block [directed to the specific one of the respective subdivisions] that has been marked to be flushed to the relocation white block. EX1004, ¶67 (“valid data is copied from the selected pink block in an order of occurrence in the

pink block to contiguous locations in the relocation block.”); EX1003, ¶147. As discussed in element 1[f], Sinclair or Sinclair combined with Yamada issues the maintenance flush request that causes data to be copied during a flush. *Id.*

3[c]

Sinclair discloses this limitation. EX1003, ¶149. For instance, Sinclair discloses that the flush operation which causes relocation of data [**the copy request**] is performed by the controller—“the write algorithm [is] managed by the controller 108” (EX1004, ¶71), the “storage address re-mapping... is executed by an application run by the controller 108,” (*id.*, ¶ 73) and the “block flush process... [is] executed by the controller 108.” (*id.*, ¶¶77, 84); EX1003, ¶149.

3[d]

Sinclair discloses this limitation. EX1003, ¶151. For instance, Sinclair discloses to fulfill this maintenance request, the memory controller relocates [**erase**] valid data from a pink block [**a source location**] to a relocation white block. EX1004, ¶67 (“In order to flush a pink block, the valid data of a pink block is sequentially relocated in an order of occurrence to a white block that has been designated as a relocation block.”), ¶111 (“in this example, pink blocks selected for flush operations will have 52.4% white capacity”), Fig. 15 (step 1506: “select pink block from pink block list for bank”). A POSITA would therefore understand that Sinclair’s maintenance request is to erase a source location. EX1003, ¶151.

3[e]

Sinclair discloses a copy request and an erase request received by the memory controller. 3[a]-[d]. This limitation further requires the copy and erase requests are received **at different times**. Sinclair discloses this limitation. EX1003, ¶152.

For instance, Sinclair discloses that in some embodiments, its valid data “may consists of obsolete data that will need to be erased upon allocation if the memory system 102 employs an “erase before use” type of procedure.” EX1004, ¶58. That is, in such situations, valid data can be “later made obsolete.” *Id.*, ¶65. Thus, at a later time [**different times**] the “accumulation of obsolete data... may be independently flushed to unrelated relocation blocks.” *Id.*, ¶72; EX1003, ¶153.

4. Claim 4

4[a]

Sinclair discloses that the memory controller comprises logic to receive a query from the host. *see* 1[f]. For instance, Sinclair discloses that the host may select pink blocks from the list of pink blocks that meet the threshold requirements. EX1004, ¶68 (“A subset of the pink blocks at or below the threshold amount of valid data may be maintained in a list **from which the host** or memory system **may select pink blocks**.”). A POSITA would understand that Sinclair’s disclosure that the host may select pink blocks from the list for flushing teaches that the host may query the memory controller for the list of pink blocks. EX1003, ¶155. At minimum, a POSITA would understand that this disclosure suggests that the host may query the

list of pink blocks that are below the threshold so that the host may then select a pink block to flush. EX1003, ¶155.

To the extent Sinclair does not teach the query is received from the host, it would have been obvious in further view of Yamada. EX1003, ¶156. *see* 1[e]. Yamada discloses moving maintenance operations from the storage controller to the host. EX1005, ¶10 (desirable to perform “maintenance” on the memory controller so that “device conditions” may be considered). For instance, Yamada discloses that a maintenance determination unit in the access module may include a “command generation unit” that “generated a maintenance command for performing maintenance for the address of the nonvolatile memory device selected by the maintenance determination unit.” *Id.*, ¶22, Fig. 1. In the combined Sinclair-Yamada system, the host would issue the query to the controller [**receive a query from the host**] for the metadata of the list of pink blocks that meet the threshold requirements. EX1003, ¶156.

4[b]

Sinclair discloses that the comparison is between the metadata and a threshold. *see* 1[i]. For instance, Sinclair discloses that the host may select pink blocks from the list of pink blocks that meet the threshold requirements. EX1004, ¶68 (“A subset of the pink blocks at or below the threshold amount of valid data may

be maintained in a list **from which the host or memory system may select pink blocks.**”); EX1003, ¶158.

4[c]

As discussed above, Sinclair teaches that the host may query the list of pink blocks that are below the threshold so that the host can select a pink block to flush. 4[a]. The only additional limitation is to provide to the host with at least one address associated with the specific one of the respective subdivisions.

A POSITA would understand that Sinclair teaches or suggests that in order for the host to select a pink block to flush, the query returns the addresses for the pink blocks in the list. EX1003, ¶160. Indeed, Sinclair discloses a “block information table (BIT)” that contains “lists of block addresses for,” among others, “pink” blocks and that the BIT is “maintained by the storage address re-mapping function.” EX1004, ¶82. Sinclair also discloses that in some embodiments the “storage address re-mapping functionality is contained totally within the host.” EX1004, ¶59, Fig. 12. Thus, a POSITA would understand that where the storage address re-mapping functionality is contained within the host, the host would query the memory controller for the list of block addresses for the pink blocks. EX1003, ¶160.

To the extent Sinclair does not teach the query returns the addresses for the pink blocks in the list, it would have been obvious in further view of Yamada.

EX1003, ¶161. Yamada discloses moving maintenance operations from the storage controller to the host. EX1005, ¶10 (desirable to perform “maintenance” on the memory controller so that “device conditions” may be considered). Yamada discloses that a maintenance determination unit in the access module may include a “command generation unit” that “generated a maintenance command for performing maintenance for the address of the nonvolatile memory device selected by the maintenance determination unit.” *Id.*, ¶22; EX1003, ¶161. When a maintenance request is received, the “logical address of the maintenance-needed block, or a logical address of the start of the maintenance-needed block and the size of the block, or a logical address of the start of the maintenance-needed block and a logical address of the end of the maintenance-needed block” is included information for “identifying the maintenance-needed block and information about the processing time for the maintenance that will be required by the nonvolatile memory module.” *Id.*, ¶101; EX1003, ¶161.

5. Claim 6

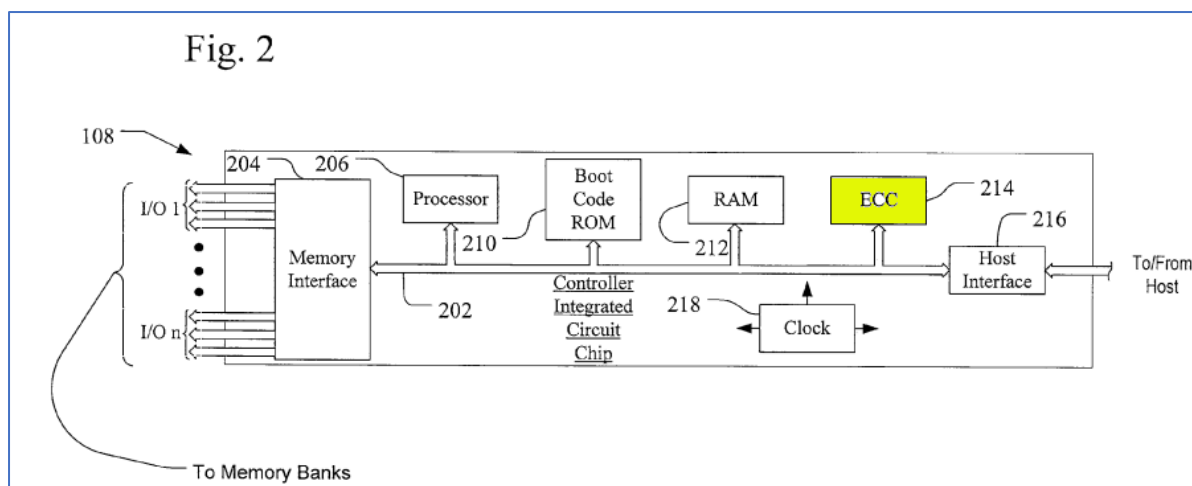
Sinclair teaches that in order for the host to select a pink block to flush, the query returns the addresses for the pink blocks in the list. *See* 4[c]. The only additional limitation is that the address comprises a logical address corresponding to the one of the non-overlapping ranges.

Sinclair teaches that the addresses for the pink blocks in the list is a logical address corresponding to one of its metablocks. EX1003, ¶163. Indeed, Sinclair discloses a “block information table (BIT)” that contains “lists of block addresses for,” among others, “pink” blocks and that the BIT is “maintained by the storage address re-mapping function.” EX1004, ¶82. Each pink block address is a logical address mapped to a megablock comprising metablocks. *Id.* ¶¶57-58, Figs. 8-9. EX1003, ¶163.

6. Claim 9

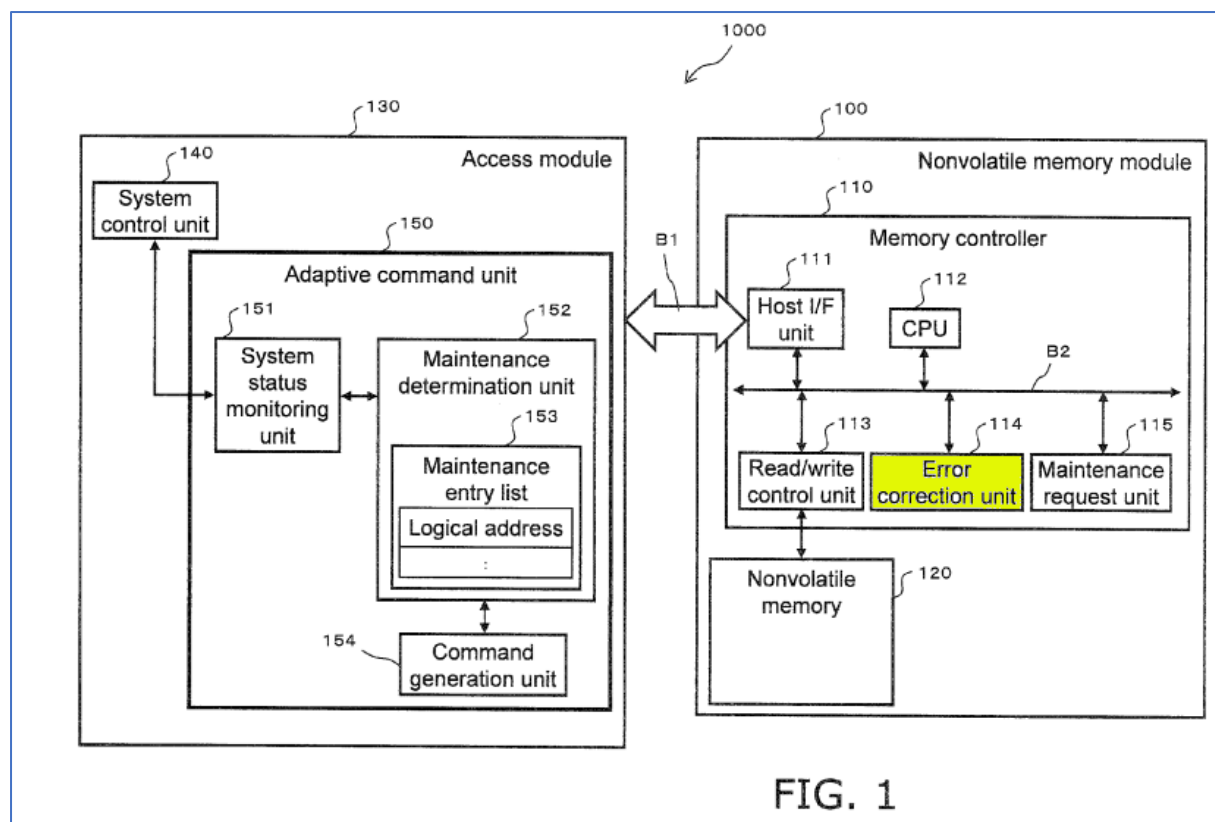
9[a]

A POSITA would understand Sinclair teaches, or at least suggests, that requiring maintenance such that data needs to be flushed (needs maintenance) is a function of a data error condition. EX1003, ¶165; EX1004, ¶72.(“...as [blocks of megablocks] each become pink blocks due to the accumulation of obsolete data, [they] may be independently flushed to unrelated relocation blocks.”). In addition, Sinclair discloses that the memory controller includes an “error correction unit” (ECC) that is shown below in yellow:



EX1004, Fig. 2, ¶45. But Sinclair is otherwise silent regarding error correction.

However, a POSITA would have found it obvious for the metadata to be a function of data error in the data being characterized in view of Yamada. EX1003, ¶166. Like Sinclair, Yamada discloses that the memory controller includes an “error correction unit” (yellow):



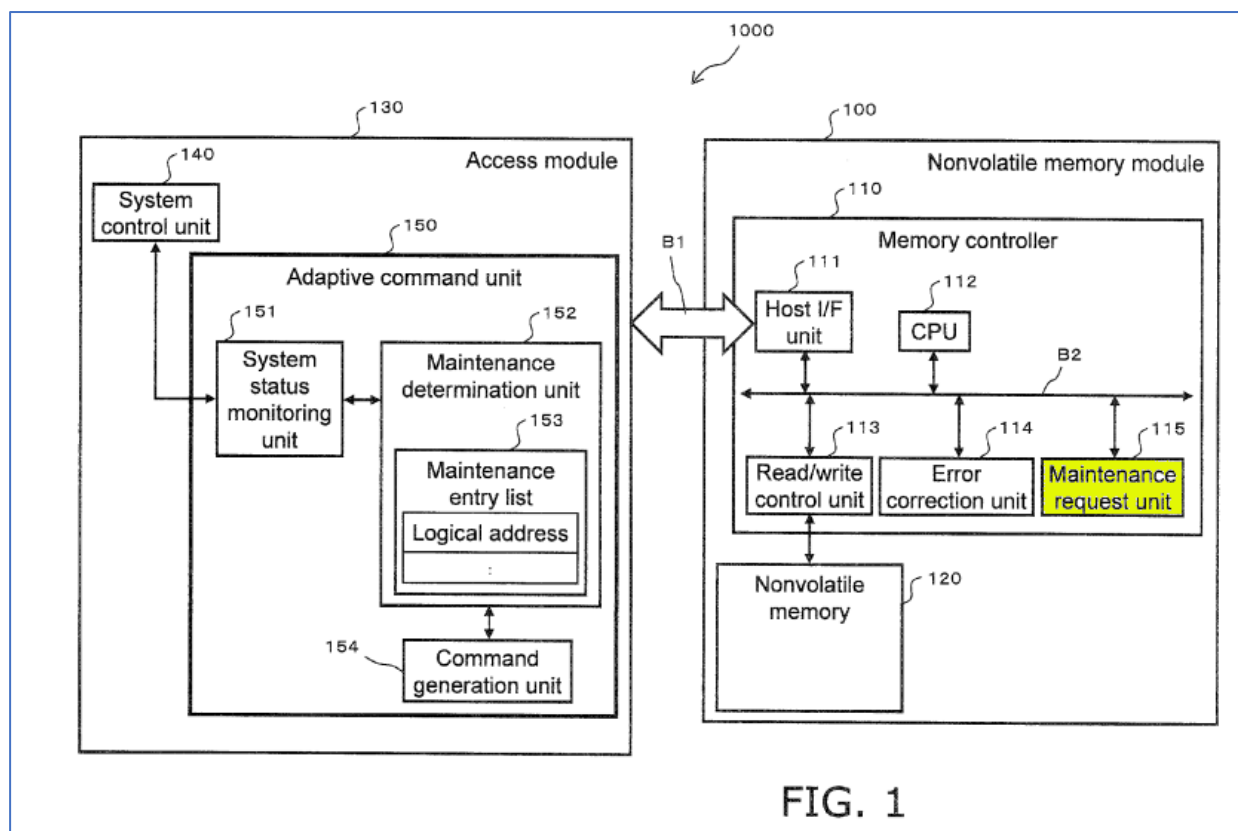
EX1005, Fig. 1, ¶76. Yamada discloses that its “maintenance request unit 1011 determines whether maintenance needs to be performed based on whether the number of errors is greater than or equal to the criterion...” EX1005, ¶188. Yamada discloses that the “error correction unit determines the number of errors included in data read from the nonvolatile memory.” *Id.*, ¶¶32, 96. A POSITA would understand that the number of errors in the data is **metadata that is a function of data error in the data being characterized**. EX1003, ¶166.

Motivation to Combine. Sinclair and Yamada would be combined as discussed in 1[e]; EX1003, ¶¶78-86. Moreover, Yamada discloses that it is important to track the number of errors and issue maintenance requests to “enhance

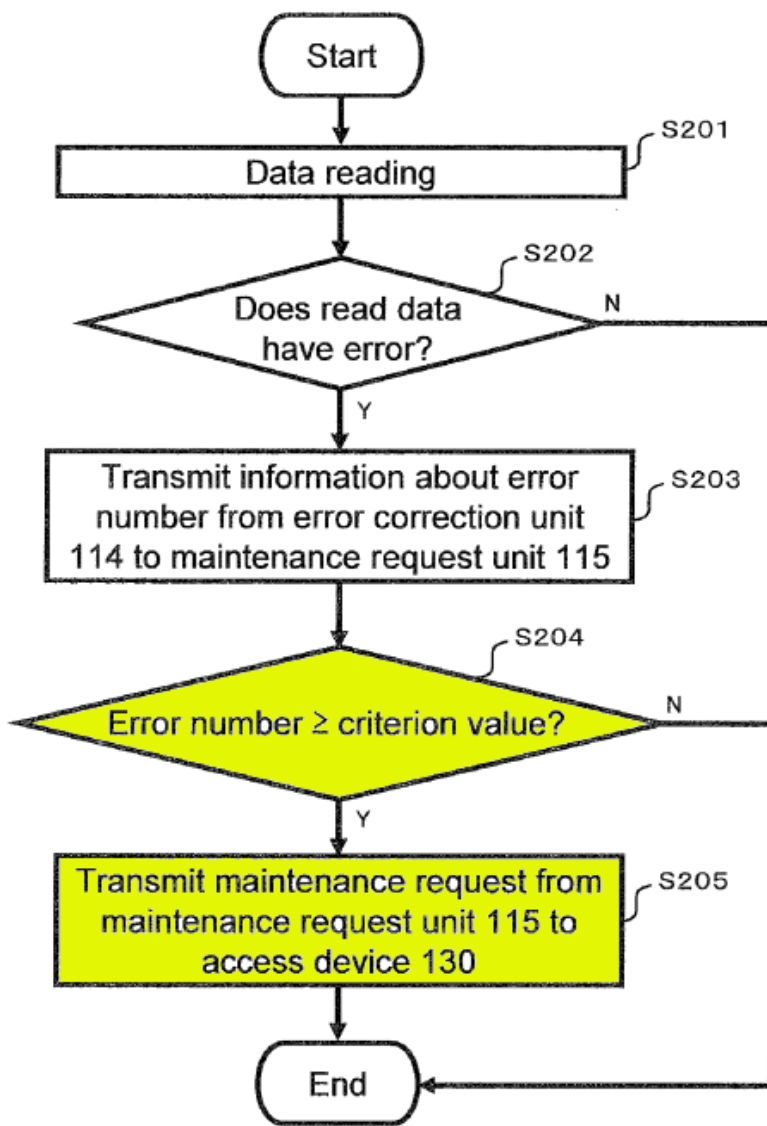
the data integrity of nonvolatile memories.” EX1005, ¶6. While Sinclair discloses that its memory controller, like Yamada, includes an error correct unit, Sinclair is silent regarding its error correction techniques. A POSITA would thus look to other references, like Yamada, that describe these management operations. *Wavetronix LLC v. Smart Microwave Sensors GmbH*, 746 Fed. Appx 984, 987, 991 (Fed. Cir. 2018) (affirming motivation finding where base reference “taught a transmitter that generates a signal” but “failed to disclose its details,” and secondary reference “discloses the details.”); EX1003, ¶¶82-83.

9[b]

Yamada discloses detecting the number of errors is greater than a threshold value [**the condition**]. EX1003, ¶168. For instance, the controller includes a “maintenance request unit” that “determines whether the read area needs maintenance based on the received information about the number of errors.”



EX1005, Fig. 1, ¶¶99; EX1003, ¶168. “When the number of errors is greater than or equal to the criterion value, the maintenance request unit 115 transmits a maintenance request to the access module,” as shown in steps s204-s205 of Figure 2:



EX1005, Fig. 2, ¶¶99; EX1003, ¶168. The maintenance request sent by the controller to the access module in response to detecting the condition involved moving data in the block to a new physical location in the NAND flash memory [**moving data...**]. EX1003, ¶169. Yamada explains that the controller “performs maintenance, such as refresh of data stored at a designated address (logical address)” (EX1005, ¶¶109-110). A POSITA would understand that a refresh or wear leveling command performs

data rewriting by moving data to in the block to a new physical location in the NAND flash memory. EX1003, ¶170; EX1005, ¶7 (“data refreshing (data rewriting)”). For instance, a POSITA would understand that the process of wear leveling typically involves moving cold data from a lower wear count block into a worn block, so that the lower wear count block can be used by hot data. EX1006, Fig. 7 (step 710-730); EX1003, ¶170.

As discussed above, a POSITA would have been motivated to combine Yamada with Sinclair in this manner to enhance data integrity and because it reflects the application of a known technique. 9[a]; EX1003, ¶84.

7. Claim 10

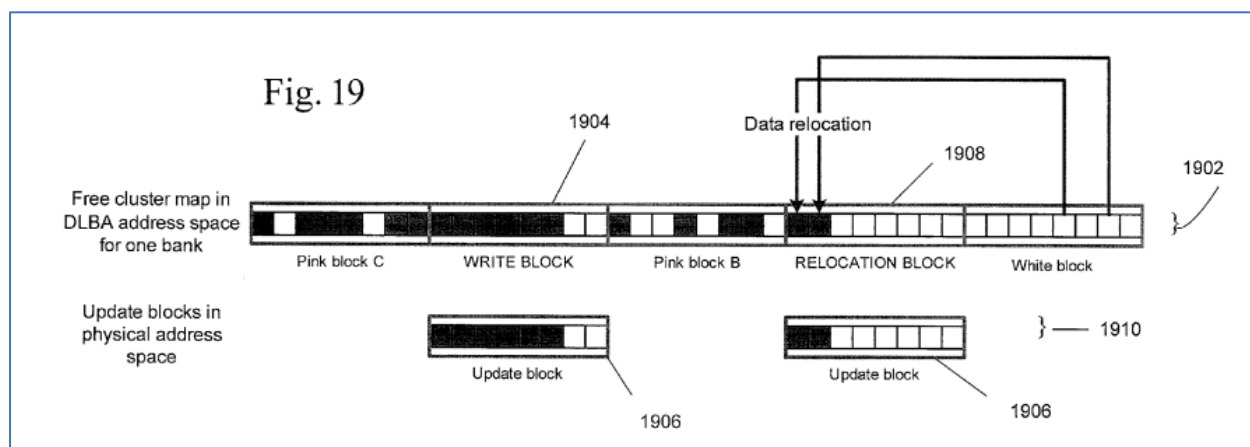
10[a]

As discussed above, Sinclair discloses that the metadata may be whether the block data is hot or cold depending on its access frequency. *see* 1[h]; EX1004, ¶69. A POSITA would thus understand that metadata indicating whether the data is “hot” and “cold” data refers to how likely it is that the data will be changed after it is written and that determining whether data is hot or cold typically involves comparing how frequently the data is accessed [*i.e.*, **data read frequency**] to some threshold. EX1003, ¶172; EX1008 at 5:13-23.

10[b]

Sinclair discloses a controller performing a comparison based on the metadata to detect a condition, namely whether a block should be flushed. *see* 1[h]. The

condition corresponds to a determination that data should be flushed from its existing block to a relocation block [data should be moved to a new physical location in the NAND flash memory]. EX1003, ¶174. For instance, as illustrated in step 1908 of Figure 19, the flush operation performs data relocation by moving data in an existing block to a relocation block:



EX1004, ¶76; EX1003, ¶174.

8. Claim 11

11[a]

As discussed above, Sinclair discloses that the metadata may be whether the block data is hot or cold. *see* 1[h]; EX1004, ¶69. A POSITA would understand that the term “hot” and “cold” data refers to how likely it is that the data will be changed after it is written and that determining whether data is hot or cold typically involves comparing how frequently the data is written [data rewrite frequency] to some threshold. EX1003, ¶176. A POSITA would thus understand that metadata

indicating whether the data is hot or cold represents a data rewrite frequency associated with the data being characterized. EX1003, ¶176.

11[b]

Sinclair discloses performing a comparison based on the metadata to detect a condition, namely whether a block should be flushed. *see* 1[h]. The condition corresponds to a determination that data should be flushed from its existing block to a relocation block [**data should be moved...**]. *see* 10[b]; EX1003, ¶178.

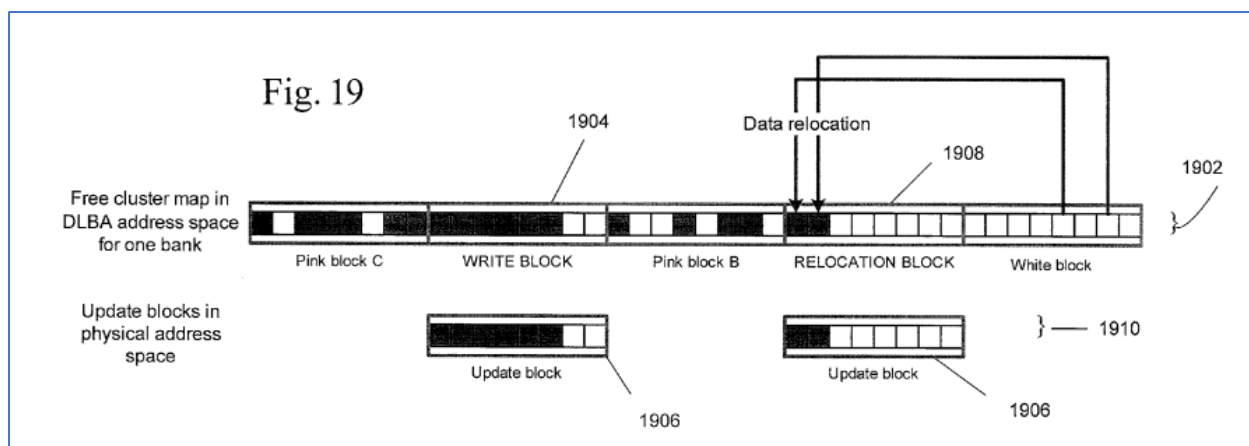
9. Claim 12

12[a]

Sinclair discloses that the “selection of pink blocks” for the flushing operation “could be based on an assumption that **data that has survived the longest** in the memory is least likely to be deleted.” EX1004, ¶69; *see* 1[g]. A POSITA would understand this disclosure to teach that Sinclair maintains and updates how long the data in a megablock has survived [**metadata comprises information representing an age-since-write...**]. EX1003, ¶180.

12[b]

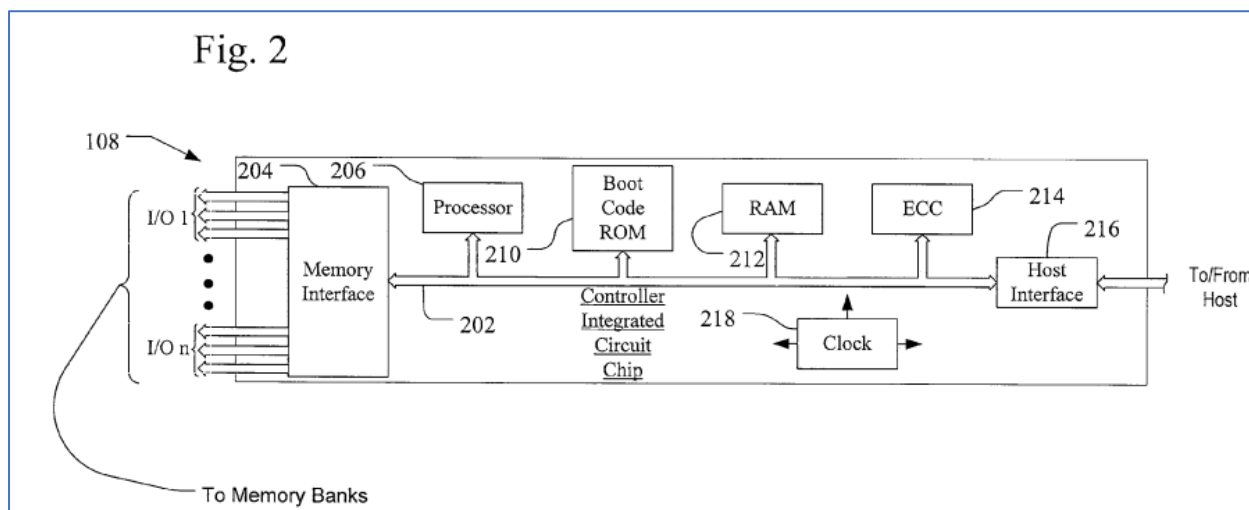
As illustrated in step 1908 of Figure 19, in response to detecting that data should be flushed, the flush operation performs data relocation by **copying** data in an existing block to a relocation block in the NAND flash memory:



EX1004, ¶76; *see* 10[b]; EX1003, ¶182.

10. Claim 13

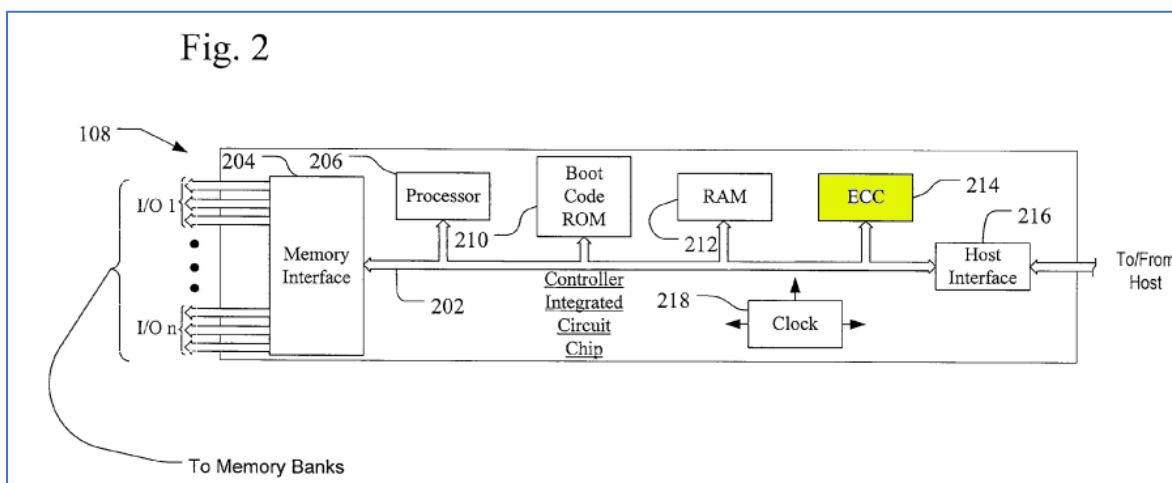
Sinclair discloses a “memory controller” having logic and a processor. *see* 1[j]. Figure 2 illustrates the memory, which a POSITA would understand stores the software instructions executed by the processor to perform the claimed functionality:



EX1003 ¶184; EX1004, Fig. 2.

11. Claim 14

A POSITA would understand that Sinclair's ECC is circuitry of the storage device system which detects failure conditions in its erase blocks:



EX1003 ¶186; EX1004, Fig. 2; 9[a]. As discussed in limitation 4[c] and 15[b], the Sinclair-Yamada system discloses transmitting information to the host when a maintenance operation is needed, e.g., when a failure condition is experienced.

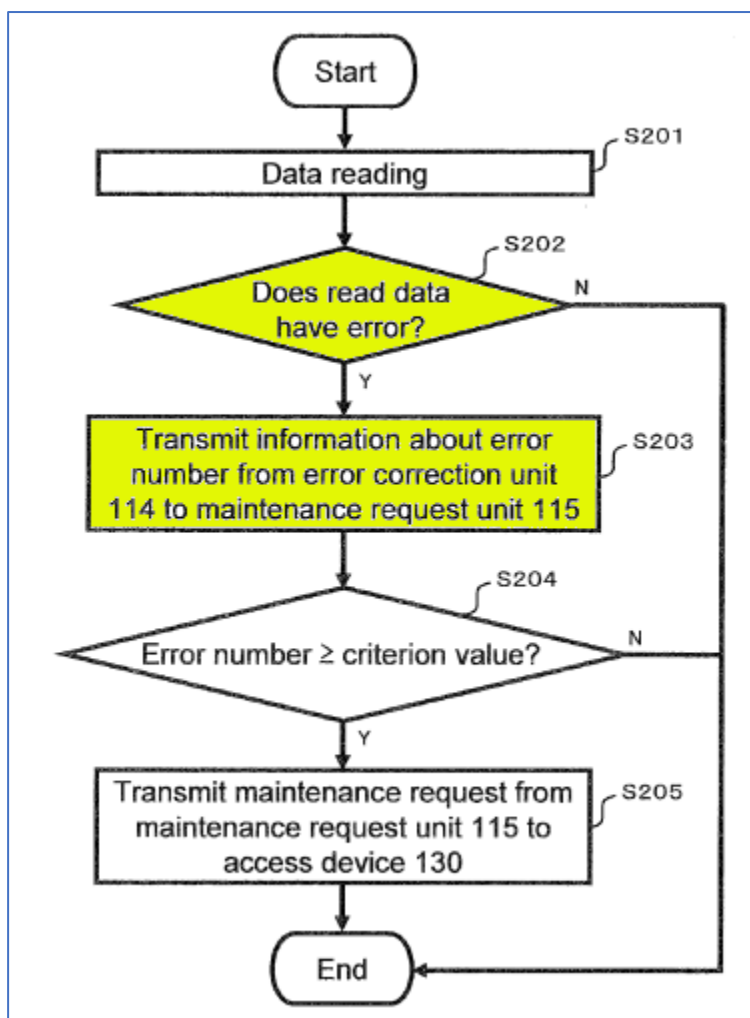
12. Claim 15

15[a]

Sinclair discloses circuitry to detect a failure condition of one of the erase blocks in the specific one of the respective subdivisions. 14[a]; EX1003 ¶188.

15[b]

In the combined Sinclair-Yamada ECC embodiment (9[a]), Yamada discloses transmitting a “**notification** of the number of data errors (S1 103)” to the host to notify it that the data stored in the nonvolatile memory is to be refreshed, as shown in Figure 11. EX1003 ¶190-91.



EX1005 Fig. 11 (annotated).

13. Claim 16

16[a]

Sinclair discloses that its controller generates and stores an LBA in its storage address table (SAT) corresponding to a DLBA run mapped to an LBA run [**address offset**]. Sinclair, ¶89 (“A DLBA run contains a backward link to its offset address within the LBA run to which it is mapped.... The SAT records the LBA offset that corresponds to the beginning of each DLBA run that is mapped to the LBA run.”);

EX 1003 ¶193. Sinclair also discloses that it maintains a “SAT page field pointer” that defines the offset address from the start of the LBA field [**address offset**]. Sinclair, ¶98; EX 1003, ¶193. This offset determines the target DLBA address to be written next. *Id.*, ¶108. A POSITA would understand that this SAT page field pointer must be generated or updated during write requests. EX 1003 ¶193.

16[b]

A POSITA would understand that Sinclair teaches that in order for the host to perform the LBA to DLBA lookup, the host needs to be aware of the updated SAT page field pointer [**transmit the address offset to the host**]. EX1003, ¶195. Sinclair also discloses that in some embodiments the “storage address re-mapping functionality is contained totally within the host.” EX1004, ¶59, Fig. 12. Thus, a POSITA would understand that the host would maintain the SAT page field pointer. EX1003, ¶195.

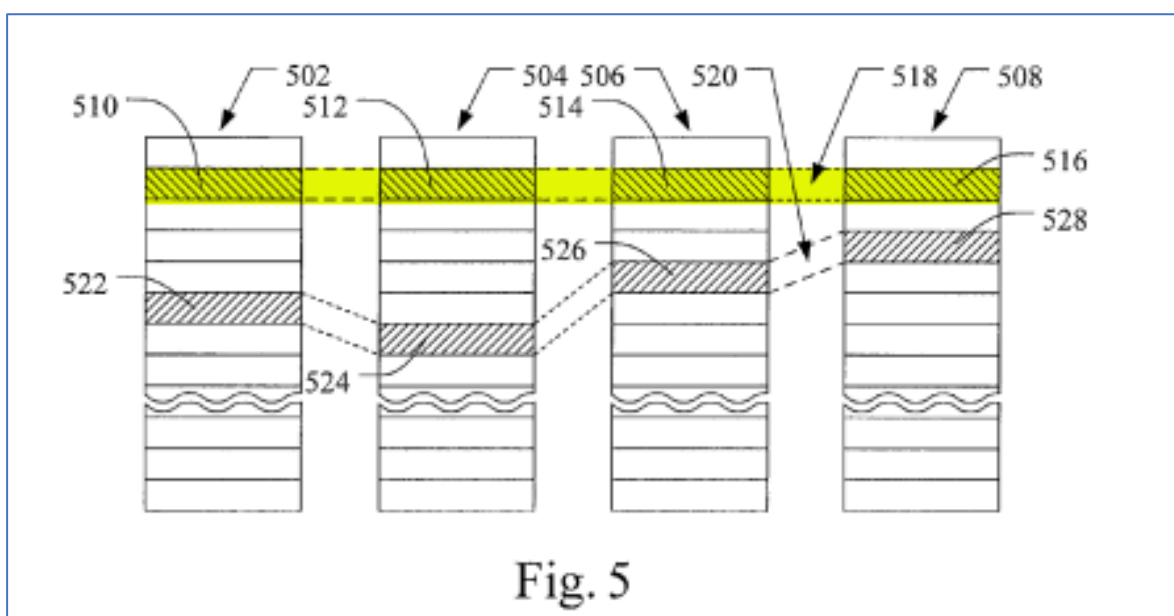
Yamada also discloses moving maintenance operations from the storage controller to the maintenance determination unit in a host. *see* 1[e]; EX1005, ¶¶10, 22. In this Sinclair-Yamada embodiment, the host maintenance determination unit would maintain the SAT page field pointer [**address offset**]. EX1003, ¶196.

14. Claim 21

21[a]

Sinclair discloses that the NAND flash memory comprises physical planes. EX1003, ¶198; EX1004, ¶46 (“...each chip may contain an array of memory cells

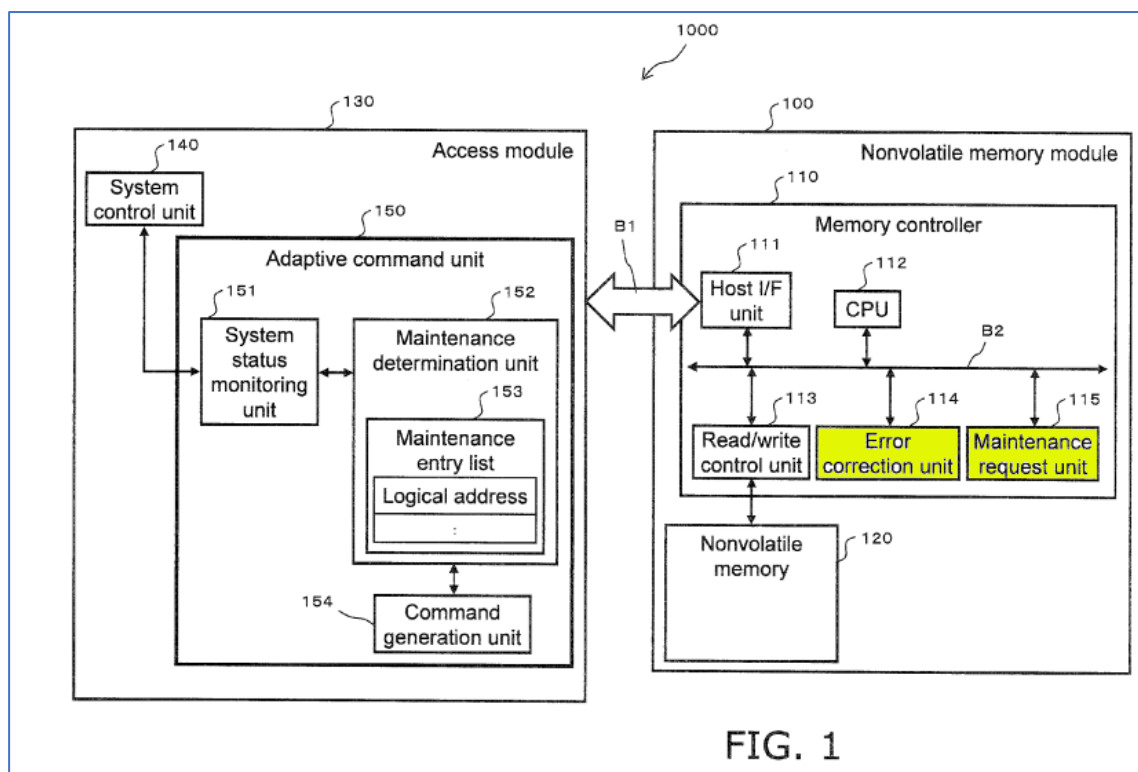
organized into multiple sub-arrays or planes.”). Sinclair discloses that a megablock is comprised of “at least one metablock 702 in each bank 107A-107D” and may form a “multiple plane configuration.” EX1004, Fig. 7, ¶56. Figure 5 illustrates a metablock (yellow) of a megablock comprising four individual blocks, 510-516 [erase blocks], that are located in respective planes 502-508 [e.g., in respective ones of the physical planes]:



EX1004, Fig. 5, ¶¶53-55; EX1003, ¶198.

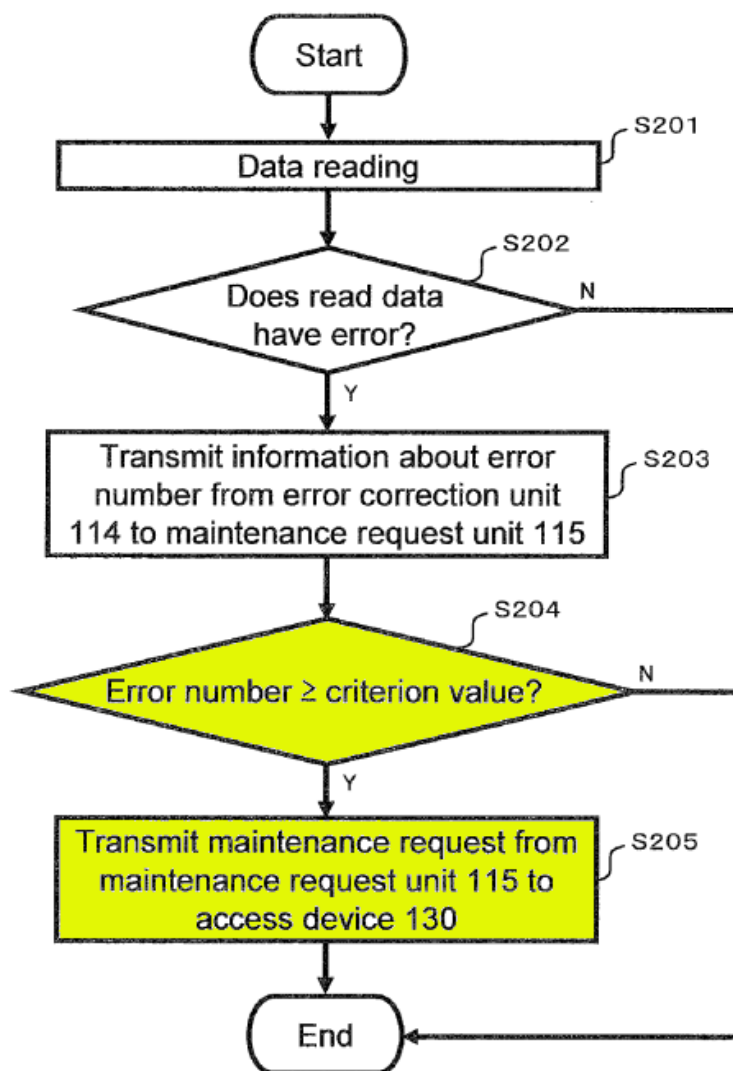
21[b]

Yamada discloses that the memory controller includes logic for an “error correction unit” (yellow) and a “maintenance request unit”:



EX1005, Fig. 1, ¶76; EX1003, ¶200.

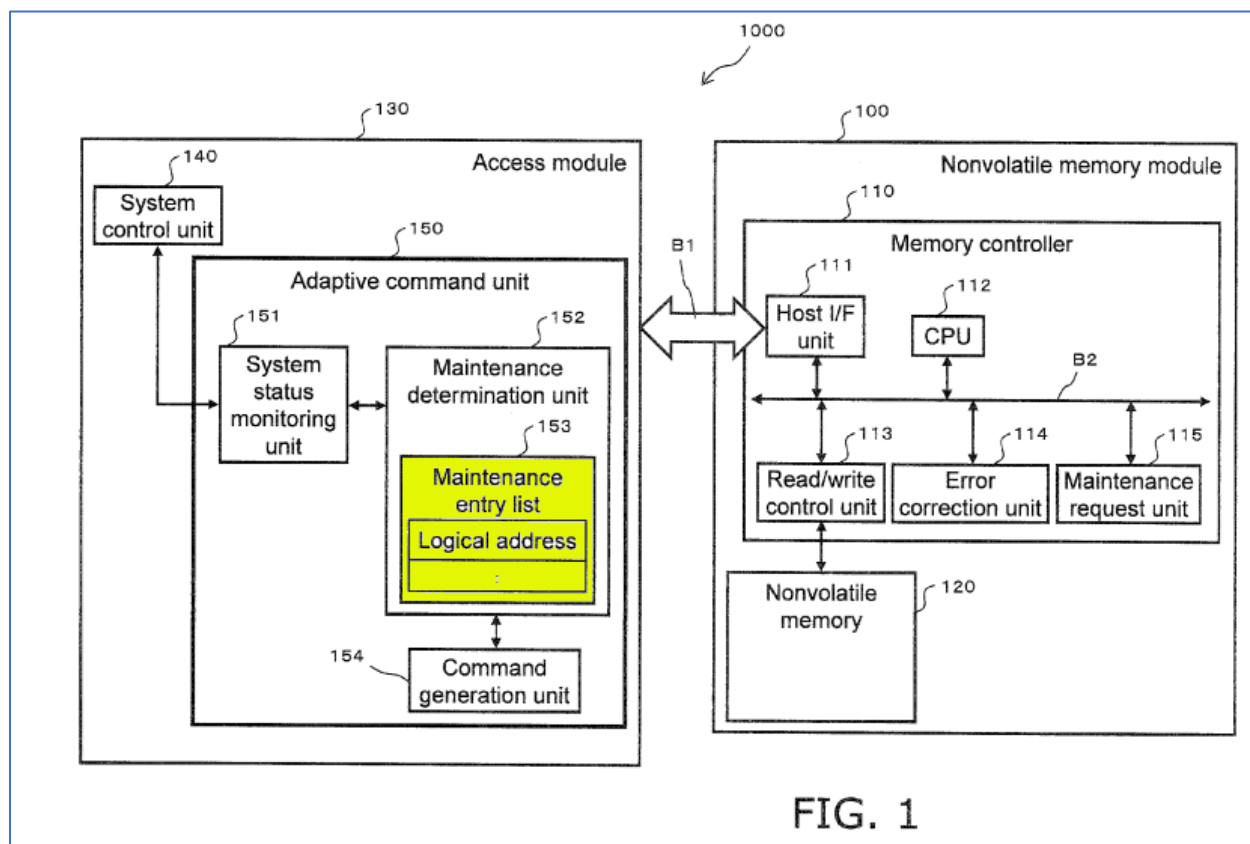
These two blocks work together to identify a need for maintenance dependent on the respective metadata and an associated logical address. EX1003, ¶201. For instance, the EC unit “determines the number of errors in the data” and “[w]hen the number of errors is greater than or equal to the criterion value, the maintenance request unit 115 transmits a maintenance request...” [**identify a need for maintenance...**], as shown below:



EX1005, Fig. 2, ¶¶96, 99; EX1003, ¶201. “The maintenance request includes a logical address of a block that needs maintenance.” EX1005, ¶¶82, 89. Thus, in the combined system, the memory controller would identify a need for maintenance of a megablock when, for instance, the error count exceeds a threshold that is dependent on the respective metadata, and would also identify the logical block address associated with the metablock to transmit with the maintenance request sent to the host, meaning, an associated logical address. EX1003, ¶202.

21[c]

Yamada discloses that “[t]he maintenance request includes a logical address of a block that needs maintenance,” which is stored in the maintenance entry list (yellow) of the access module [host]:

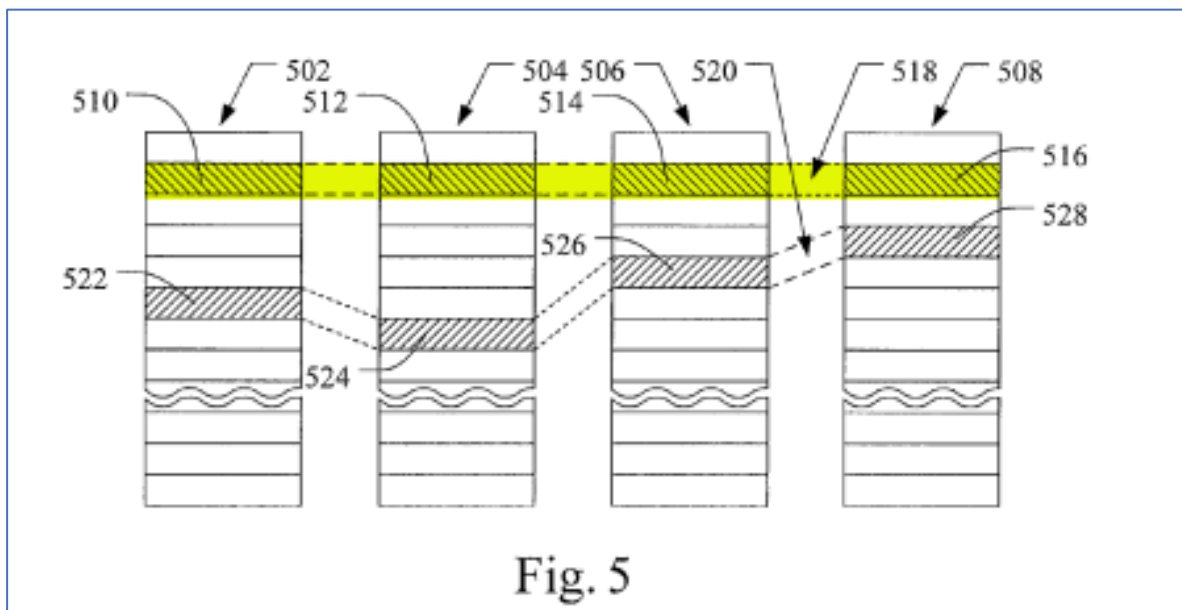


EX1005, ¶¶82, 89-91; EX1003, ¶205. The host then issues a “maintenance command... for an addressed stored in the maintenance entry list,” such as a “refresh and/or wear leveling” command for the block associated with the logical address. EX1005, ¶¶90-91, 30. A POSITA would understand that a refresh and/or wear leveling command includes an erase request. EX1003, ¶206. For instance, a POSITA would understand that wear leveling involves selecting a worn block,

copying its contents to a block with a lower wear count, and then erasing the worn superblock. EX1003, ¶206; EX1006, ¶82 (wear leveling process involves host instructing the controller to “cop[y] the contents of the worn superblock to the target superblock and erase[] the worn superblock”), Fig. 7.

21[d]

In the combined Sinclair-Yamada system, the associated logical address may be associated with a metablock having erase blocks that span two or more physical plans, as disclosed in Sinclair. EX1003, ¶208; EX1004, Fig. 5, ¶¶53-56. For instance, it may be for the metablock (yellow) that is shown in Figure 5, which has four individual blocks, 510-516 [**erase blocks**] that are located in respective planes 502-508:



EX1004, Fig. 5, ¶¶53-55; EX1003, ¶208. Sinclair discloses that the controller erases the blocks of the metablock, which correspond to a specific megablock, together

[**control erasure of two or more of the erase blocks...**]. EX1004, ¶¶50, 54. A POSITA would understand this disclosure to be a control erasure of two or more of the erase blocks... including erase blocks in each of at least two different ones of the physical planes. EX1003, ¶208.

15. Claim 24

In the combined Sinclair-Yamada system, in association with the physical erasure, the system modifies the information representing the extent to which the one or more of the erase blocks of the specific one of the respective subdivisions are full. EX1003, ¶210. For instance, Sinclair discloses that its system tracks the number of red blocks, pink blocks, and white blocks. EX1004, Fig. 8, ¶¶58, 67. Sinclair discloses that white blocks are “blocks with no valid data, and thus containing only unwritten capacity,” such that in association with the physical erasure (which will transform a red or pink block into a white block) the system will update the number of a red, pink, and white blocks [**information...**]. EX1004, Fig. 8, ¶¶58, 67 (“This determination may be made based on a total number of white blocks that currently exist...”); EX1003, ¶210.

IX. GROUND II: CLAIMS 1-4, 6, 9-16, AND 21, 24 OBVIOUS IN VIEW OF TERASAKI AND YAMADA.

A. Overview of the Prior Art

1. Terasaki (EX1018)

Terasaki is prior art under at least 35 U.S.C. § 102(a). *See* EX1003, ¶¶60-65.

EX1018 is a certified translation of Terasaki.

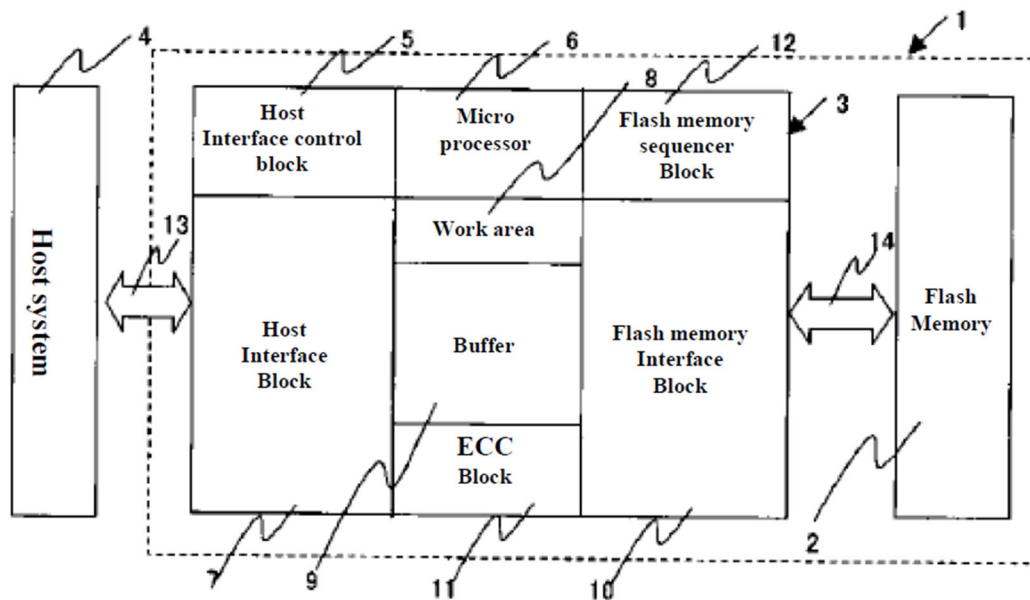
B. Claim-by-Claim Analysis²

1. Claim 1

(a) 1[pre]

Terasaki discloses a storage device. EX1003, ¶216.

(FIGURE 1)



² *See* Table of Claims, *supra*, for full text of limitations.

EX1018, Fig. 1. The Terasaki flash memory system includes a flash memory 2 and controller 3 that controls the flash memory and is attached to a host system 4, such as a personal computer or digital camera it interfaces with. EX1018, ¶¶18-19, 21.

1[a]

Terasaki discloses a flash memory that is non-volatile memory. EX1018, ¶18-19, 2-3; EX1003, ¶218.

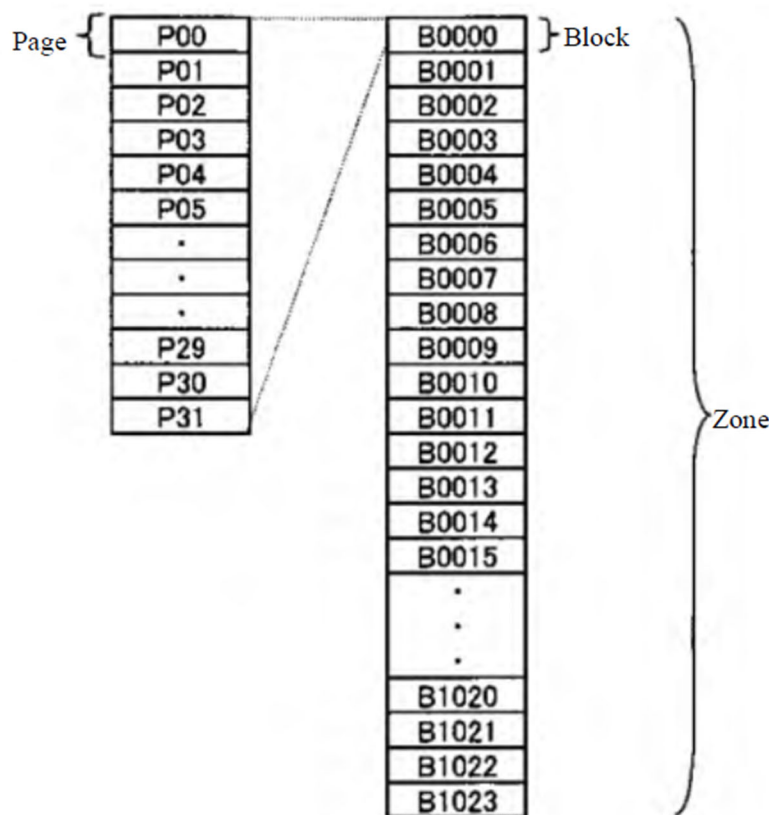
1[b]

Terasaki's system includes a controller 3 with logic to perform the claimed limitations and control the flash memory and interface with host system 4, such as a personal computer or digital camera. EX1003, ¶220; EX1018, ¶¶18-19, 21.

1[c]

Terasaki discloses zones consisting of a plurality of blocks having consecutive physical block addresses within the memory [**subdivisions of physical storage in the NAND flash memory**]. EX1018, Abstract, ¶41, Fig. 5. A block is a “unit of erase processing” such that the data on all of the pages in a block [**erase blocks**] must each be physically **erased as a unit**. EX1003, ¶222; EX1018, ¶3. Figure 5 illustrates the relationship between the zones, blocks, and pages of the flash memory whereas the block is the unit of erasing:

(FIGURE 5)



EX1018, Figs. 5-8. For instance, a Zone is illustrated comprising blocks B000-B1023 with each block comprising multiple pages, such as block B000 comprising pages P00-P31. *Id.*

Terasaki discloses its LBAs are mapped to physical logical block addresses in memory by the use of an address translation table that is updated each time data is written. EX1018, ¶¶5-6. In Terasaki’s system, the LBAs of the data stored within adjacent blocks will be “continuous” [**non-overlapping range**] “when aligning the blocks constituting said zone in sequence of the physical block address.” EX1018, ¶10. Figures 6-8 illustrate Terasaki’s zone management table at different states:

(FIGURE 6)

Zone	Logical block address
Zone 1	LBA10000
Zone 2	LBA11000
Zone 3	LBA12000
Zone 4	LBA13000
Zone 5	LBA14000
Zone 6	Erased
Zone 7	Erased
Zone 8	Erased
Zone 9	Erased
Zone 10	Erased
Zone 11	Erased
Zone 12	Erased
Zone 13	Erased
Zone 14	Erased
Zone 15	Erased
Zone 16	Erased
Zone 17	Erased
Zone 18	Erased
Zone 19	Erased
Zone 20	Erased

(FIGURE 8)

Zone	Logical block address
Zone 1	LBA10000
Zone 2	Erase request
Zone 3	LBA12000
Zone 4	LBA13000
Zone 5	LBA14000
Zone 6	LBA15000
Zone 7	LBA11000
Zone 8	Erased
Zone 9	Erased
Zone 10	Erased
Zone 11	Erased
Zone 12	Erased
Zone 13	Erased
Zone 14	Erased
Zone 15	Erased
Zone 16	Erased
Zone 17	Erased
Zone 18	Erased
Zone 19	Erased
Zone 20	Erased

EX1018, Figs. 6-8; EX1003, ¶223. The zone management table stores the state of each zone (write state, erase state, erase request state, etc.). EX1018, ¶49.

1[d]

Terasaki discloses receiving write commands from the host system, where each write command contains data and a logical address supplied from the host system **[receive write requests, accompanying addresses and accompanying data from a host and, with respect to each one of the write requests]**. EX1003, ¶225; EX1018, ¶7 (“when a write command for data is received from the host system, said address translation table is used as the basis to determine whether or

not there is a block where the same data as that data has been stored in a logical address (the address that may be supplied from the host system).”). Terasaki discloses that in its flash memory system, data and logical block addresses (or a “host address”) may be supplied from host system. EX1003 ¶225; EX1018, ¶¶38, 43.

1[d.i]

Terasaki discloses that a corresponding LBA indicates which LBA the block corresponds to when data is stored in the block [**identify a specific one of the respective subdivisions dependent on which one of the non-overlapping ranges encompasses the address accompanying the one of the write requests**]. EX1018, ¶40; EX1003 ¶227.

Specifically, Terasaki discloses a zone management table that manages the storage status of each zone and manages the LBA of the data written onto each zone by zone, as show in Fig. 6. EX1018, ¶45.

(FIGURE 6)

Zone	Logical block address
Zone 1	LBA10000
Zone 2	LBA11000
Zone 3	LBA12000
Zone 4	LBA13000
Zone 5	LBA14000
Zone 6	Erased
Zone 7	Erased
Zone 8	Erased
Zone 9	Erased
Zone 10	Erased
Zone 11	Erased
Zone 12	Erased
Zone 13	Erased
Zone 14	Erased
Zone 15	Erased
Zone 16	Erased
Zone 17	Erased
Zone 18	Erased
Zone 19	Erased
Zone 20	Erased

EX1018, Fig. 6; EX1003 ¶228. This initial stored data is written onto the zone management table above for the following LBAs:

Zone 1: LBA10000-LBA10999
Zone 2: LBA11000-LBA11999
Zone 3: LBA12000-LBA12999
Zone 4: LBA13000-LBA13999
Zone 5: LBA14000-LBA14999

EX1018, ¶45.

1[d.ii]

Terasaki discloses that in its flash memory system, write processing in zone unit are performed “in order to enable high-speed data processing of data in a single

group with consecutive host addresses in continuous write processing” [**program the data accompanying with the one of the write requests**] referred to as group data write processing. EX1018, ¶44; EX1003 ¶231.

During write processing, Terasaki’s zone management table is written with data corresponding to the LBA into the next available zone. EX1003 ¶232; EX1018, ¶45. If an external host command is received to write data corresponding to LBA15000-LBA15999, an empty erased zone is found within the zone management table, and the data is written onto that erased zone [**program the data...**]. EX1018, ¶46. This operation is described with reference to Figures 6 and 7, where Zone 6 is the next erased zone available to be written into, and the management table “is revised from the erased flag to LBA15000 (the initial logical block address of the stored data).” EX1018, ¶46; EX1003 ¶232.

(FIGURE 6)

Zone	Logical block address
Zone 1	LBA10000
Zone 2	LBA11000
Zone 3	LBA12000
Zone 4	LBA13000
Zone 5	LBA14000
Zone 6	Erased
Zone 7	Erased
Zone 8	Erased
Zone 9	Erased
Zone 10	Erased
Zone 11	Erased
Zone 12	Erased
Zone 13	Erased
Zone 14	Erased
Zone 15	Erased
Zone 16	Erased
Zone 17	Erased
Zone 18	Erased
Zone 19	Erased
Zone 20	Erased

(FIGURE 7)

Zone	Logical block address
Zone 1	LBA10000
Zone 2	LBA11000
Zone 3	LBA12000
Zone 4	LBA13000
Zone 5	LBA14000
Zone 6	LBA15000
Zone 7	Erased
Zone 8	Erased
Zone 9	Erased
Zone 10	Erased
Zone 11	Erased
Zone 12	Erased
Zone 13	Erased
Zone 14	Erased
Zone 15	Erased
Zone 16	Erased
Zone 17	Erased
Zone 18	Erased
Zone 19	Erased
Zone 20	Erased

→

EX1018, Figs. 6-7 (annotated).

Terasaki further discloses an embodiment that includes a redundant region of the zones as well as the zone management information table, and data is written to the physical data region as described with reference to Figures 9-13. EX1003 ¶233; EX1018, ¶53. When an external command is received from the host system to write data corresponding to a logical block address for LBA10000-LBA10999 in this

erased zone, data will be written in order from B0001 set to the write start pointer [next available storage location], as shown by Figure 9:

(FIGURE 9)

		Data region	Redundant region
Erased Start B0001 End B0001 Bad B0004 B0009 B1021		B0000	
		B0001	
		B0002	
		B0003	
		B0004	Bad
		B0005	
		B0006	
		B0007	
		B0008	
		B0009	Bad
		B0010	
		B0011	
		B0012	
		B0013	
		B0014	
		B0015	
		.	.
		.	.
		.	.
		.	.
		.	.
		B1020	
		B1021	Bad
		B1022	
		B1023	

EX1018, Fig. 9; EX1018, ¶53; EX1003 ¶233. In this write processing, the data corresponding to a logical block address for LBA10000-LBA10999 is sequentially written using write registers starting at B0001 of the physical block address. *Id.*

1[d.iii]

Terasaki discloses that its zone management table is updated to change the storage status of each zone after a host write request [**in association with the programming...update**], managing the LBAs of the data written onto each zone by zone. EX1018, ¶¶45-47. As such, the zone management table maintains updated “information indicating the state of each zone (write state, erase state, erase request state, etc.)” and “information on the data stored in each zone (information on the physical block address of the block where the data is stored, information on the logical block address of the stored data, etc.).” EX1018, ¶49; EX1003 ¶236.

Such information represents **an extent to which the erase blocks in the zones are full**. EX1003 ¶¶236-37. A zone in a write state contains the initial logical block address of the stored data. EX1018, ¶¶45-46. A zone for which all blocks are in an erase state are referred to as erased zones having “erased” flags indicating this status. *Id.* Finally, a zone with a stored erase request flag indicates that erase processing is necessary to convert the zone to an erased zone. EX1018, ¶47. Thus, a zone having information on data stored indicates a zone that is completely full, whereas a zone flagged as an erased zone indicates that the zone is completely empty. EX1003, ¶237; EX1018, ¶¶45-47.

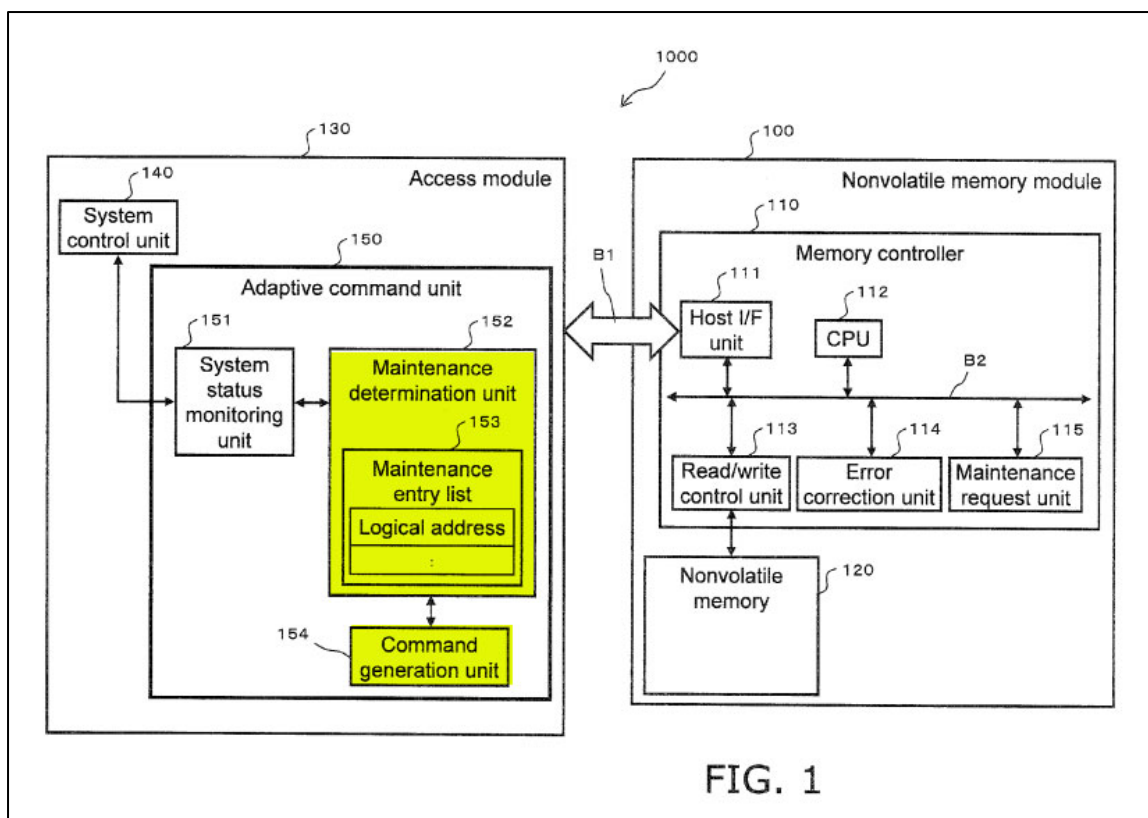
1[e]

Terasaki discloses zones with erase request flags indicating a need for maintenance. EX1003 ¶239. As discussed in 1[d], Terasaki discloses that a zone management table contains the state of each zone (write/erase/erase request) and information on the data stored in each zone such as its size. EX1018, ¶49 (“Each zone may store up to 1000 blocks of data, and 24 blocks are spare blocks that take into account the occurrence of bad blocks or the like.”). Terasaki further discloses that a zone with a stored “erase request” flag indicates that erase processing is necessary to convert the zone to an erased zone. EX1018, ¶47. When data is rewritten, the written flag on the first page of the initial good block in the zone in which the original data is written is changed to an erase request flag for the up to 1000 blocks of that one zone [**store a value representing a size of the NAND flash memory that is to be physically erased**]. EX1003, ¶239; EX1018, ¶68. This erase request flag indicates that erase processing for that zone is necessary. *Id.*

Thereafter, during a subsequent write request from the host when maintenance is needed, an erase process is performed [**fulfilling a maintenance request issued by the host**] where the block is changed to an erased block. EX1003 ¶240; EX1018, ¶¶60, 68, 65 (“the zone management information that has been written on the first page of the initial block... and then the data of the initial good block is erased.... In this change, the erase request flag... is changed to the erased flag”).

Terasaki also discloses that requests that change the erase request flag to an erased flag [**maintenance**] may be initiated by the host [**request issued by the host**]. EX1003, ¶241. Terasaki discloses that its controller and host exchange data/addresses and status/command information via host interface block which includes a task file register storing addressess and commands. EX1018, ¶¶24-25. A POSITA would understand these disclosures to teach that the necessary erase processing that is performed on zones flags as “erase request” is initiated from command information sent by the host to the controller. EX1003, ¶241.

To the extent Terasaki does not teach the erase processing request may be initiated by the host, it would have been obvious in further view of Yamada. EX1003, ¶242. Yamada discloses moving maintenance operations from the storage controller to its “maintenance determination unit” and “command generation unit” at the host for causing maintenance for data retention, such as refresh, to be performed for an address stored in the maintenance entry list. EX1005, ¶10, 22.



Ex 1005, Fig. 1, ¶¶89-90. In the combined Terasaki-Yamada system, the host would issue the erase processing request [**maintenance request issued by the host**] to change zones flagged as “erase requested” to an “erased status.” EX1003, ¶242.

Motivation to Combine: A POSITA would be motivated to combine Terasaki and Yamada for similar reasons as explained in Ground 1 Limitation 1[e], so that a management request, such as erase processing, is issued by the host, because as with Sinclair, the prior art includes express teachings, suggestions and motivations to combine the teachings of Terasaki and Yamada such that the erase processing request is issued by the host. EX1003, ¶¶212-214.

1[f]

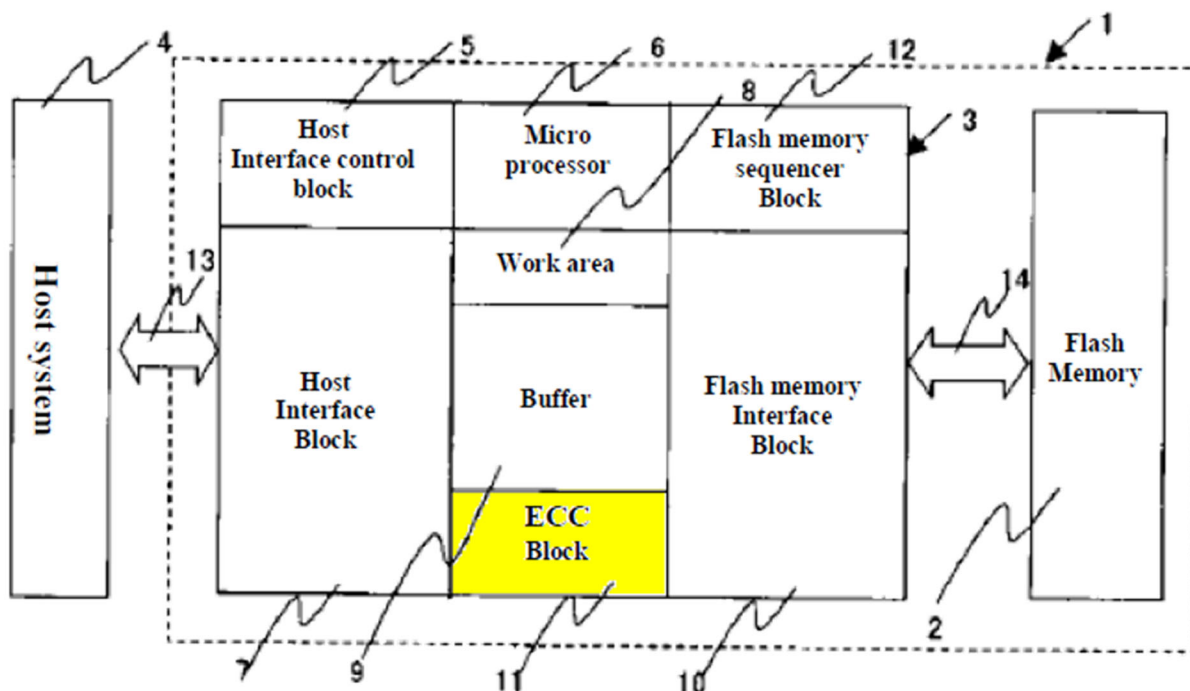
Terasaki discloses that the controller and host exchange data, address information, status information [**provide to the host... the stored value**], and command information via host interface block 7 which includes a task file register storing addressess and commands, and an error register that may be set in the event of an error. EX1003, ¶246; EX1018, ¶¶24-25. Specifically, status information is “fed into host system 4 using host interface block 7 as an outlet.” EX1018, ¶24. For instance, Terasaki discloses that to pass error information to the host, the host interface block that is used to pass information from the controller to the host may include an error register that may be set in the event of an error. EX1018, ¶25; EX1003, ¶246.

This limitation is further made obvious in view of Yamada. EX1003, ¶247. As explained in 1[e], in the combined Terasaki-Yamada system, the host would issue the erase processing request. EX1005, ¶¶14, 109-111. A POSITA would understand these disclosures to teach that in such an embodiment the host would maintain Terasaki’s zone management table including the updated status of each zone, in order for the host to successfully perform this task. EX1003, ¶247.

1[g]

Terasaki discloses that to keep track and perform error correction on memory, its flash memory system includes an error collection code (ECC) block 11, illustrated in Figure 1:

(FIGURE 1)



EX1018, Fig. 1 (annotated). ECC block 11 generates an error collection code [metadata] identifying when an error is detected during writing of data to the flash memory [represents a characteristic of the data] in each zone. EX1018, ¶30 (“ECC block 11 is a functional block that generates an error collection code to be appended to the data to be written to flash memory 2, and it also detects and corrects errors contained in the retrieved data based on the error collection code appended to the data that was read out.”). EX1003, ¶249.

Terasaki also discloses that its zone management information table stores “information related to the number of erase operations that have been performed in said zone” as an erase count [**metadata**]. EX1003, ¶250; EX1018, ¶¶16, 47-49. This erase count information for each zone indicates the number of erases performed for a zone, and is used to manage the order of priority of the erased zones to be used [**a characteristic of the data in the specific one of the respective subdivisions**]. *Id.*; EX1018, ¶49 (“when managing the erase count, etc., for each zone to ensure that each zone is used equally, it is preferable to write the erase count, etc., for that zone in the management information storage block.”). A POSITA would understand that this order of priority is a characteristic of the data in each zone. EX1003, ¶250.

Terasaki maintains the zone information of the zone management table up to date [**update metadata**] so that priority of the erased zones is possible. EX1003, ¶251; EX1018, ¶49 (“As long as information showing the state of each zone, information related to the data that has been stored in each zone, or the zone management information such as the erase count performed for a zone, etc., is written in the management information storage block set within each zone in this manner, it will be possible to prepare the zone management table based on that information and to manage the order of priority of the erased zones to be used.”). As a result, this erase count may be used to perform wear-leveling operations so

writing processing for the data into a zone is balanced. EX1003, ¶251; EX1018, ¶49 (“manage the order of priority of the erased zones to be used.”).

1[h]

Terasaki further discloses that an error collection code [**metadata**] is used to detect an error **condition** where the errors will need to be corrected. EX1003, ¶253. The “error collection code is additional information for correcting errors contained in the data... and it is generated by ECC block 11.” EX1018, ¶39. Terasaki’s flash memory system stores this error information in pages constructed of byte regions, where “additional information, such as error collection codes, corresponding logical block addresses, and the block status, may be stored.” EX1018, ¶38, Fig. 4.

Terasaki discloses that based on the stored error collection code, if the number of errors contained in the data in a user area is less than or equal to a predetermined number [**comparison based on the metadata to detect a condition**], then errors will be corrected. EX1003, ¶254; EX1018, ¶39. In the event of an error, an error register is set. EX1018, ¶38 (“host interface block 7 includes... an error register (not shown in the figure) that may be set in the event of an error.”).

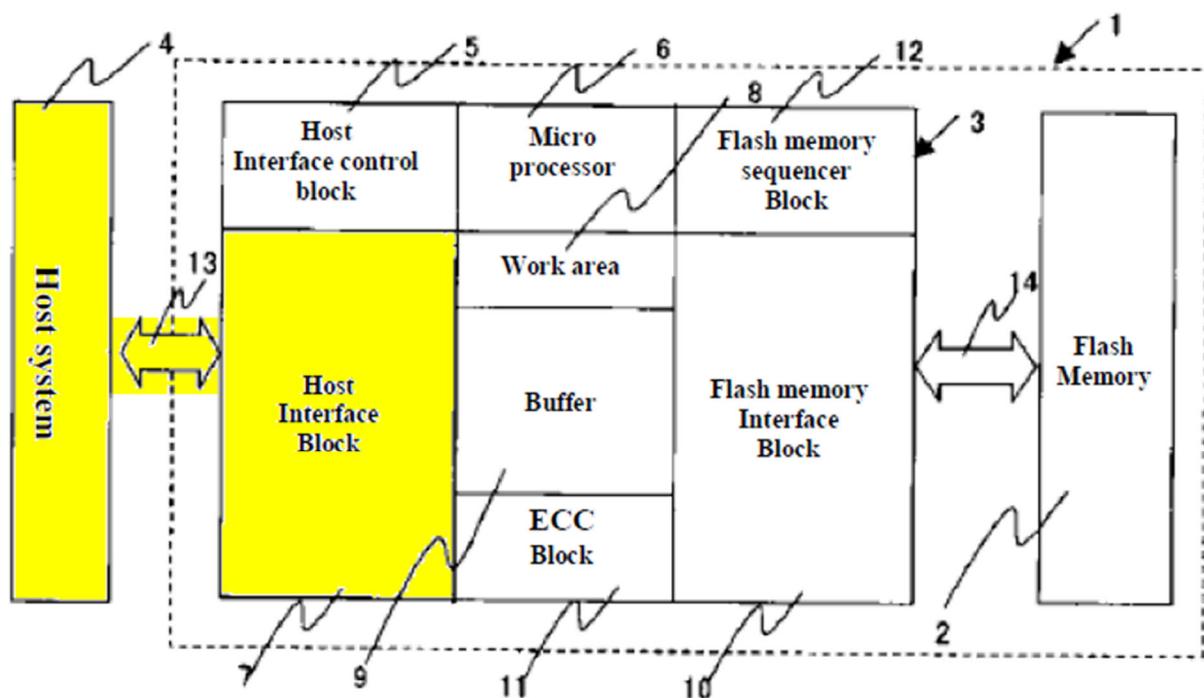
Terasaki also maintains a zone erase count [**metadata**] that is used to prioritize zones to be used for writing host data. *see* 1[g]. EX1003, ¶255; EX1018, ¶¶16, 47-49. Terasaki explains that the stored erase count performed for a zone is written in the management information storage block set within each zone so that “it

will be possible to prepare the zone management table based on that information and to manage the order of priority of the erased zones to be used.” EX1018, ¶49. A POSITA would understand that in order to prioritize erase zones to be used based on their zone erase count value, a comparison of the zone erase counts from one zone to another is necessary [**perform a comparison based on the metadata**] in order to prioritize the zone with the least amount of erase counts [**condition**]. EX1003, ¶256; EX1018, ¶49.

1[i]

Terasaki discloses that based on the stored error collection code used to detect that errors need to be corrected [**responsive to detection of the condition**], an “error register” included in host interface block “may be set in the event of an error” [**store information... wherein the information... indicates that the condition was detected**]. EX1018, ¶25; EX1003, ¶258. This error register is included in the “host interface block” that is used to “exchange...status information” [**information is accessible to the host**]. EX1018, ¶¶24-25; EX1003, ¶258. Host interface block 7 is illustrated in Figure 1:

(FIGURE 1)



EX1018, Fig. 1 (annotated). Status information including the “error register” containing information set in the event of an error is “fed into host system 4 using host interface block 7 as an outlet.” EX1018, ¶24; EX1003, ¶258.

Terasaki also discloses storing erase count information responsive to detection of the condition. EX1003, ¶259; *see* 1[b]. The erase counts is **responsive to the detection of the condition** because erase counts are increased when a zone is identified for erasing based on the detection of the aforementioned conditions. *Id.* Terasaki discloses that the erase count information is also accessible to the host via the host interface 7 and indicates that the condition was detected. EX1003, ¶259; EX1018, ¶¶24-25; *see* 1[f].

This limitation is further made obvious in view of Yamada. EX1003, ¶260. As explained in 1[e], a POSITA would understand Terasaki's disclosures of exchanging status information with the host to teach that the zone erase count or zone to be prioritized [**information**] is provided to the host issuing erase requests in order for the host to successfully perform this task. EX1003, ¶261.

1[j]

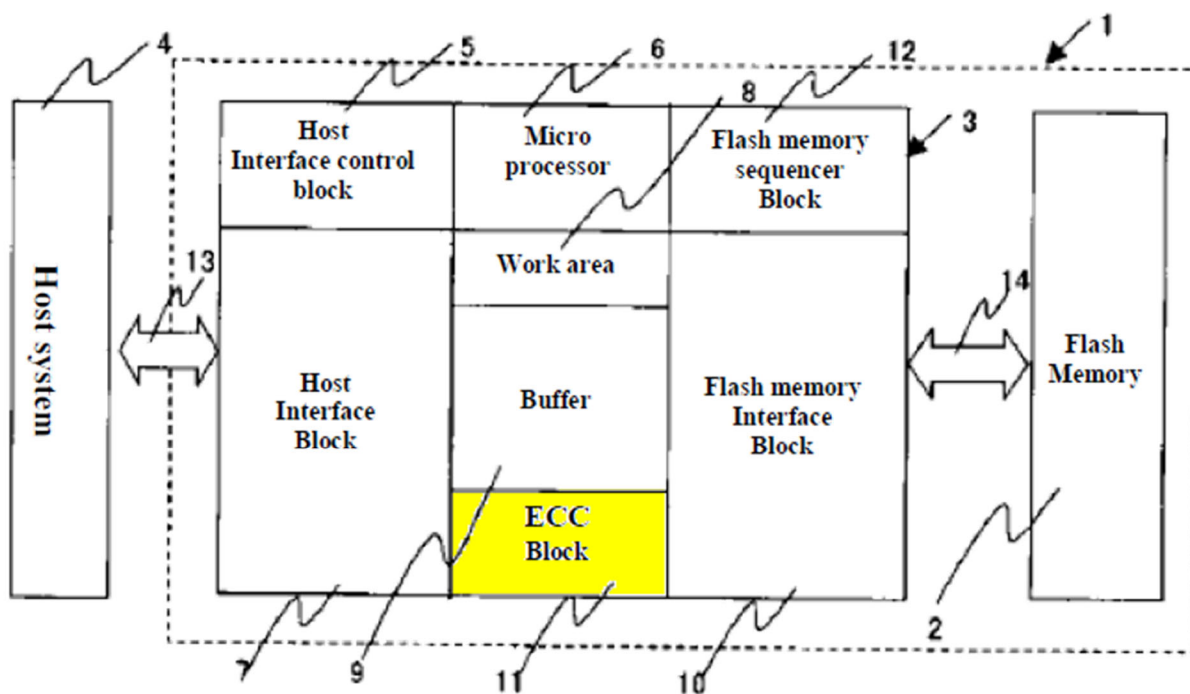
Terasaki discloses a memory controller having logic. EX1003, ¶263; 1[b]. Terasaki further discloses a microprocessor 6 that controls the overall operations of the controller (EX1018, ¶22) and that its flash memory system may be hosted on a personal computer or digital camera. EX1003, ¶263; EX1018, ¶¶18-19.

2. Claim 9

9[a]

Terasaki discloses that to keep track and perform error correction on memory, its flash memory system includes an ECC block 11:

(FIGURE 1)



EX1018, Fig. 1 (annotated). The ECC generates an error collection code [**the metadata is a function of data error in the data being characterized**] for data written to the flash memory and detects and corrects errors contained in the data based on that collection code. EX1003, ¶266; EX1018, ¶¶30, 38-39, Fig. 4; *see* 1[g]. A POSITA would understand this error collection code to be metadata that is a function of data error in the data being characterized. EX1003, ¶266.

Terasaki further discloses a “bad block flag” [**metadata is a function of data error in the data being characterized**] that “indicates whether the block is a bad block.” EX1003, ¶267; EX1018, ¶41. If it is determined that a memory block is a “bad block” then the system will set the bad block flag. *Id.* Figure 9 illustrates this information on bad blocks present within a zone:

(FIGURE 9)

	Data region	Redundant region
Erased	B0000	
Start	B0001	
B0001	B0002	
End	B0003	
B0001	B0004	Bad
Bad	B0005	
B0004	B0006	
B0009	B0007	
B1021	B0008	
	B0009	Bad
	B0010	
	B0011	
	B0012	
	B0013	
	B0014	
	B0015	
	.	.
	.	.
	.	.
	.	.
	B1020	
	B1021	Bad
	B1022	
	B1023	

EX1018, Fig. 9 (annotated), ¶52.

Terasaki also discloses an “error register [] that may be set in the event of an error” which a POSITA would understand to be metadata that is a function of data error in the data being characterized. EX1003, ¶268; EX1018, ¶24-25; *see* 1[g].

9[b]

Terasaki discloses that, “[b]ased on this error collection code, if the number of errors contained in the data stored... is less than or equal to a predetermined

number” [condition] then “the errors will be corrected” [**determination... data should be moved**]. EX1018, ¶39.

A POSITA would understand that when a block that holds valid data is determined to be bad, the data contained in that block needs to be moved to a new physical location, i.e., a good block, to prevent data loss. EX1003, ¶272. For instance, a POSITA would understand that that if the number of read errors in one block (correctable errors) surpasses some threshold, that data needs to be moved before the block is no longer readable. EX1006, Fig. 7 (step 710-730); EX1005, ¶7 (“data refreshing (data rewriting)”); EX1003, ¶272.

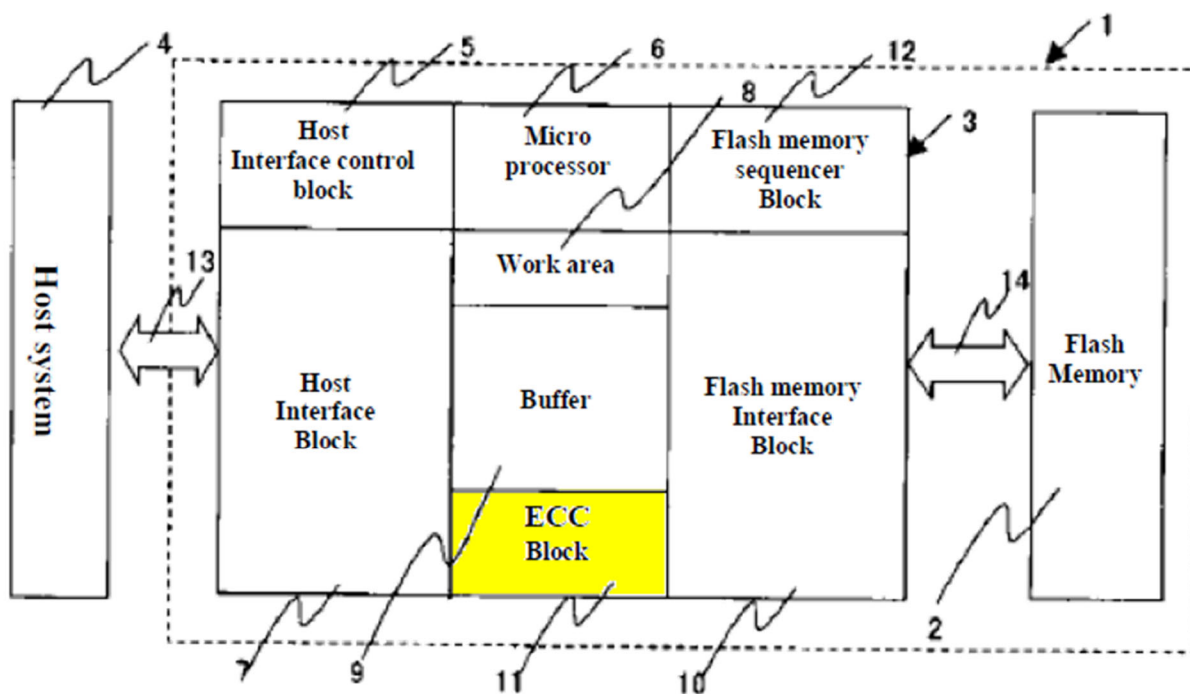
3. Claim 13

Terasaki discloses a microprocessor 6 [**at least one processor**] that controls the overall operations of the controller (EX1018, ¶22) and non-volatile flash memory [**at least one physical storage medium**]. EX1003, ¶274; EX1018, ¶¶18-19.

4. Claim 14

Terasaki discloses that its flash memory system includes an ECC block 11 [**circuitry to detect a failure condition**] as illustrated in Figure 1:

(FIGURE 1)



EX1018, Fig. 1 (annotated); EX1003, ¶276. The ECC block generates an error collection code for data written to the flash memory and detects and corrects errors contained in the data based on that collection code [**detect a failure condition...**]. EX1003, ¶277; EX1018, ¶30; *see also* 1[g].

Terasaki also discloses an “error register []” that may be set in the event of an error” that was detected [**information to the host that identifies...the failure condition**]. EX1003, ¶278; EX1018, ¶¶24-25. This error register is included in the “host interface block 7” that is used to “exchange data, address information, status information, and external command information with host system 4” [**transmit information to the host**]. *Id.*

5. Claim 15

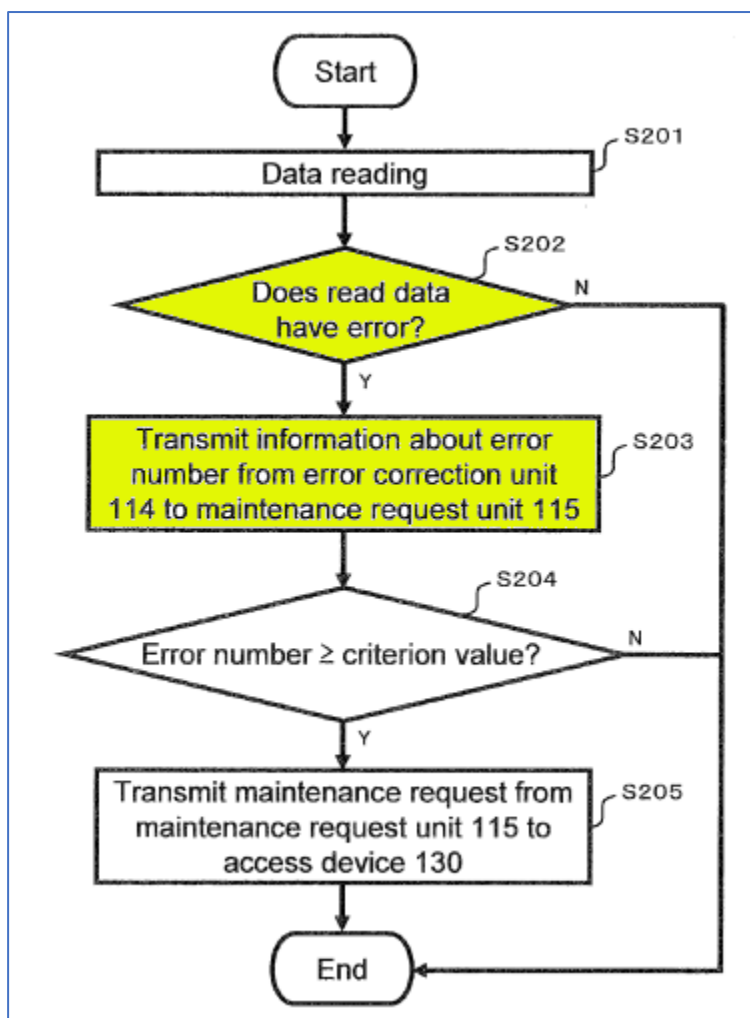
15[a]

Terasaki discloses circuitry to detect a failure condition of one of the blocks in the specific one of the respective subdivisions. EX1003, ¶280; *see* 14[a].

15[b]

Terasaki discloses that the “error register” information with the detected error collection code and “address information” are both “exchanged” with the host via host interface block 7. EX1018, ¶¶24-25.

A POSITA would also have found it obvious for combine the additional notification features of the error correction unit of Yamada to implement the disclosed error correction ECC Block of Terasaki. EX1003, ¶284; *see* 1[e]. Yamada discloses transmitting a “**notification** of the number of data errors (S1103)” to the host to notify it that the data stored in the nonvolatile memory is to be refreshed, as shown in Figure 11:



X. GROUND III: CLAIM 22 IS OBVIOUS IN VIEW OF SINCLAIR, YAMADA AND OLBRICH

A. Overview of the Prior Art

1. Olbrich (EX1010)

Olbrich is prior art under at least AIA 35 U.S.C. § 102(a). See EX1003, ¶¶66-

69.

B. Claim-by-Claim Analysis³

1. Claim 22

22[a]

Sinclair discloses this limitation. EX1003, ¶286; *see* 21[a].

22[b]

Sinclair discloses that for “increased parallelism” a set of erase blocks may be grouped together and “operated in larger metablock units” and that the erase blocks may span “all of the planes.” EX1004, ¶¶53-54, Fig. 5.

To the extent Sinclair does not disclose using a “common address offset for each of the respective ones of the physical planes,” it would have been obvious in view of Olbrich. EX1003, ¶289.

Multi-plane commands were well-known in the art. EX1003, ¶290; EX1011, §1.3 (“SSD manufacturers have provided advanced commands, such as... multi-plane...”), §2.2 (“Multi-plane command activates multiple read, program or erase operations in all planes of the same die”), Fig. 4. A POSITA further understood that a multi-plane command typically has the same “page addresses” across planes. EX1003, ¶290; EX1011, §2.3 (“The pages executing a multi-plane read/write operation must have the same chip, die, block and page addresses,” but may have different plane addresses), Fig. 3. Olbrich discloses a multi-plane command for a

³ *See* Table of Claims, *supra*, for full text of limitations.

flash memory that is, like Sinclair, subdivided into groups of blocks that comprised of pages and logically linked together. EX1010, 15:60-67, Fig. 12. Olbrich further teaches that its multi-plane command uses the “same address location” when writing to pages that are “spread across” multiple dies (and planes). EX1010, 15:57-67. For instance, Figure 12 shows “SuperPage 30 is spread across four separate dies” and consists of “four flash Pages” that have the “same address location” across planes. *Id.*, Fig. 12; EX1003, ¶290. A POSITA would understand that pages are specified by an offset and that pages that “spread across four separate dies” are on different planes. EX1003, ¶290. Thus, Sinclair’s disclosure that pages on different planes have the “same address location” discloses **a common address offset for each of the respective ones of the physical planes**. EX1003, ¶290.

XI. GROUND IV: CLAIM 23 IS OBVIOUS IN VIEW OF SINCLAIR, YAMADA, SINCLAIR-II, AND YU

A. Overview of the Prior Art

1. Sinclair-II (EX1015)

Sinclair-II is prior art under at least AIA 35 U.S.C. § 102(a). *See* EX1003, ¶¶70-73.

2. Yu (EX1016)

Yu is prior art under at least 35 U.S.C. § 102(a). *See* EX1003, ¶¶74-77.

B. Claim-by-Claim Analysis⁴

1. Claim 23

23[a]

This limitation is obvious in view of Sinclair-II. EX1003, ¶292. Sinclair-II teaches that if a defective block is detected in a metablock [**detect a defect in a given erase block**] a different block is responsively substitute [**responsive to detecting...**]. EX1003, ¶292; EX1015, ¶77.

A POSITA would also understand that detection of a defective block “during the lifetime of the memory” discloses that the detection may be “*in connection with physical erasure of the one or more erase blocks.*” EX1003, ¶293.

To the extent that Sinclair-II does not teach detecting defective blocks in connection with an erasure, it is obvious in view of Yu’s disclosure of “a process for detection and handling of bad pages and bad blocks” where “*[a]fter the block is erased, the status of each page in that block is checked*” to determine if it is bad. EX1003, ¶294; EX1016, ¶¶241-243.

A POSITA would have been motivated to apply Sinclair-II and Yu’s teachings to Sinclair and would have had a reasonable expectation of success. EX1003, ¶295. Sinclair and Sinclair-II are similar references from the same inventor and assignee. The two reference describe a flash memory device with a similar architecture.

⁴ See Table of Claims, *supra*, for full text of limitations.

EX1003, ¶295. While Sinclair does not discuss its bad block management, a POSTIA would recognize the importance of detecting bad blocks and that blocks may go bad, and that it reflects the application of a known technique to a known device (Sinclair’s flash memory) that is ready for improvement. EX1003, ¶295. A POSITA would also have recognized that integrating bad block detection with erasure improves efficiency and accuracy. EX1003, ¶296.

23[b]

Sinclair-II discloses “the substitution of a redundant block for a defective block originally assigned to the metablock” [**a logical address which has been remapped responsive to detection of the defect, to a storage location associated with the different erase block**]. EX1003, ¶298; EX1015, ¶77. “When the host writes data to the memory system, the controller converts the logical addresses assigned by the host to physical addresses within a metablock” for the substituted block [**direct ones of the write requests from the host, for which the accompanying address designates a logical address which has been remapped**]. *Id.* A POSITA would have been motivated to combined Sinclair and Sinclair-II for the reason discussed above. EX1003, ¶298; *see* 23[a].

XV. MANDATORY NOTICES AND FEES

A. Real Party-In-Interest

The real party-in-interest for Petitioner is Samsung.

B. Related Matters

Radian Memory Systems LLC, v. Samsung Electronics Co., Ltd. et al., Case No. 2:24-cv-1073 (E.D. Tex.).

C. Counsel and Service Information

Petitioner provides the following counsel and service information. Petitioner consents to electronic service the email addresses listed in the table below. Pursuant to 37 C.F.R. § 42.10(b), a Power of Attorney accompanies this Petition.

LEAD COUNSEL	BACK-UP COUNSEL
James M. Glass (Reg. No. 46,729) jimglass@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 295 Fifth Avenue, 9th Floor New York, NY 10016 Tel: (212) 849-7000 Fax: (212) 849-7100	John T. McKee (Reg. No. 65,926) johnmckee@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 295 Fifth Ave. New York, NY 10016 Tel: (212) 849-7000 Fax: (212) 849-7100 Luiz Miranda (Reg. No. 74,762) QUINN EMANUEL URQUHART & SULLIVAN, LLP 191 N. Wacker. Dr Suite 2700 luizmiranda@quinnemanuel.com Tel: (312) 705-7400 Fax: (312) 705-7401

D. Payment of Fees

The undersigned authorizes the Office to charge the fee required for this Petition for *Inter Partes* Review to Deposit Account No. 50-5708. Any additional fees that might be due are also authorized.

XVI. CONCLUSION

For these reasons, Petitioner requests *inter partes* review of the '801 patent.

DATED: July 28, 2025

Respectfully submitted,

By /s/ John T. McKee

Reg. No. 65,926

QUINN EMANUEL URQUHART &

SULLIVAN, LLP

295 Fifth Ave.

New York, NY 10016

Tel: (212) 849-7000

Fax: (212) 849-7100

Attorney for Petitioners

CERTIFICATE OF WORD COUNT

Pursuant to 37 C.F.R. §42.24(d), Petitioner hereby certifies, in accordance with and reliance on the word count provided by the word-processing system used to prepare this Petition, that the number of words in this paper is 13,530. Pursuant to 37 C.F.R. §42.24(d), this word count excludes the table of contents, table of authorities, mandatory notices under §42.8, certificate of service, certificate of word count, appendix of exhibits, and any claim listing.

Date: July 28, 2025

Respectfully submitted,

By: /s/ John T. McKee
Reg. No. 65,926

Attorney for Petitioners

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. §§ 42.6(e), 42.105(a), the undersigned hereby certifies service on the Patent Owner of a copy of this Petition and its respective exhibits at the official correspondence address for the attorneys of record for the '801 patent as shown in USPTO P-TACTS via FedEx:

Radian Memory Systems, LLC
6275 West Plano Parkway
Suite 500
Plano, TX 75093

Courtesy copies were also sent via electronic mail to Patent Owner's counsel of record in the related district court proceeding:

andrea@millerfairhenry.com
radian@caldwellcc.com

DATED: July 28, 2025

Respectfully submitted,

By: /s/ John T. McKee
Reg. No. 65,926

Attorney for Petitioners