

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SAMSUNG ELECTRONICS CO., LTD. AND
SAMSUNG ELECTRONICS AMERICA, INC.,
Petitioners,

v.

RADIAN MEMORY SYSTEMS, LLC,
Patent Owner.

Case IPR2025-01378

U.S. Patent No. 11,347,657

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

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. STANDING	2
III. TECHNICAL BACKGROUND	2
IV. OVERVIEW OF THE '657 PATENT	3
A. Priority	3
B. Prosecution History	3
C. Disclosure	4
V. GROUNDS	5
VI. LEVEL OF ORDINARY SKILL	5
VII. CLAIM CONSTRUCTION	6
VIII. GROUND I: CLAIMS 1-4, 6-9, 13, 15, AND 19-22 ARE RENDERED OBVIOUS BY REITER IN LIGHT OF ZBC, SBC-4, AND SPC-4	6
A. Reiter	6
B. ZBC, SBC-4, And SPC-4	7
C. Motivation to Combine	12
D. Claim-by-Claim Analysis	14
1. Claim 1	14
(a) 1[pre]	14
(b) 1[a]	15
(c) 1[b]	16
(d) 1[c]	20
(e) 1[c][i]	20
(f) 1[c][ii]	22
(g) 1[c][iii]	23
(h) 1[c][iv]	26
(i) 1[d]	30

	(j) 1[e]	30
2.	Claim 2	31
	(a) 2[pre]	31
	(b) 2[a]	31
	(c) 2[b]	32
3.	Claim 3	32
	(a) 3[pre]	32
	(b) 3[a]	33
	(c) 3[b]-[c]	36
4.	Claim 4	36
	(a) 4[pre]	36
	(b) 4[a]	36
5.	Claim 6	38
	(a) 6[pre]	38
	(b) 6[a]	38
6.	Claim 7	41
	(a) 7[pre]	41
	(b) 7[a]	41
7.	Claim 8	42
	(a) 8[pre]	42
	(b) 8[a]	42
8.	Claim 9	43
	(a) 9[pre]	43
	(b) 9[a]	43
9.	Claim 13	44
	(a) 13[pre]	44
	(b) 13[a]	44
	(c) 13[b]	44
	(d) 13[c]	45

10.	Claim 15.....	48
	(a) 15[pre].....	48
	(b) 15[a].....	48
	(c) 15[b].....	48
	(d) 15[b][i].....	48
	(e) 15[b][ii]-[iii]	49
11.	Claim 19-21.....	49
12.	Claim 22.....	50
	(a) 22[pre]:	50
	(b) 22[a].....	50
IX.	GROUND II: CLAIMS 5, 10-13, AND 18 ARE RENDERED OBVIOUS BY REITER IN LIGHT OF ZBC, SBC-4, SPC-4, AND SINCLAIR-367.....	51
A.	Sinclair-367	51
B.	Motivation to Combine	53
C.	Claim-by-Claim Analysis.....	54
1.	Claim 5.....	54
	(a) 5[pre].....	54
	(b) 5[a].....	54
2.	Claim 10.....	57
	(a) 10[pre].....	57
	(b) 10[a].....	57
	(c) 10[b].....	58
	(d) 10[c].....	59
	(e) 10[d].....	60
3.	Claim 11	60
	(a) 11[pre].....	60
	(b) 11[a]	60
	(c) 11[b]-[c]	60
4.	Claim 12.....	61

	(a)	12[pre].....	61
	(b)	12[a].....	61
	(c)	12[b].....	61
	(d)	12[c]-[e]	62
5.	Claim 13		62
	(a)	13[pre]-[b].....	62
	(b)	13[c]	63
6.	Claim 18		63
	(a)	18[pre].....	63
	(b)	18[a]	63
	(c)	18[b].....	64
X.	GROUND III: CLAIMS 14 AND 16-17 ARE RENDERED OBVIOUS BY REITER IN LIGHT OF ZBC, SBC-4, SPC-4, AND YAMADA		66
A.	Yamada.....		66
B.	Motivation to Combine		68
C.	Claim-by-Claim Analysis.....		69
	1.	Claim 14	69
		(a)	14[pre].....69
		(b)	14[a]
		(c)	14[b].....69
		(d)	14[c]
	2.	Claim 16	70
		(a)	16[pre].....70
		(b)	16[a]
		(c)	16[b].....70
		(d)	16[b][i]
		(e)	16[b][ii].....71
	3.	Claim 17	73
		(a)	17[pre].....73

	(b)	17[a].....	73
	(c)	17[b]-[c].....	73
XI.	GROUND IV: CLAIMS 1-2, 8-9, 18-22 ARE RENDERED OBVIOUS BY SINCLAIR-233		74
A.	Overview of the Prior Art.....		74
	1.	Sinclair-233	74
B.	Claim-by-Claim Analysis.....		76
	1.	Claim 1	76
	(a)	1[pre].....	76
	(b)	1[a]	76
	(c)	1[b].....	76
	(d)	1[c]	78
	(e)	1[c][i]	78
	(f)	1[c][ii]	79
	(g)	1[c][iii]	79
	(h)	1[c][iv]	81
	(i)	1[d].....	84
	(j)	1[e]	84
	2.	Claim 2	85
	(a)	2[pre].....	85
	(b)	2[a]	85
	(c)	2[b].....	86
	3.	Claim 8	86
	(a)	8[pre].....	86
	(b)	8[a]	87
	4.	Claim 9	88
	(a)	9[pre].....	88
	(b)	9[a]	88
	5.	Claim 18	89

(a)	18[pre].....	89
(b)	18[a].....	89
(c)	18[b].....	90
6.	Claim 19-21.....	91
7.	Claim 22.....	91
(a)	22[pre].....	91
(b)	22[a].....	92
XII.	MANDATORY NOTICES AND FEES	92
A.	Real Party-In-Interest	92
B.	Related Matters.....	92
C.	Counsel and Service Information.....	93
D.	Payment of Fees	93
XIII.	CONCLUSION.....	93
	CERTIFICATE OF WORD COUNT.....	95
	CERTIFICATE OF SERVICE	96

EXHIBIT LIST

Ex.	Description
1001	U.S. Patent No. 11,347,657 (“’657 Patent”)
1002	Prosecution History of U.S. Patent No. 11,347,657
1003	Expert Declaration of Dr. Heiner Litz.
1004	U.S. Patent No. 8,301,861 (“Reiter”)
1005	U.S. Patent 7,984,233 (“Sinclair-233”)
1006	Working Draft Revision 1a of the Small Computer System Interface Zoned Block Commands (“ZBC”)
1007	Working Draft Revision 2 of SCSI Block Commands – 4 (“SBC-4”)
1008	Working Draft Revision 37 of SCSI Primary Commands – 4 (“SPC-4”)
1009	U.S. Publication No. 2005/0144367 (“Sinclair-367”)
1010	U.S. Publication No. 2011/0271032 (“Yamada”)
1011	Patent Owner’s District Court Infringement Chart for ’657 Patent
1100	Declaration of Rich Deglin
1101	“T10 Working Drafts”, T10, http://www.t10.org/drafts.htm (last visited Jul. 17, 2025)
1102	“File Access Monitor”, T10, https://www.t10.org/cgi-bin/ac.pl?t=f&f=sbc5r08.pdf (last visited Jul. 17, 2025)
1103	“T10 Working Drafts”, T10 (archived Jul. 15, 2014), https://web.archive.org/web/20140715124727/http://www.t10.org/drafts.htm (last visited Jul. 17, 2025)
1104	“File Access Monitor”, T10 (archived May 19, 2014), https://web.archive.org/web/20140519064154/http://www.t10.org/cgi-bin/ac.pl?t=f&f=spc4r37.pdf (last visited Jul. 17, 2025)
1105	“File Access Monitor”, T10 (archived Apr. 16, 2015), https://web.archive.org/web/20150416184431/http://www.t10.org/cgi-bin/ac.pl?t=f&f=sbc4r02.pdf (last visited Jul. 17, 2025)
1106	“File Access Monitor”, T10 (archived May 19, 2014), https://web.archive.org/web/20140519063435/http://www.t10.org/cgi-bin/ac.pl?t=f&f=zbc-r01a.pdf (last visited Jul. 17, 2025)

Ex.	Description
1107	May 17, 2014 Email From T10
1108	May 24, 2014 Email From T10
1109	“Introduction to T10”, T10, https://www.t10.org/intro.htm (last visited Jul. 20, 2025)
1200	“Google Patents Search”, Google, https://patents.google.com/?q=SCSI&q=%22Small+Computer+System+Interface%22&before=publication:20140901&num=100 (last visited Jul 7, 24, 2025) (searching for “SCSI” or “Small Computer System Interface” patent publications prior to Sep. 1, 2014)
1201	Somayaji & Murty, “Securing Internet Protocol (IP) Storage: A Case Study” (2009)
1202	White et al., “Performance Comparison of IDE and SCSI” (2001)
1203	L. Columbus, “Exploring the World of SCSI” (2000)
1204	U.S. Publication No. 2014/0215147
1205	U.S. Publication No. 2009/0271659
1206	James S. Plank, “The RAID-6 Liberation Codes” (2008)
1207	U.S. Patent No. 5,857,208
1208	Patterson et al., “A Case for Redundant Arrays of Inexpensive Disks (RAID)” (1998)
1209	Chen et al., “RAID: High-Performance, Reliable Secondary Storage” (Jun. 1994)
1210	U.S. Publication No. 2009/0271562
1211	U.S. Patent No. 5,404,485
1212	U.S. Pat. Pub. 2008/0189485A1
1213	Mark Jones Deposition Transcript (May 13, 2025)

CLAIM LISTING

Claim 1	
1[pre]	A storage device comprising:
1[a]	flash memory comprising erase units; and
1[b]	circuitry to receive from a host, and control execution of, incoming requests including data write requests and erase requests, wherein each of the incoming requests is accompanied by incoming address information;
1[c]	wherein said circuitry is to, for each of the incoming requests:
1[c][i]	derive, from the accompanying incoming address information, a first address portion and a second address portion
1[c][ii]	identify an addressed block device from the first address portion,
1[c][iii]	use a first division operation on the second address portion, to identify an addressed segment within the addressed block device, wherein the addressed segment corresponds to a subset of the erase units of the flash memory, and
1[c][iv]	for each of the data write requests, identify from the first division operation a third address portion, and use a second division operation on the third address portion to identify an address value to select one of the erase units from the subset which corresponds to the addressed segment and to identify a storage location within the selected one of the erase units;
1[d]	wherein said circuitry is to execute each one of the data write requests by programming accompanying data into the storage location that was identified for that one of the data write requests; and
1[e]	wherein said circuitry is also to execute each one of the incoming erase requests by controlling erasure in the flash memory of one or more of the erase units in the subset corresponding to the addressed segment that was identified from the incoming address information accompanying the one of the incoming erase requests.

Claim 2	
2[pre]	The storage device of claim 1 wherein:
2[a]	said circuitry comprises at least one processor; and
2[b]	the storage device further comprises instructions stored on at least one physical storage medium, said instructions when executed to control the at least one processor.
Claim 3	
3[pre]	The storage device of claim 1 wherein:
3[a]	said circuitry can receive a request from a host to copy host-selected data from one storage location in flash memory to another storage location in the flash memory, and is to issue commands to the flash memory, in response to the request from the host to copy, to read the host-selected data from a first physical storage location in the flash memory and write the host-selected data to a second physical storage location in the flash memory without sending the data being copied to the host;
3[b]	the request from the host to copy is to designate destination address information corresponding to the second physical storage location; and
3[c]	the circuitry is to use the first division operation on the designated destination address information to identify a destination segment within a destination block device.
Claim 4	
4[pre]	The storage device of claim 3 wherein
4[a]	the circuitry is to identify from the destination address information a destination logical erase unit and is to select the second physical storage location to be within a physical erase unit that is already mapped to the destination logical erase unit.
Claim 5	

5[pre]	The storage device of claim 1 wherein
5[a]	the storage device is to store information identifying a number of structures in the flash memory and is to transmit the information identifying the number of structures to a host, and wherein the storage device is to store the values representing a time since data was programmed on a basis that is respective to each structure of the number of structures, wherein each structure comprises at least one of a page or a unit of memory comprising one or more erase units of the flash memory, and wherein each value represents time since data was programmed into the respective structure.
Claim 6	
6[pre]	The storage device of claim 1 wherein
6[a]	the storage device is to store information identifying a number of structures in the flash memory and is to transmit the information identifying the number of structures to a host, and wherein the circuitry is to fulfill the at least one of the incoming requests by issuing commands to the flash memory for which a quantum of flash memory selected by the commands which are issued in response to the at least one request is dependent on the information transmitted to the host identifying the number of structures.
Claim 7	
7[pre]	The storage device of claim 1 wherein
7[a]	the storage device is to store information identifying a number of structures in the flash memory and is to transmit to a host information that identifies an amount of flash memory that is to be erased as a unit.
Claim 8	
8[pre]	The storage device of claim 1 wherein
8[a]	the third address portion specifies a location within a sequential range of locations that correspond to the erase units in the subset.

Claim 9	
9[pre]	The storage device of claim 8 wherein
9[a]	the first division operation and the second division operation each subdivide a value obtained from the incoming address information accompanying each of the write requests into respective bitfields.
Claim 10	
10[pre]	The storage device of claim 1 wherein
10[a]	the storage device is to store, for each segment having an erase unit which stores valid data, a value representing a time since data was programmed;
10[b]	the storage device is to compare the values representing time since data was programmed to at least one threshold;
10[c]	upon detection that a specific one of the values has satisfied the one or more thresholds, the circuitry is to copy data stored at one or more physical storage locations associated with the specific one of the values from a first physical erase unit to a second physical erase unit; and
10[d]	the storage device is to remap a logical address for the data being copied from the first physical erase unit to the second physical erase unit.
Claim 11	
11[pre]	The storage device of claim 10 wherein:
11[a]	the flash memory is to have block devices, one of the block devices being the addressed block device; and
11[b]	the storage device is to identify, for the data being copied, an address of a logical erase unit, the logical erase unit being associated with the addressed block device to the exclusion of each other one of the block devices; and
11[c]	the storage device is to remap a correspondence between the address of the logical erase unit and the first physical erase unit to instead be

	between the address of the logical erase unit and the second physical erase unit.
Claim 12	
12[pre]	The storage device of claim 1 wherein:
12[a]	the flash memory is to have block devices, including the addressed block device; and
12[b]	the storage device is to store metadata on an independent basis for portions of flash memory associated with each one of the block devices, wherein the stored metadata is dependent on an erase count of associated physical memory space;
12[c]	the storage device is to compare the metadata to at least one threshold;
12[d]	upon detection that specific metadata has satisfied the one or more thresholds, the circuitry is to copy data stored at one or more physical storage locations associated with the specific metadata from a first physical erase unit to a second physical erase unit; and
12[e]	the storage device is to remap a logical erase unit value associated with the third address portion from the first physical erase unit to the second physical erase unit.
Claim 13	
13[pre]	The storage device of claim 1 wherein:
13[a]	the erase units of the flash memory are physical erase units;
13[b]	the circuitry is to use the second division operation to identify a logical erase unit, wherein each logical erase unit is mapped to one of the physical erase units; and
13[c]	the storage device is to detect a failure condition of the physical erase units and is to remap each logical erase unit which corresponds to one of the physical erase units for which the failure condition is detected to instead correspond to one of the physical erase units for which the failure condition has not been detected and is to utilize the remapped

	correspondence to service one or more of the incoming data access requests.
Claim 14	
14[pre]	The storage device of claim 1 wherein the storage device is to:
14[a]	store, for each segment having an erase unit which stores valid data, a value representing a time since data was programmed;
14[b]	compare the values representing time since data was programmed to at least one threshold; and
14[c]	responsive to detection of a condition that a specific one of the values representing time since data was programmed has satisfied the one or more thresholds, transmit information to a host wherein such transmitted information conveys address information for data stored at one or more physical storage locations associated with the specific one of the values.
Claim 15	
15[pre]	The storage device of claim 1 wherein:
15[a]	the flash memory is to have block devices, including the addressed block device; and
15[b]	the storage device is further to:
15[b][i]	store metadata on an independent basis for portions of flash memory associated with each one of the block devices,
15[b][ii]	receive a query, issued by a host, directed to a specific one of the block devices, and
15[b][iii]	responsive to the query, compare the metadata to at least one threshold, to identify specific metadata, and transmit information to the host wherein such transmitted information conveys address information for data stored at one or more physical storage locations associated with the specific metadata.
Claim 16	

16[pre]	The storage device of claim 1 wherein:
16[a]	the flash memory is to have block devices, including the addressed block device; and
16[b]	the storage device is further to:
16[b][i]	store metadata on an independent basis for portions of flash memory associated with each one of the block devices, wherein the stored metadata represents an erase count of associated physical memory space, and
16[b][ii]	receive a query, issued by a host, wherein said query is specific to one of the block devices, and responsive to the query, identify specific metadata, associated with the specified one of the block devices, and transmit information to the host that conveys the specific metadata.
Claim 17	
17[pre]	The storage device of claim 1 wherein the storage device is further to:
17[a]	store metadata on an independent basis for portions of flash memory associated with each one of the block devices, wherein the stored metadata represents an erase count of associated physical memory space;
17[b]	compare the metadata to at least one threshold; and
17[c]	responsive to detection that specific metadata has satisfied the one or more thresholds, transmit information to a host wherein such transmitted information conveys address information for data stored at one or more physical storage locations associated with the specific metadata.
Claim 18	
18[pre]	The storage device of claim 1 wherein:
18[a]	the flash memory comprises multiple planes; and
18[b]	the circuitry is to, for each one of a set of the data write requests having sequential addressing, identify a commonly addressed page and concurrently write data accompanying the set of write requests

	into respective ones of the multiple planes in a manner corresponding to the commonly addressed page.
Claim 19	
19[pre]	A storage device comprising:
19[a]	flash memory comprising erase units; and
19[b]	circuitry to receive from a host, and control execution of, incoming requests including data write requests and erase requests, wherein each of the incoming requests is accompanied by incoming address information;
19[c]	wherein said circuitry is to, for each of the incoming requests:
19[c][i]	derive, from the accompanying incoming address information, a first address portion and a second address portion,
19[c][ii]	identify an addressed block device from the first address portion,
19[c][iii]	use a first operation on the second address portion to subdivide the incoming address information into first bit fields, and to identify an addressed segment within the addressed block device from a first one of the first bit fields, wherein the addressed segment corresponds to a subset of the erase units of the flash memory, and
19[c][iv]	for each of the data write requests, use a second one of the first bit fields as a third address portion and use a second operation to subdivide the third address portion into second bit fields, to identify an address value to select one of the erase units from a first one of the second bit fields and to identify a storage location within the selected one of the erase units from a second one of the second bit fields;
19[d]	wherein said circuitry is to execute each one of the data write requests by programming accompanying data into the storage location that was identified for that one of the data write requests;
19[e]	wherein said circuitry is also to execute each one of the incoming erase requests by controlling erasure in the flash memory of one or more of the erase units in the subset corresponding to the addressed

	segment that was identified from the incoming address information accompanying the one of the incoming erase requests.
Claim 20	
20[pre]	A storage device comprising:
20[a]	flash memory, the flash memory to have one or more block devices at a hierarchically-superior level, segments within each of the block devices at a hierarchically-intermediate level, and addressable storage locations within each of the segments at a hierarchically-inferior level, each segment corresponding to a respective, mutually exclusive set of one or more erase units, each segment being associated with only one of the block devices, each erase unit being associated with only one of the segments; and
20[b]	circuitry to receive from a host, and control execution of, incoming requests including data write requests and erase requests, wherein each of the incoming requests is accompanied by incoming address information;
20[c]	wherein said circuitry is to, for each of incoming requests:
20[c][i]	derive a first address portion and a second address portion,
20[c][ii]	identify an addressed one of the block devices from the first address portion,
20[c][iii]	use a first operation to subdivide the second address portion into first bit fields, and to identify an addressed segment within the addressed block device from a first one of the first bit fields, and to identify a third address portion from a second one of the first bit fields, wherein the addressed segment corresponds to a subset of the erase units of the flash memory, and
20[c][iv]	for each of the data write requests, use a second operation to subdivide the third address portion into second bit fields, to identify an address value to select one of the erase units from a first one of the second bit fields, and to identify a storage location within the selected one of the erase units from a second one of the second bit fields;

20[d]	wherein said circuitry is to execute each one of the data write requests by programming accompanying data into the storage location that was identified for that one of the data write requests;
20[e]	wherein said circuitry is also to execute each one of the incoming erase requests by controlling erasure in the flash memory of one or more of the erase units in the subset corresponding to the addressed segment that was identified from the incoming address information accompanying the one of the incoming erase requests.
Claim 21	
21[pre]	A storage device comprising:
21[a]	flash memory, the flash memory to have one or more block devices at a hierarchically-superior level, segments within each of the block devices at a hierarchically-intermediate level, and addressable storage locations within each of the segments at a hierarchically-inferior level, each segment corresponding to a respective, mutually exclusive set of one or more erase units, each segment being associated with only one of the block devices, each erase unit being associated with only one of the segments; and
21[b]	circuitry to receive from a host, and control execution of, incoming requests including data write requests and erase requests, wherein each of the incoming requests is accompanied by incoming address information;
21[c]	wherein said circuitry is to, for each of the incoming requests:
21[c][i]	derive, from the accompanying incoming address information, a first address portion and a second address portion,
21[c][ii]	identify an addressed one of the block devices from the first address portion,
21[c][iii]	use a first operation to subdivide the second address portion into first bit fields, and to identify an addressed segment within the addressed block device from a first one of the first bit fields and a third address portion from a second one of the first bit fields, wherein the addressed

	segment corresponds to a subset of the erase units of the flash memory, and
21[c][iv]	for each of the data write requests, use a second operation to subdivide the third address portion into second bit fields, to identify, from a first one of the second bit fields, an address value to select one of the erase units and, from a second one of the second bit fields a storage location within the selected one of the erase units;
21[d]	wherein said circuitry is to execute each one of the data write requests by programming accompanying data into the storage location that was identified for that one of the incoming data write requests; and
21[e]	wherein said circuitry is also to execute each one of incoming erase requests by controlling erasure in the flash memory of the one or more erase units in the subset corresponding to the addressed segment that was identified from the incoming address information accompanying the one of the incoming erase requests.
Claim 22	
22[pre]	The storage device of claim 1 wherein
22[a]	the second address portion comprises a first set of bits and a second set of bits, the bits of the first set of bits being mutually-exclusive to the bits of the second set of bits, wherein the first division operation is to separate the first set of bits from the second set of bits, and the circuitry is to identify the addressed segment from the first set of bits.

I. INTRODUCTION

Samsung Electronics Co., Ltd. and Samsung Electronics America, Inc., (“Samsung”) respectfully requests *inter partes* review of claims 1-22 of U.S. Patent No. 11,347,657 (“’657 Patent”). A reasonable likelihood exists that Petitioner will prevail on at least one challenged claim.

The ’657 Patent relates to systems and methods for a flash memory controller with a logical-to-physical address translation. EX1001, FIG. 1. The host system (*e.g.*, PC connected to the flash SSD) provides the controller a logical address. EX1001, FIG. 1. The controller translates the logical address to a physical address that details the actual location in flash memory. EX1001, FIG. 1. The ’657 Patent purports to disclose controller for a flash memory storage device with “negligible logical-to-physical address translation time,” where “address trans[lation] may be implemented entirely with logical gates and look-up tables of a memory controller integrated circuit, without requiring processor cycles.” EX1001, Abstract. The ’657 Patent allegedly achieves this improvement with “hierarchal address virtualization,” which means the logical address is translated to sub-addresses associated with virtual flash memory structures (*e.g.*, virtual erase unit and virtual page), and each virtual sub-address is translated to the physical address for the corresponding memory structure (*e.g.*, erase unit and page). EX1001, Abstract.

The '657 Patent represents common methods and techniques applied to address translation for flash memory. The claimed novelty, virtual hierarchal address virtualization, was known in the art prior to the effective filing date of the '657 Patent. As an example, Reiter teaches a flash memory controller that translates a logical block address (hereinafter "LBA") from the host to a logical block number (*i.e.*, erase unit number) and logical page number, which are used to determine the physical erase unit number and physical page number. EX1004, 8:54-9:25, FIG. 4A.

The '657 Patent lacks any inventive concept and for the reasons herein is rendered obvious by Grounds I-IV.

II. STANDING

Petitioner certifies under 37 C.F.R. §42.104(a) that the '657 Patent is available for *inter partes* review and that Petitioner is not barred or estopped from requesting an *inter partes* review on the grounds set forth herein.

III. TECHNICAL BACKGROUND

Flash memory is organized into discrete units called erase units or blocks. EX1001, 4:10-35. Blocks are comprised of pages. *Id.* Pages represent the smallest unit that may be written. *Id.* To write the pages of a block with new data, the block must first be erased. *Id.*

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

spanning multiple planes operated as a single unit, termed, *e.g.*, “metablocks,” EX1009, ¶[0012].

To increase the longevity of flash memory, SSDs perform management operations including wear-leveling and other maintenance where data may be copied, moved or erased to prolong data storage life. EX1004, 4:4-37. To identify data for maintenance, metadata is tracked describing the history of the units of memory. *Id.*

IV. OVERVIEW OF THE '657 PATENT

A. Priority

The '657 Patent issued on May 31, 2022, and claims priority to Provisional Application No. 61/048,162, filed on September 9, 2014.

The prior art references relied upon in this Petition predate September 9, 2014, the earliest possible critical date.

B. Prosecution History

The nonprovisional application that became the '657 Patent was filed on September 20, 2021.

The examiner issued a non-final rejection on January 4, 2022. EX1002. The examiner rejected claims 1-22 under 35 U.S.C. § 112(b). EX1002. Additionally, claim 2 was rejected under 35 U.S.C. 101. EX1002. On March 30, 2022, Applicant amended the claims to traverse the rejections. EX1002. On April 20, 2022, the

examiner issued a Notice of Allowance. EX1002. The '657 Patent issued May 31, 2022.

C. Disclosure

The '657 Patent describes a logical-to-physical address translation scheme for a memory controller taking commands from a host to manage flash memory (*i.e.*, SSD). EX1001, Abstract, 2:66-3:20.

The host system sends a command, such as a read command, with accompanying address information to the controller. *Id.* The controller translates the address information to sub-addresses associated with virtual structures of the flash memory. EX1001, 2:66-3:21. Each virtual sub-address is translated to a physical sub-address in the underlying flash memory structure. *Id.* For instance, in one embodiment a logical address from the host received by the controller is made up of several bit fields (or “sub-addresses”), including bit fields associated with a virtual channel number, virtual die number, virtual erase unit number, and virtual page number. EX1001, 4:23-35, 15:55-65. The logical address is broken up into those bit fields. *Id.* Each virtual sub-address is translated by look-up tables to a physical sub-address corresponding to the same flash memory structure (*i.e.*, channel, die, erase unit, and page) and identifying the location in physical flash memory. EX1001, 15:37-65, 16:49-65. The '657 Patent refers to this scheme as “hierarchical address virtualization.” EX1001, Abstract.

V. GROUNDS¹²

Ground	Basis	Reference(s)	Claims
I	§ 103	Reiter, ZBC, SBC-4, SPC-4	1-4, 6-9, 13, 15, 19-22
II	§ 103	Reiter, ZBC, SBC-4, SPC-4, Sinclair-367	5, 10-13, 18
III	§ 103	Reiter, ZBC, SBC-4, SPC-4, Sinclair-367, Yamada	14, 16-17
IV	§ 103	Sinclair-233	1-2, 8-9, 18-22

VI. LEVEL OF ORDINARY SKILL

A person of ordinary skill in the art (“POSITA”) at the time of the ’657 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, ¶¶13-16, 26-28. More education can supplement practical experience and vice versa. *Id.* Petitioner’s expert exceeded this skill level as of the priority date. *Id.*

¹ None of these references relied upon by this Petition were before the Examiner, much less included in a rejection, cited during prosecution.

² Petitioners are not presently aware of any secondary considerations of non-obviousness applicable to the ’657 Patent. In the co-pending litigation, Radian has objected to identifying secondary considerations for specific patents and believes it need only “generally address secondary considerations” without any particularity or showing of nexus. Radian has also marked its response as confidential such that Petitioner cannot reproduce its response here.

VII. CLAIM CONSTRUCTION

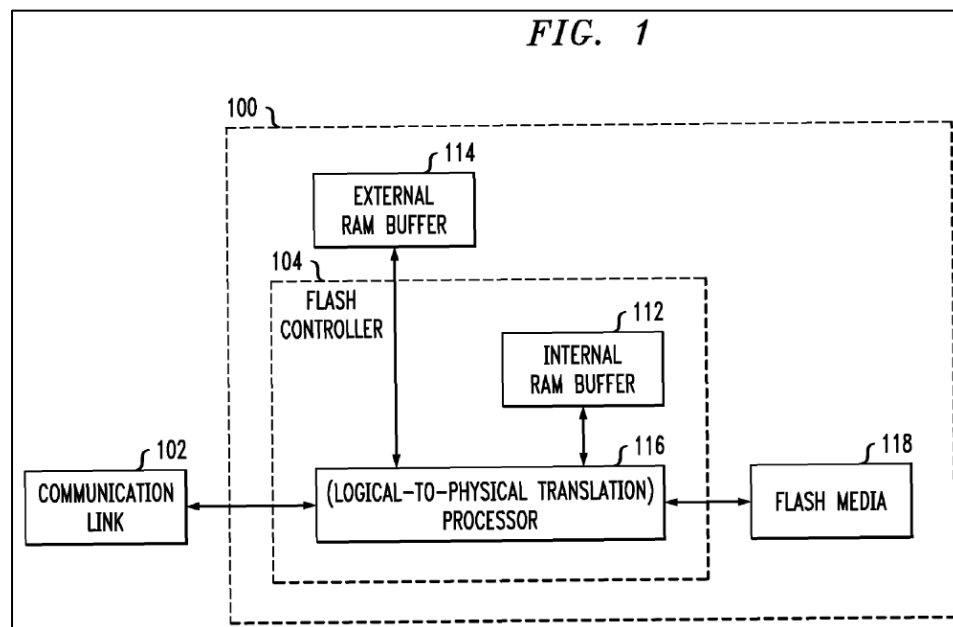
For the purposes of this proceeding, Petitioner contends that claim terms be given their plain and ordinary meaning under *Philips*.

VIII. GROUND I: CLAIMS 1-4, 6-9, 13, 15, AND 19-22 ARE RENDERED OBVIOUS BY REITER IN LIGHT OF ZBC, SBC-4, AND SPC-4

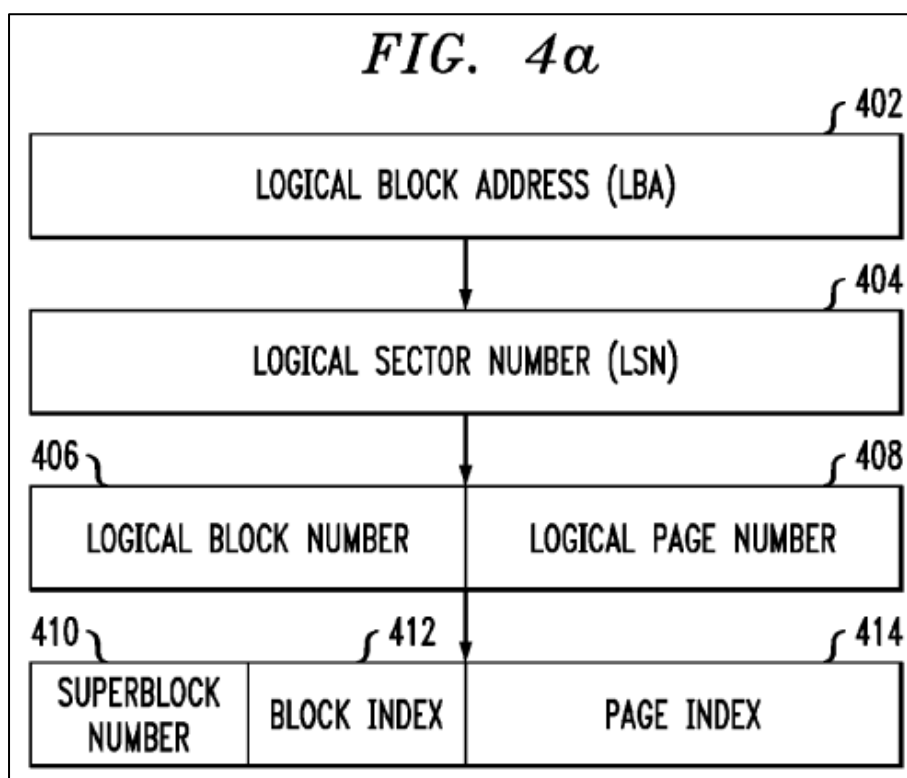
A. Reiter

U.S. Patent 8,301,861 (“Reiter”) was filed on April 29, 2010, was published on March 24, 2011, issued on October 30, 2012, and is prior art under 35 U.S.C. §102(a)(1).

Reiter discloses a flash system. EX1004, 6:63-66. The flash system uses a flash controller to receive and execute a host’s read/write commands sent over a communication link. EX1003, 45-47; EX1004, 8:1-6, FIG. 1:



The controller performs logical-to-physical translation on the accompanying logical address and executes those commands on flash media at the resulting physical address. EX1004, 7:31-35, 8:1-10. Reiter's addressing scheme includes (1) an LBA, (2) logical sector number, (3) logical block number and logical page number, and (4) superblock number, block index, and page index. EX1004, 3:41-60, FIG. 4a:



B. ZBC, SBC-4, And SPC-4

ZBC, SBC-4, and SPC-4 (collectively, “T10 Standards”) are printed publications. EX1003, ¶¶59-64 As set forth in Rich Deglin’s declaration, T10 Standards qualify because of publication on the website of T10, an INCITS Technical Committee responsible for SCSI architecture standards, as of May 15,

2014, May 20, 2014, and May 17, 2014, respectively. EX1100, ¶¶2-10; EX1103; *see also* EX1109 (depicting T10’s website). T10 Standards are working drafts that were made available to both T10 members and guests (i.e., the public at large) for download on T10’s website, as shown in the archived versions. EX1100, ¶11; *see also Valve Corporation v. Ironburg Inventions Ltd.*, 8 F.4th 1364, 1374 (Fed. Cir. Aug. 17, 2021) (establishing public availability with archive.org).

The website’s well-known nature is confirmed by Dr. Litz. EX1003, ¶¶59-60. Dr. Litz was aware of T10 and the SCSI standards published by T10, and that T10 Standards would have been readily available upon publication on T10’s website. EX1003, ¶. T10 “was well known to the community interested in the subject matter of” T10 Standards, and a POSITA “interested in” storage standards “would have been independently aware of [T10] as a prominent forum for discussing such technologies.” *Voter Verified, Inc. v. Premier Election Sols., Inc.*, 698 F.3d 1374, 1380 (Fed. Cir. 2012); EX1003, ¶¶59-60.

Moreover, a POSITA could have reasonably located T10 Standards on T10’s website. Via a “Drafts” hyperlink at the top of the website, (*see, e.g.*, EX1109.) the website has a “T10 Working Drafts” webpage linking a limited number of standards and working drafts all related to accessing/managing storage devices. *See* EX1103; *Roku, Inc. v. Universal Electronics, Inc.*, 2025 WL 1692663, at *2 (Fed. Cir. Jun. 17, 2025) (holding that a POSITA could reasonably locate a user manual on a

webpage amongst several hundred manuals because a POSITA could reasonably find the webpage in the website, and because the webpage's manuals were the "same subject matter as the invention"). The webpage also categorizes those documents into sections, and provides a table of contents with descriptive section names and hyperlinks to each section, including "Block Commands (ZBC)", "Block Commands (SBC)", and "Primary Commands (SPC)" leading to ZBC, SPC-4, and SPC-4, respectively. EX1103; *see Roku*, 2025 WL 1692663 at *2 (finding public availability for a user manual amongst other manuals on a website because the website's manuals "were ... sorted by subject matter"). Thus, an interested POSITA would have been aware of the availability of, and could easily locate, the T10 Standards on T10's website.

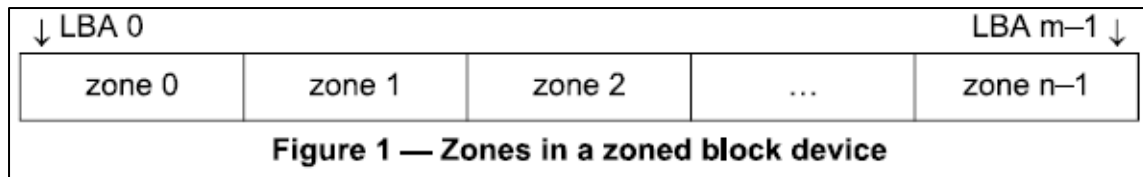
Even if T10 Standards' availability was limited to T10 members, that is no bar to publicly availability. The Board regularly finds documents published by standards bodies, including working drafts like T10 Standards, are publicly available. *See, e.g., ZTE Corp v. IPR licensing, Inc.* (IPR2014-00525) (crediting expert that GPRS Standards "were available to an interested member of the public as of the date listed on the top of the specification."); *M&K Holdings, Inc. v. Samsung Electronics Co., Ltd.*, 985 F.3d 1376, 1380–83 (Fed. Cir. 2021) (finding that three papers associated with an industry standard task force (JCT-VC), one being a working draft document, which were all posted on the organization's public

website publicly accessible prior art). In *Ericsson Inc. v. Intellectual Ventures I LLC*, IPR2014-00527, Paper No. 1, Petition at 7 (PTAB March 21, 2014), the Board found a paywalled IEEE article as publicly available, and that an IEEE copyright line on the first page of the reference sufficiently evidenced publication date and public accessibility. Here, T10 Standards bear T10 copyright dates and/or otherwise indicate “publi[cation].” EX1006, iv; EX1007, iv; EX1008, iv.

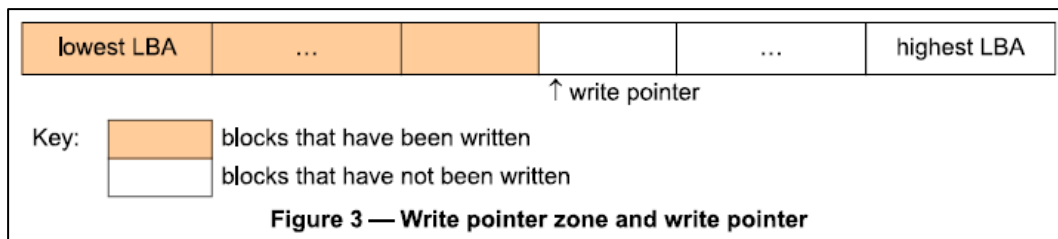
Moreover, T10 Standards are authentic. See EX1100, ¶¶1-17. Mr. Deglin personally received notice of publications and contemporaneously downloaded T10 Standards, and confirmed that the metadata matches the metadata listed on the T10 website as of the publication date. EX1100, ¶¶12-16. T10 Standards used in this Petition are the same those downloaded by Mr. Deglin.

SCSI is a commonly used standard for storage, such as hard disk drives. EX1003, ¶¶61-62. ZBC is a SCSI command set for a zoned block device model. EX1006, 1. The zoned block device model also accepts commands from core SCSI specifications such as SBC-4 and SPC-4. EX1006, x, 12-13, 22, Figure 0.

SCSI creates logical block devices, including logical units, volumes, and/or partitions (collectively, “logical units”), on physical storage devices. EX1008, 14. Each logical unit has an LBA range from zero to its highest LBA. EX1008, 14. When a logical unit is set as a zoned block device, the LBA range is divided into zones. EX1006, 1, 13, Figure 1:



ZBC allows for different types of zoned block devices including “host aware zoned block device[s]” and “host managed zoned block device.” EX1006, 4, 13. ZBC also supports different types of zones including a “sequential write preferred zone” and a “sequential write required zone.” EX1006, 14. These zones have write pointers, which generally track the next available LBA in the zone. EX1006, 14, 16-17, Figure 3:



The write pointer increments with each write command until the zone is full or a reset write pointer command is applied. EX1006, Table 15. A reset write pointer command resets the write pointer back to the lowest LBA in the zone, and resets the data in the zone to the manufacturer’s default initialization pattern. EX1006, 3, 15-18.

C. Motivation to Combine

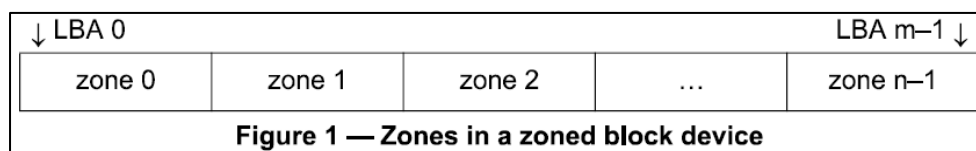
A POSITA would have been motivated to use ZBC, alongside SBC-4 and SPC-4, as the command set governing requests sent to Reiter's flash system. EX1003, ¶¶60-71.

As an initial matter, a POSITA would be motivated to apply the teachings of ZBC, SBC-4, and SPC-4. All three are SCSI specifications designed to be used together. EX1003, ¶¶60-66. ZBC is a "command set extension" of SCSI that builds on the core command sets defined in SBC-4 and SPC-4. EX1006, x, 1, 12-13, 22, Figure 0. ZBC expressly discloses being "implemented in conjunction with ... SPC-4," and SPC-4 is listed as "shared for all device types." EX1006, 1, Figure 0. ZBC also repeatedly relies on SBC-4. *E.g.*, EX1006, 3-4, 12, 22 ("[C]ommands for host aware zoned block devices are defined in SBC-4.").

A POSITA would further be motivated to apply ZBC's teachings (including SBC-4 and SPC-4) to Reiter's flash system. EX1003, ¶¶67-70. Reiter does not require a specific command set, but expressly discloses support for the widely used SCSI protocols. EX1004, 7:9-18. Reiter also discloses "receiv[ing] ... requests ... such as read or write operations from ... external devices," which "generally include a[] ... [LBA] where data should be read or written." EX1004, 8:1-6. A POSITA would thus be motivated to include support for commands such as those in ZBC to enable specific read and write operations based on Reiter's memory structure.

A POSITA would particularly be motivated to implement ZBC commands in Reiter’s system because SCSI protocols and commands such as those disclosed in ZBC are widely used, thereby improving interoperability. EX1003, ¶¶68, 70.

In addition, commands based on ZBC’s zoned block device model (EX1006, 1) would enable more effective utilization of Reiter’s memory organization scheme. EX1003, ¶¶67-70. ZBC is designed for zoned block devices, where address space comprising LBAs is “divided into ranges called zones.” EX1006, 13, 29, Table 16, Figure 1:



ZBC zones are analogous to Reiter’s superblocks, which also comprise “a logical collection of blocks representing a fixed range of LBAs.” EX1004, 9:11-13. Zones and superblocks both include metadata that indicate the next available memory location within the zone/superblock. EX1006, 14, 16-17, Figure 3 (illustrating a zone with a write pointer indicating next available LBA); EX1004, 9:52-10:1, FIG. 4b (illustrating superblock summary page with “free page pointer” indicating the next available memory location). And like ZBC, which uses LBAs to access different zones, Reiter’s superblocks are addressed through the received LBA. EX1004, 8:54-9:25, FIG. 4a; EX1006, 13.

A POSITA would have a reasonable expectation of success implementing Reiter’s flash system to support ZBC. EX1003, ¶¶59-61, 70-71. SCSI commands and protocols were widely known and would have been familiar to a POSITA. EX1003, ¶70. SCSI specifications were intended to be used across a wide range of storage devices, including flash memory. EX1003, ¶¶59-62. A POSITA would have also understood that Reiter’s superblock-based memory organization is suitable for zone-based commands like ZBC. EX1003, ¶¶62-64. Further, a POSITA would have understood that ZBC’s zoned block device model is a logical construct that can be layered on top of any storage device. EX1003, ¶¶67-69.

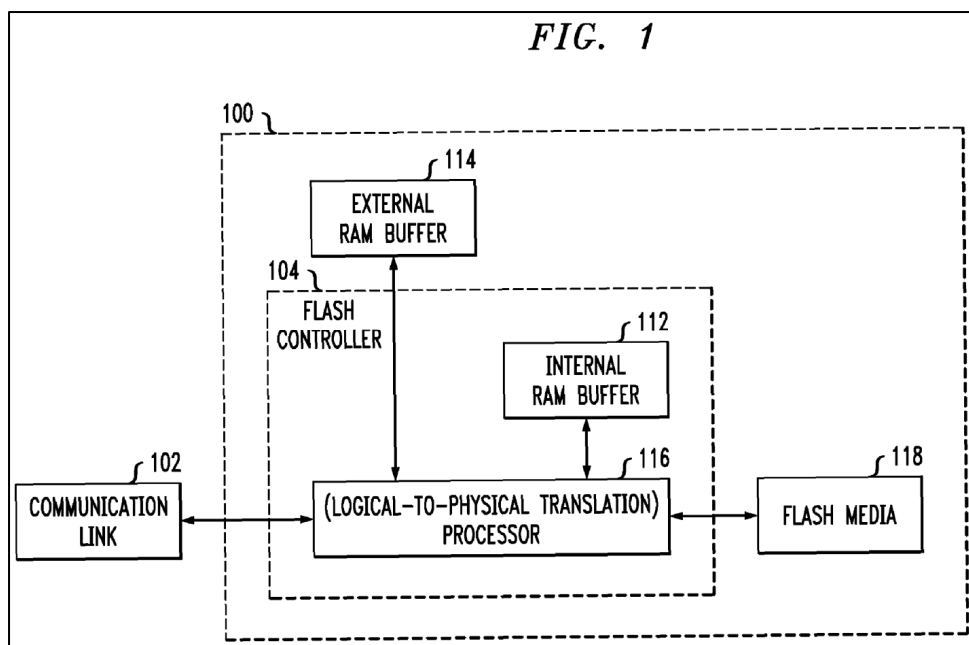
Reiter and ZBC are analogous art—relating to storage devices, including accessing blocks of memory. EX1003, ¶71.

D. Claim-by-Claim Analysis

1. Claim 1

(a) 1[pre]

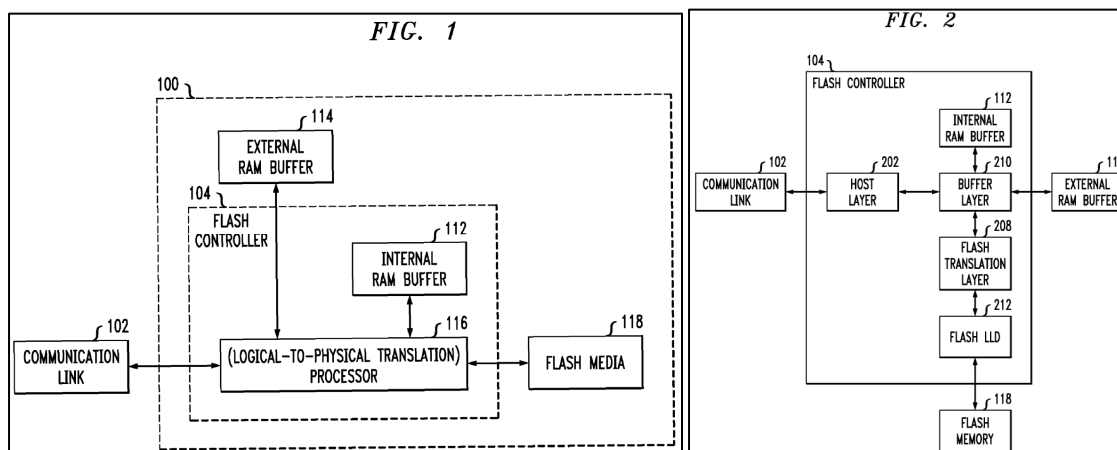
Reiter discloses a “flash ... system 100” (*storage device*) “implementing a logical-to-physical translation” scheme. EX1003, ¶¶72-73; EX1004, 6:63-66, FIG. 1:



(b) 1[a]

Reiter's flash system 100 includes "flash media 118"—"*flash memory* 118."

EX1004, 6:63-7:5, 7:56-61, FIGS. 1, 2:

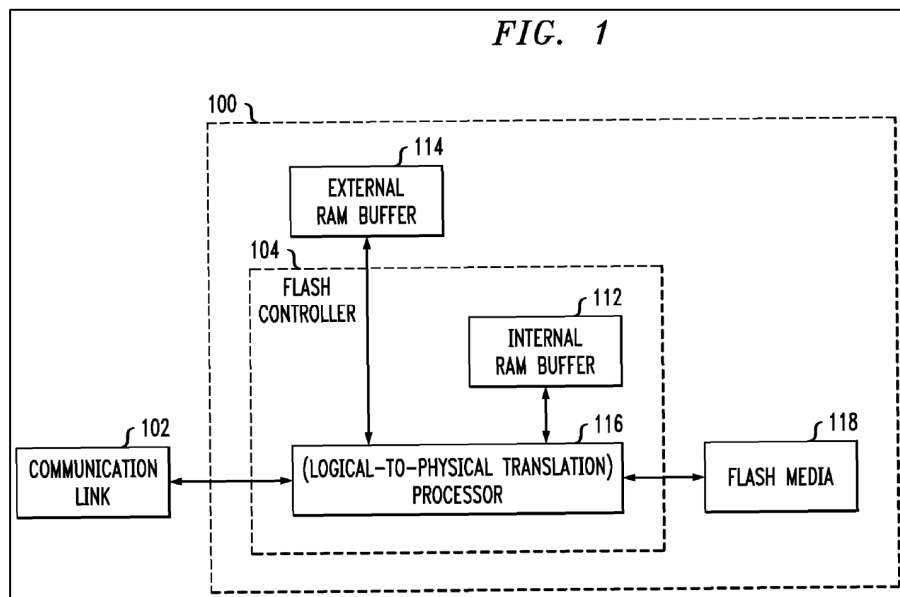


Reiter discloses that a "commonly employed type of flash memory technology is NAND flash memory." EX1004, 1:40-45. "There are three basic operations for NAND devices: read, write, and erase." EX1004, 1:51-52. Read and write

operations are performed on “[p]ages,” which “are typically arranged in blocks.” EX1004, 1:52-57. “[A]n erase operation is performed on a block by block basis.” EX1004, 1:56-57. Reiter also expressly discloses that its flash media comprises “blocks” that may be erased. EX1004, 6:60-62, 17:61, cls. 2, 9. Reiter thus discloses that its flash memory is made up of independently erasable blocks (*flash memory having erase units*). EX1003, ¶¶74-76.

(c) 1[b]

Reiter’s flash system 100 includes a “flash controller 104” (*circuitry*). EX1003, ¶77; EX1004, 6:63-7:2, FIG. 1:



Reiter’s “[F]lash controller 104 *receives* one or more requests for flash media access, such as read or write operations [*(incoming requests)*] from one or more external devices via communication link 102” (*from a host*). EX1004, 8:1-4. The “requests for access to flash media 118 generally include at least one ... LBA[]

where data should be read or written” (*incoming requests is accompanied by incoming address information*). EX1003, ¶¶78-79; EX1004, 8:4-6.

For example, Reiter discloses a “flash media write operation 630” that is “*executed* by FTL 208” in flash controller 104. EX1004, 13:38-46, FIG. 2. The “flash media write operation” is executed “in response to a flash media write request received from an external device,” which “includes a[n] ...LBA[.]” EX1004, 13:38-46.

A POSITA would be motivated to implement Reiter’s flash controller 104 to accept and execute ZBC commands. Section VIII.C. The ZBC command set includes a write command—a request disclosed in Reiter (EX1004, 8:1-6.)—and a reset write pointer command (*incoming requests including data write requests and erase requests*). EX1003, ¶80; EX1007, Table 112; *see also* EX1006, 22 (“The commands for host aware zoned block devices are defined in SBC-4.”). Both ZBC commands include an LBA (*wherein each of the incoming requests is accompanied by incoming address information*)—also as disclosed with Reiter’s requests (EX1004, 8:1-6, 11:55-63, 13:3-46.). EX1006, 29, Table 16; EX1007, Table 112. A ZBC write command includes an LBA. EX1007, Table 112:

Table 112 — WRITE (32) command									
Byte	Bit	7	6	5	4	3	2	1	0
0		OPERATION CODE (7Fh)							
1		CONTROL							
2		Reserved							
...									
5									
6		Reserved		GROUP NUMBER					
7		ADDITIONAL CDB LENGTH (18h)							
8	(MSB)	SERVICE ACTION (000Bh)							
9									(LSB)
10		WRPROTECT		DPO	FUA	Reserved	Obsolete	Reserved	
11		Reserved							
12	(MSB)	LOGICAL BLOCK ADDRESS							
...									
19									(LSB)
20	(MSB)	EXPECTED INITIAL LOGICAL BLOCK REFERENCE TAG							
...									
23									(LSB)
24	(MSB)	EXPECTED LOGICAL BLOCK APPLICATION TAG							
25									(LSB)
26	(MSB)	LOGICAL BLOCK APPLICATION TAG MASK							
27									(LSB)
28	(MSB)	TRANSFER LENGTH							
...									
31									(LSB)

A ZBC reset write pointer command “sets a write pointer to the lowest LBA of a write pointer zone.” EX1006, 3. A reset write pointer command includes a “ZONE ID,” which is the first LBA of the targeted zone. EX1006, 29, Table 16:

Table 16 — RESET WRITE POINTER command								
Bit Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (9Fh)							
1	Reserved			SERVICE ACTION (14h)				
3	(MSB)							
...	ZONE ID							
9	(LSB)							
10								
...	Reserved							
13								
14	Reserved							RESET ALL
15	CONTROL							

A POSITA would be motivated to implement the reset write pointer command to erase data in a zone. EX1003, ¶¶81-84 After applying a reset write pointer command, the zone’s data is no longer accessible. EX1006, 16-17. For instance, “[a] read operation on an LBA that has not been written since the last reset write pointer operation ... shall return user data set to the last initialization pattern ... set by ... the manufacturer[.]” EX1006, 16-17.

If the data is no longer accessible, it should be erased immediately or at least immediately queued for erasure. EX1003, ¶¶83-84. Immediate erasure is a simpler and with less risk of error. EX1003, ¶¶83-84. The effects of the reset write pointer command is fulfilled immediately, and cannot get tangled up with later commands. EX1003, ¶¶83-84. Alternatively, immediate queueing for erasure provides flexibility for erasing when normal read and write operations are not being processed. EX1003, ¶¶83-84. Reiter expressly complains about delays of normal

operations for erasure. EX1004, 4:19-26. “[g]arbage collection routines,” such as for “erasing out-of-date blocks,” “impact performance of the flash memory system by utilizing processor resources and potentially delaying write operations.” EX1004, 4:19-26.

(d) 1[c]

EX1003, ¶85; Sections VIII.D.1.e-h.

(e) 1[c][i]

From ZBC write and reset pointer commands (*accompanying incoming address information*), flash controller 104 *derives* (1) a number for a logical unit, volume, or partition (*first address portion*) and (2) a bitstring made up of superblock number 410, block index 412, and page index 414, or alternatively logical sector number 404 (*second address portion*). EX1003, ¶¶86-92.

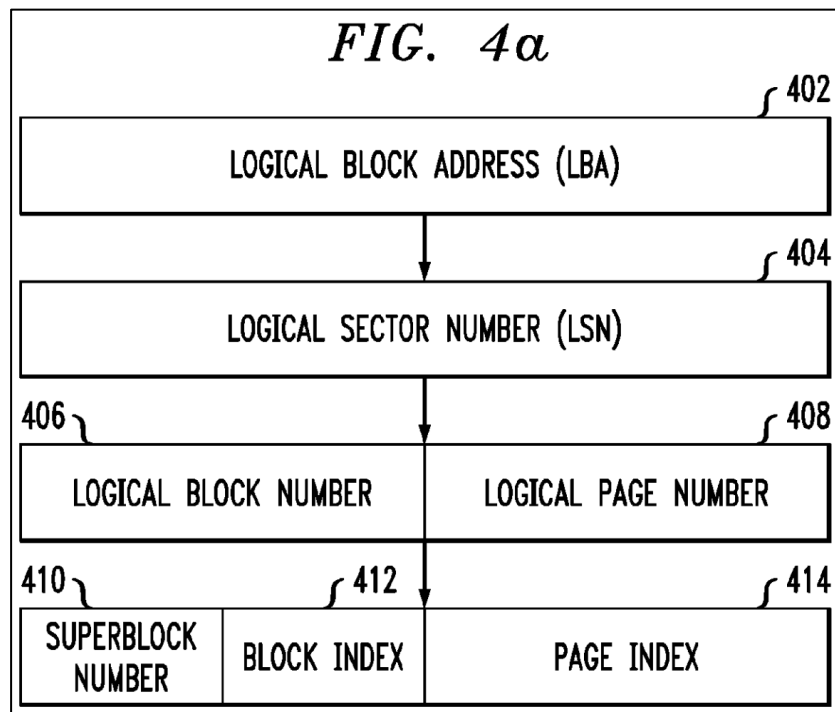
First Address Portion: A POSITA would be motivated to incorporate a logical unit number in ZBC commands, and motivated to modify flash controller 104 to decode (*derive*) that number (*first address portion*) from the command (*from the accompanying address information*). EX1003, ¶¶87-88. ZBC read commands target a “logical unit.” EX1008, 14, 61; *see also* EX1006, x, 1, 4. An “objective of ... [ZBC]” is to allow “communicat[ion] with a logical unit that declares itself to be a zoned block device.” EX1006, 1. But logical units are logical structures, where multiple can exist on the same physical storage device. *See* EX1008, 14, 462;

EX1003, ¶87. Each logical unit “shall begin with block zero and be contiguous up to the last logical block of that logical unit.” EX1008, 61. With multiple logical units, and overlapping LBA spaces, an LBA by itself is not enough to identify a location on a physical storage device. A POSITA would look to incorporate commands to distinguish the same LBA to two different logical units. EX1003, ¶¶87-88.

A POSITA would be motivated to incorporate the separate address found in SPC-4 into ZBC commands. EX1003, ¶88; Section VIII.C. SPC-4 recites that a “logical unit” is “[a]n **externally addressable** entity within a SCSI target device.” EX1008, 14. A logical unit is addressed by a “logical unit number”—“[a]n encoded 64 bit identifier for a logical unit.” EX1008, 14; *see also* EX1006, 4. ZBC already mentions logical unit number. EX1006, 4. A POSITA would be motivated to incorporate that address in its commands via the teachings in SPC-4. EX1003, ¶88.

Second Address Portion: A POSITA would be motivated to use Reiter’s flash controller 104 to *derive* the bitstring made up of superblock number 410, block index 412, and page index 414, or alternatively logical sector number 404, (*second address portion*) from the LBA in the ZBC write and reset write pointer commands (*from the accompanying address information*). EX1003, ¶¶89-92; *see* EX1004, FIG. 6b. ZBC write and reset write pointer commands include LBAs. EX1006, Table 16; EX1007, Table 112. Reiter discloses that read and write “request[s] ... include[] a

... LBA.” EX1004, 8:4-6, 8:62-67. “FIG. 4a shows an exemplary relation of a[n]
... LBA ... to a physical address ... as managed by FTL.” EX1004, 8:56-59, FIG.
4a:



Translating an LBA results in logical sector number 404, followed by a
bitstring of superblock number 410, block index 412, and page index 414. EX1003,
¶¶90-92; EX1004, 8:64-67, 9:5-20, FIG. 4a.

(f) 1[c][ii]

The logical unit number (*from the first address portion*) (Section VIII.D.1.d.)
identif[ies] the targeted logical unit (*addressed block device*). A logical unit has an
independent range of LBAs starting at zero. EX1003, ¶¶93-94; EX1008, 61; *see*
also EX1001, 7:54-63, FIG. 5.

(g) 1[c][iii]

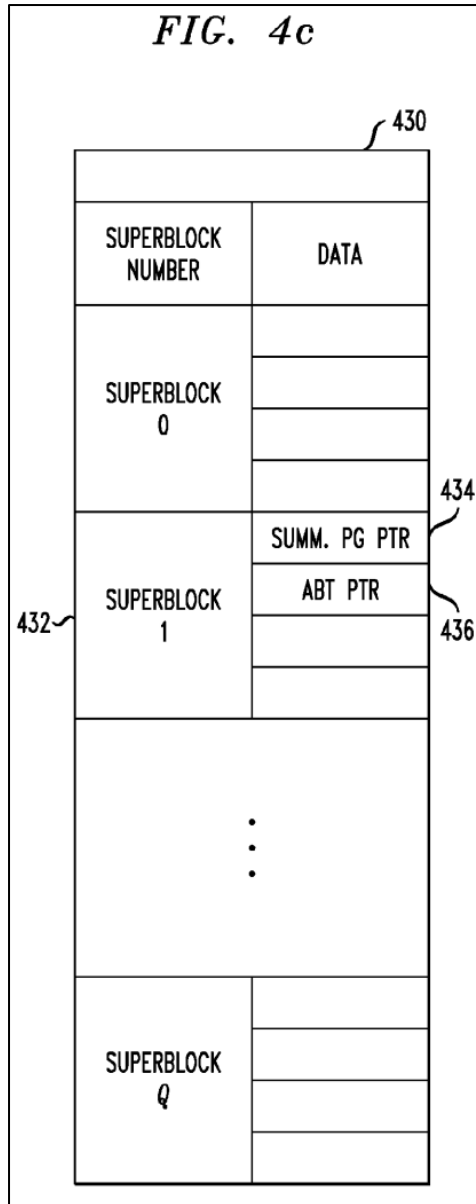
A POSITA would understand *using*, or be motivated to *use*, integer *division operation on the* superblock-block-page bitstring (*second address portion*) to extract superblock number 410. EX1003, ¶¶95-101. Alternatively, Reiter discloses *using* integer *division operation[s]* on logical sector number 404 (*second address portion*) to obtain superblock number 410. EX1004, 8:64-67, 9:5-20, FIG. 4a. Superblock number 410 *identifies* a superblock, which a POSITA would be motivated to correspond the superblock to a ZBC zone (*an addressed segment*). EX1003, ¶95; Section VIII.C; *see also* EX1011 at 22 (“Samsung’s Accused Instrumentalities ... identify an addressed zone (segment).”).

Integer Division On Superblock-Block-Page Bitstring: Superblock number 410 is a bitfield in the bitstring of superblock number 410, block index 412, and page index 414. EX1004, FIG. 4a; EX1003, ¶96. Integer division by a power-of-two extracts bitfields out of bitstrings. EX1003, ¶96. Specifically, the quotient of integer division by power-of-two will extract the most significant bitfields (*i.e.*, bits on the lefthand portion of a bitstring). EX1003, ¶96. Integer division is fast, because it can be implemented with a simple bit shift operation. EX1003, ¶96. Superblock number 410 bitfield is the higher order or most significant bitfield (*i.e.*, leftmost bits in the bitstring) of the superblock-block-page bitstring. EX1004, FIG. 4a; EX1003, ¶96. Integer division causes superblock number 410 to be extracted through the quotient.

EX1003, ¶96. A POSITA would be motivated to implement integer division via a bit shift operation to perform the extraction quickly. EX1003, ¶96.

Alternative, Integer Division On Logical Sector Number 404: Reiter discloses “deriv[ing] [logical block number] 406 by dividing LSN 404 by a number of sectors per block of flash media 118,” and “deriv[ing] [logical page number] 408 by dividing LSN 404 by a number of sectors per page of flash media.” EX1004, 9:5-9, FIG. 4a. From logical block number 406, superbloc number 410 is obtained. EX1004, 9:9-11, 9:13-17, FIG. 4a.

Identify An Addressed Segment: Superblock number 410 identifies a superbloc by selecting, from page global director (PGD) 430, the “summary page associated with [the corresponding] Superblock.” EX1004, 10:13-19, FIG. 4c:



“[A] Superblock generally is a logical collection of blocks” (*addressed segment corresponds to a subset of the erase units of the flash memory*). EX1003, ¶98-99; EX1004, 9:11-14, FIG. 4b.

Further, superblock number 410 is derived from LBA 402, (EX1004, FIG. 4a.) which is from a ZBC command’s LBA (Section VIII.D.1.e.). Because a ZBC LBA is within the LBA range specified by the targeted logical unit, (*see* Sections

VIII.D.1.e-f; EX1006, x, 1, 4; EX1008, 61.) superblock number 410 identifies a superblock *within* the targeted logical unit (*addressed block device*). EX1003, ¶¶100-01.

(h) 1[c][iv]

A POSITA would understand using, or be motivated to use, integer division on the superblock-block-page bitstring, or alternatively integer divisions on logical sector number 404, (*first division operation*) to obtain superblock number 410. Section VIII.D.1.g. Such division operations leave as a remainder a block-page bitstring of block index 412 and page index 414 (EX1003, ¶102-10; EX1004, FIG. 4a.) or alternatively, for the latter, results a block-page bitstring of logical block number 406 and logical page number 408, (EX1004, 9:5-9, FIG. 4a.) (*identify from the first division operation a third address portion*).

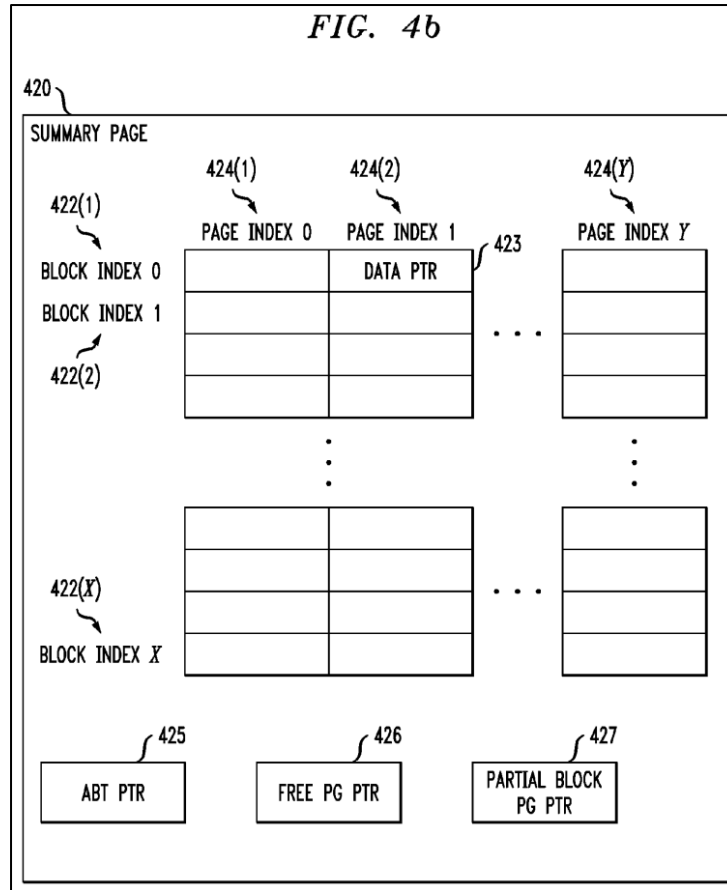
Reiter discloses further translation for write requests (*i.e.*, ZBC write commands (Section VIII.D.1.c.)) (*for each of the data write requests*) on the block-page bitstrings. EX1003, ¶103; EX1004, 8:56-58, 9:60-63 (indexing summary page 420 with block index 412 and page index 414 to retrieve data pointer 423), 13:38-46, FIGs. 4a-b, 6b.

Second Division Operation On Bitstring Of Block Index 412 And Page

Index 414: A POSITA would understand using, or be motivated to use, the same division operation (*use a second division operation on the third address portion*),

that subdivided off superblock number 410, (Section VIII.D.1.g.) to separate the bitstring of block index 412 and page index 414 into respective bitfields—block index 412 (*to identify an address value*) and page index 414 (*to identify a storage location*). EX1003, ¶¶108-10. Block index 412 and page index 414 would be the quotient and remainder, respectively, from integer division.

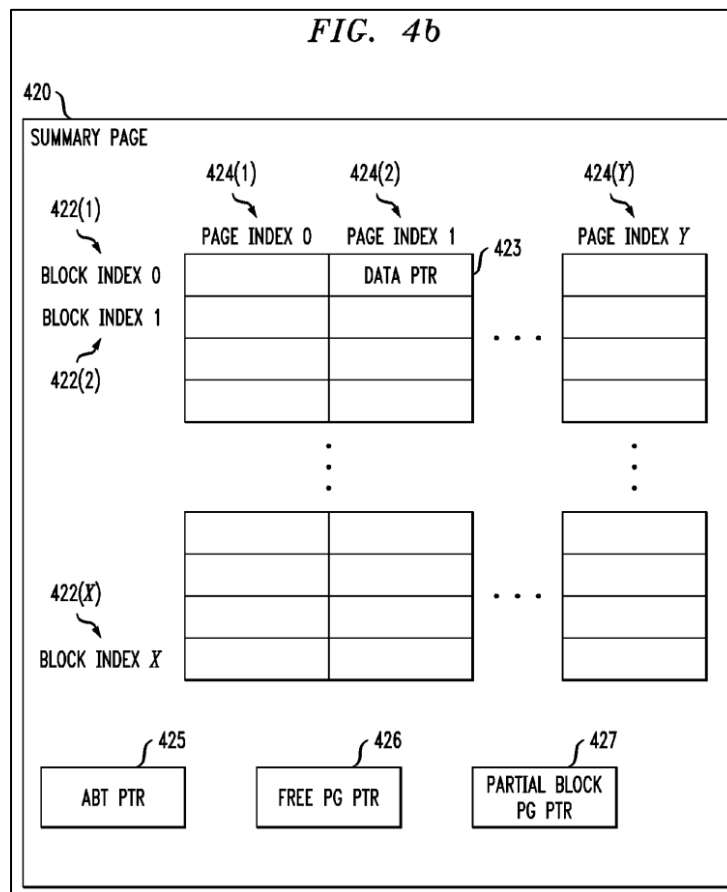
As discussed in the prior sub-section, block index 412 (*address value*) *selects* one of the physical blocks in the superblock (*erase units from the subset which corresponds to the addressed segment*), and page index 414 (*storage location*) selects one of the pages within one of the selected physical blocks (*within the selected one of the erase units*). EX1003, ¶109; EX1004, FIG. 4b:



Alternative, Second Division Operation On Logical Block Number 406

And Logical Page Number 408: Reiter discloses performing a second set of division operations on logical block number 406 and logical page number 408 (*use a second division operation on the third address portion*) to identify block index 412 (*address value*) and page index 414 (*storage location*). EX1003, ¶104-07. Specifically, Reiter discloses “deriv[ing] ... block index 412 from LBN 406 by dividing LBN 406 by a number of blocks per Superblock, where ... block index 412 corresponds to the remainder,” and “[p]age index 414 is derived from LPN 408 by dividing LPN 408 by a number of pages per block.” EX1004, 9:13-20.

Block index 412 (*address value*) *selects* one of the physical blocks in the superblock (*erase units from the subset which corresponds to the addressed segment*). EX1003, ¶¶104-07. “[A] Superblock generally is a logical collection of blocks” EX1004, 9:11-14. And the block index 412 selects a block within the superblock’s collection. See EX1004, 9:13-17, 9:52-58, FIGS. 4b-c. “Each Superblock has a summary page ... 420,” which “contains mapping data for the ... Superblock.” EX1004, 9:52-54, FIG. 4b. “For example, summary page 420 contains the block index and page index ... for all X blocks and Y pages in [the] Superblock.” EX1004, 9:54-58, FIG. 4b:



Block index 414 indexes the “summary page block indices [that] include all physical blocks ... within the Superblock.” EX1004, 9:58-60, FIG. 4b.

Similarly, page index 414 (*storage location*) selects one of the pages within one of the selected physical blocks (*within the selected one of the erase units*). EX1003, ¶107. As previously stated, “summary page 420 contains the block index and page index ... for all X blocks and Y pages in [the] Superblock.” EX1004, 9:54-58, FIG. 4b (above).

(i) 1[d]

A POSITA would be motivated to implement Reiter’s flash controller 104 (*said circuitry*) to *execute* ZBC commands, including write commands (*each one of the data write requests*). Section VIII.D.1.c. Reiter discloses flash controller 104’s “determin[ing] the Superblock number, Block index, and Page index of the first page to be written,” (*storage location that was identified for that one of the data write requests*) and the “requested data is written at the active page address at step 644” (*programming accompanying data*). EX1003, ¶¶111-12; EX1004, 13:38-60, FIG. 6b.

(j) 1[e]

A POSITA would be motivated to implement Reiter’s flash controller 104 (*said circuitry*) to accept and *execute* ZBC commands, including reset write pointer commands (*each one of the incoming erase requests*). Section VIII.D.1.c. A

POSITA would be motivated to implement the reset write pointer command to erase data in the zone targeted by the command (*addressed segment that was identified from the incoming address information accompanying the one of the incoming erase requests*). Sections VIII.D.1.c, VIII.D.1.e-g. A POSITA would be motivated to use the superblocks disclosed in Reiter as zones for ZBC commands. Section VIII.C. “[A] Superblock generally is a logical collection of blocks.” EX1004, 9:11-14. A POSITA would understand that erasing a superblock erases the blocks of that superblock (*controlling erasure in the flash memory of one or more of the erase units in the subset corresponding to the addressed segment*). EX1003, ¶¶113-14.

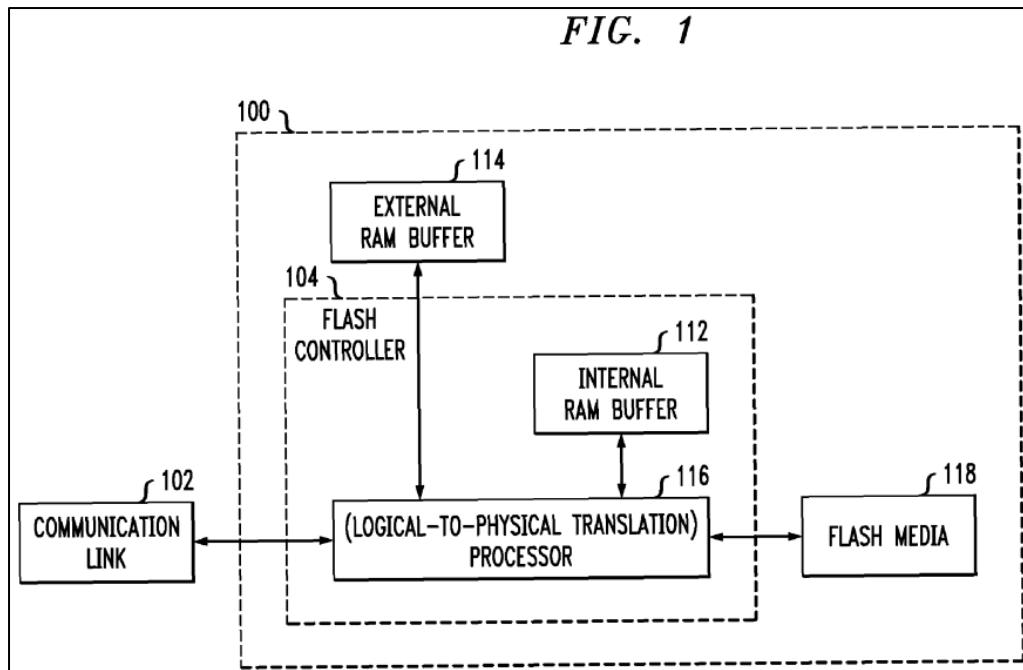
2. Claim 2

(a) 2[pre]

EX1003, ¶115; Section VIII.D.1.

(b) 2[a]

Reiter’s flash controller 104 (*said circuitry*) *comprises* “processor 116” (*at least one processor*). EX1003, ¶¶116-17; EX1004, 7:21-22, 7:31-39, FIG. 1:



(c) 2[b]

Reiter’s flash system (*storage device*) includes flash controller 104. EX1004, FIG. 1. And flash controller 104 includes processor 116. EX1004, 7:19-39, FIG.1. “Processor 116 includes software and/or firmware as needed for operation” (*said instructions when executed control the at least one processor*), which a POSITA would understand as including *instructions stored on at least one physical storage medium*. EX1004, 7:31-34; EX1003, ¶¶118-19.

3. Claim 3

(a) 3[pre]

EX1003, ¶120; Section VIII.D.1.

(b) 3[a]

A POSITA would be motivated to implement Reiter’s flash controller 104 (*said circuitry*) to *receive* and execute ZBC commands. Section VIII.D.1.c. ZBC includes an extended copy command (*request from a host to copy*). EX1006, 1, 22; EX1007, Table 33; EX1008, 287-89, Table 128. The extended copy command “cop[ies] data from one set of copy sources (e.g.,... logical units) to a set of copy destinations (e.g.,... logical units)” (*copy host-selected data from one storage location in flash memory to another storage location in flash memory*). EX1003, ¶121; EX1008, 287-89 Specifically, the extended copy command includes a parameter list, which has segment descriptors. EX1008, Table 128:

Table 128 — EXTENDED COPY(LID4) parameter list (part 1 of 2)								
Bit Byte	7	6	5	4	3	2	1	0
0	PARAMETER LIST FORMAT (01h)							
1	Reserved		STR	LIST ID USAGE		PRIORITY		
2	(MSB) _____							
3	HEADER CSCD DESCRIPTOR LIST LENGTH (0020h)							(LSB)
4	_____							
...	Reserved							_____
14	_____							
15	Reserved					G_SENSE		IMMED
16	HEADER CSCD DESCRIPTOR TYPE CODE (FFh)							
17	_____							
18	Reserved							_____
19	_____							
20	(MSB) _____							
...	LIST IDENTIFIER							_____
23	_____							(LSB)
24	_____							
...	Reserved							_____
41	_____							
42	(MSB) _____							
43	CSCD DESCRIPTOR LIST LENGTH (n-47)							(LSB)
44	(MSB) _____							
45	SEGMENT DESCRIPTOR LIST LENGTH (m-n)							(LSB)
46	(MSB) _____							
47	INLINE DATA LENGTH (k-m)							(LSB)
CSCD descriptor list								
48	_____							
...	CSCD descriptor [ID 1] (see 6.4.5)							_____
79	_____							
⋮								
n-31	_____							
...	CSCD descriptor [ID x] (see 6.4.5)							_____
n	_____							

Table 128 — EXTENDED COPY(LID4) parameter list (part 2 of 2)								
Bit Byte	7	6	5	4	3	2	1	0
	Segment descriptor list							
n+1	Segment descriptor (see 6.4.6) [first]							
...								
	⋮							
...	Segment descriptor (see 6.4.6) [last]							
m								
m+1	Inline data							
...								
k								

The segment descriptors detail the source LBA (*one storage location*) and destination LBA (*another storage location*). EX1003, ¶122; EX1008, Table 153:

Table 153 — Block device to block device segment descriptor													
Bit Byte	7	6	5	4	3	2	1	0					
0	DESCRIPTOR TYPE CODE (02h or 0Dh)												
1	Reserved						DC	CAT					
2	(MSB)	DESCRIPTOR LENGTH (0018h)						(LSB)					
3													
4	(MSB)	SOURCE CSCD DESCRIPTOR ID						(LSB)					
5													
6	(MSB)	DESTINATION CSCD DESCRIPTOR ID						(LSB)					
7													
8	Reserved												
9	Reserved												
10	(MSB)	BLOCK DEVICE NUMBER OF BLOCKS						(LSB)					
11													
12	(MSB)	SOURCE BLOCK DEVICE LOGICAL BLOCK ADDRESS						(LSB)					
...													
19		DESTINATION BLOCK DEVICE LOGICAL BLOCK ADDRESS						(LSB)					
20	(MSB)												
...								(LSB)					
27													

A POSITA would understand that executing the extended copy command (*in response to the request from the host to copy*) includes flash controller 104 (*circuitry*) to *issue commands reading* data from the source LBA (*host-selected data from a first physical storage location in the flash memory*) and *writing* the data to the destination LBA (*host-selected data to a second physical storage location in the flash memory*). EX1003, ¶123; EX1008, 245, 270.

SPC-4 discloses executing the extended copy command only via a “copy manager” which “may be contained in... a standalone SCSI target device”—Reiter’s flash system 100—(*without sending the data being copied to the host*). EX1003, ¶¶124-25; EX1008, 245-47, 287-89.

(c) 3[b]-[c]

SPC-4 discloses the extended copy command from the host (*the request from the host to copy*) includes a destination LBA (*designate destination address information*). Section VIII.D.3.b. A POSITA would be motivated to use flash controller 104 (*circuitry*) and perform the same translation as provided to ZBC write commands on the destination LBA to a corresponding physical storage location (*designate destination address information corresponding to the second physical storage location*), which would include *using the first division operation on the designated destination address information to identify a destination segment within a destination block device*. EX1003, ¶¶126-27; See Sections VIII.D.1.d-h.

4. Claim 4

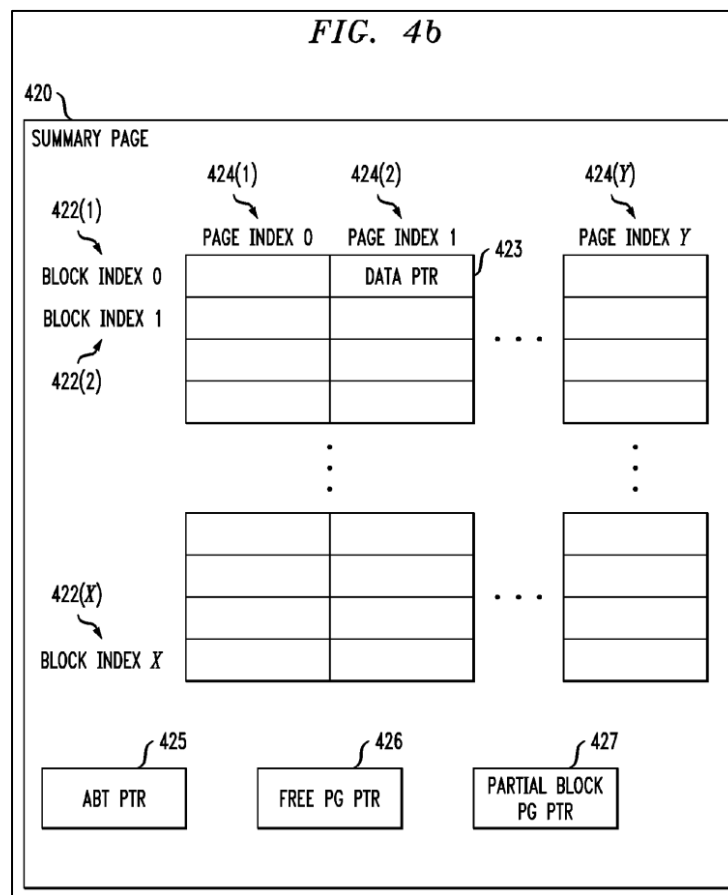
(a) 4[pre]

EX1003, ¶128; Section VIII.D.3.

(b) 4[a]

A POSITA would be motivated to implement Reiter's flash controller 104 (*circuitry*) to accept and execute ZBC commands, including extended copy commands. Section VIII.D.3.b. A POSITA would understand, or find it obvious, that the destination LBA decoded from a ZBC extended copy command (*identify from the destination address information*) is translated by Reiter's flash controller 104 into block index 412 (*logical erase unit*). EX1003, ¶129; Section VIII.D.1.e-h.

Block index 412 is a *logical erase unit*. EX1003, ¶130-32. Block index is a logical unit (*i.e.*, “*logical erase unit*”), indexing a superblock’s “logical collection of blocks”—not the physical block number itself. EX1004, 9:11-13, FIGs. 4b-c. Block index 412 refers to an erase unit (*i.e.*, “*logical erase unit*”), as block index 412 indexes the blocks in a superblock, and “an erase operation is performed on a block by block basis.” EX1004, 1:56-57. “For example, summary page 420 contains the block index and page index ... for all X blocks and Y pages in [the] Superblock.” EX1004, 9:54-58, FIG. 4b:



And block index 414 indexes the “summary page block indices [that] include all physical blocks ... within the Superblock.” EX1004, 9:58-60, FIG. 4b. And as disclosed in Reiter, “[p]ages are typically arranged in blocks, and an erase operation is performed on a block by block basis.” EX1004, 1:56-57.

5. Claim 6

(a) 6[pre]

EX1003, ¶133; Section VIII.D.1.

(b) 6[a]

A POSITA would be motivated to implement Reiter’s flash memory controller 104 (*storage device*) to accept and execute ZBC commands. Section VIII.D.1.c. The ZBC command set includes a report zones command. EX1006, 24. The report zones command optionally “return[s]” (*transmit*) a “[l]ist [of] zones in the zoned block device” back *to [the] host* via “parameter data” that includes the length of the zone list and zone descriptors indicating the starting LBA and length of each zone (*the information identifying the number of structures to a host*). EX1003, ¶¶134-35; EX1006, 25, Table 11, Tables 12, 13:

Table 12 — REPORT ZONES parameter data								
Bit Byte	7	6	5	4	3	2	1	0
1	(MSB)							
...	ZONE LIST LENGTH (n-63)							
3	(LSB)							
4	Reserved							SAME
5	Reserved							
...								
63								
Zone descriptors list								
64	Zone descriptor [first]							
...								
127								
	⋮							
n-63	Zone descriptor [last]							
...								
n								

Table 13 — Zone descriptor format								
Bit Byte	7	6	5	4	3	2	1	0
0	Reserved				ZONE TYPE			
1	ZONE CONDITION				Reserved			RESET
2	Reserved							
...								
7								
8	(MSB)							
...	ZONE LENGTH							
15	(LSB)							
16	(MSB)							
...	ZONE START LBA							
23	(LSB)							
24	(MSB)							
...	WRITE POINTER LBA							
31	(LSB)							
32	Reserved							
...								
63								

Because flash system 100 transmits the zone list information back to the host, a POSITA would understand that flash system 100 also stores that information (*store information identifying a number of structures in the flash memory*). EX1003, ¶135-38. Or a POSITA would be motivated to store such information. EX1003, ¶135. As opposed to wasting time re-calculating such information with every command requiring zone metadata, flash system could store zone metadata and update it as zone settings are changed. EX1003, ¶135.

Later read and reset write pointer commands by the host (*at least one of the incoming requests*) depend the information reported by the zone report command to ensure valid read and reset write pointer commands (*at least one request is dependent on the information transmitted to the host identifying the number of structures*). EX1003, ¶136. Valid read commands require reading an “LBA that has been written since the last reset write pointer operation.” EX1006, 16-17. Valid reset write pointer commands require the “ZONE ID field [to] specif[y] the zone start LBA ...” EX1006, 29. In order to keep read and reset write pointer commands valid, the list of zones with the zone metadata is needed. EX1006, 25, Tables 11-13.

A POSITA would be motivated to implement Reiter’s flash system’s 100 flash controller 104 (*storage device comprises circuitry*) to accept and execute ZBC read and reset write pointer commands (*fulfill the at least one request by issuing*

commands to the flash memory for which a quantum of flash memory selected by the commands). EX1003, ¶¶137-38; Sections VIII.D.1.c-j, VIII.D.3.c.

6. Claim 7

(a) 7[pre]

EX1003, ¶139; Section VIII.D.1.

(b) 7[a]

A POSITA would be motivated to implement Reiter’s flash memory controller 104 (*storage device*) to accept and execute ZBC commands. Section VIII.D.1.c. The ZBC command set includes a report zones command. EX1006, 24. The list of zones reported by the report zone command (*the information transmitted to the host identifying the number of structures*) also provides the length of each zone (*identifies an amount of flash memory that is to be erased as a unit*). EX1003, ¶¶140-42. The report zones command optionally “return[s] ... parameter data” that includes a zone descriptor for each zone indicating the length of each zone. EX1006 25, Tables 11-13:

Table 13 — Zone descriptor format								
Bit Byte	7	6	5	4	3	2	1	0
0	Reserved				ZONE TYPE			
1	ZONE CONDITION				Reserved			RESET
2	Reserved							
...								
7								
8	(MSB)	ZONE LENGTH						(LSB)
...								
15								
16	(MSB)	ZONE START LBA						(LSB)
...								
23								
24	(MSB)	WRITE POINTER LBA						(LSB)
...								
31								
32	Reserved							
...								
63								

A POSITA would be motivated to implement the reset write pointer command to cause erasure of the targeted zone. Section VIII.D.1.c Because the reset write pointer command erases an entire zone, the zone length *identifies an amount of flash memory that is to be erased as a unit*. EX1003, ¶¶141-42.

7. Claim 8

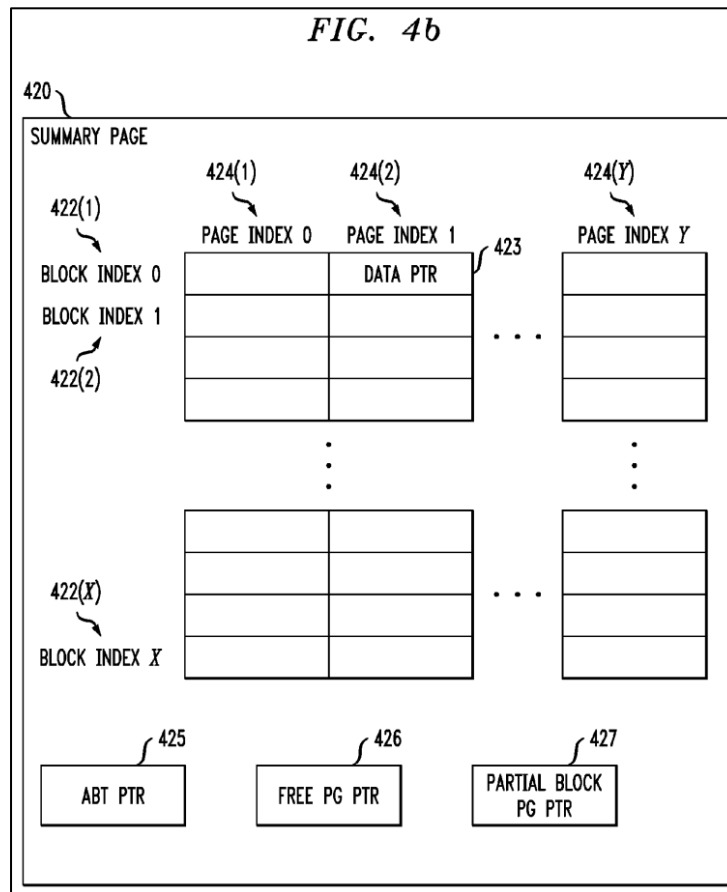
(a) 8[pre]

EX1003, ¶143; Section VIII.D.1.

(b) 8[a]

The bitstring of block index 412 and page index 414, and alternatively logical block number 406 and logical page number 408, constitutes the *third address portion*. Section VIII.D.1.g. Both mappings are from a set of sequential indexes for

selecting within a superblock's list of blocks and pages within those blocks (*specifies a location within a sequential range of locations that correspond to the erase units in the subset*). EX1003, ¶¶144-45; Section VIII.D.1.g. Specifically, “summary page 420 contains the block index and page index ... for all X blocks and Y pages in [the] Superblock.” EX1004, 9:54-58, FIG. 4b:



8. Claim 9

(a) 9[pre]

EX1003, ¶146; Section VIII.D.7.

(b) 9[a]

EX1003, ¶147; See Sections VIII.D.1.g-h.

9. Claim 13

(a) 13[pre]

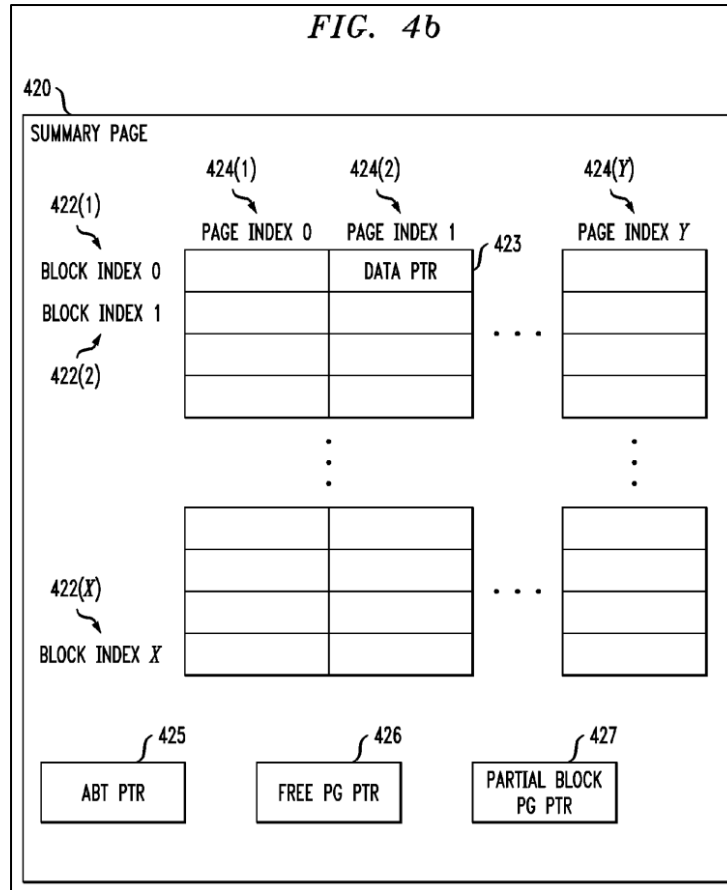
EX1003, ¶148; Section VIII.D.1.

(b) 13[a]

EX1003, ¶149; Section VIII.D.1.b.

(c) 13[b]

A POSITA would understand, or find it obvious, to use integer division (*second division operation*) to extract block index 412 (*logical erase unit*). EX1003, ¶¶150-52; Section VIII.D.1.g. Block index 412 is a *logical erase unit* that *maps to one of the physical erase units*. Block index is a logical unit (*i.e.*, “logical erase unit”), referring to an index number in a logical list—a superblock, which is a “logical collection of blocks”—to obtain “data pointer 423, which points to the physical address” (*each logical erase unit is mapped to one of the physical erase units*). EX1004, 9:11-13, 9:54-58, 9:60-63, FIG. 4b:



And a block is an *erase unit* because “erase operation[s] [are] performed on a block by block basis.” EX1004, 1:56-57.

(d) 13[c]

Reiter’s flash storage system 100 (*storage device*) *detects* bad blocks (*failure condition of the physical erase units*). EX1003, ¶¶153-54. Upon detecting a bad block, a POSITA would be motivated to *remap* logical block number 406, and block index 412, (*logical erase unit*) from the bad block (*one of the physical erase units for which the failure condition is detected*) to *instead* a working block (*one of the*

physical erase units for which the failure condition has not been detected).

EX1003, ¶¶153-58.

Detect A Failure Condition Of The Physical Erase Unit: Reiter discloses “maintain[ing] a record of bad blocks” by scanning for bad blocks after powering on flash system 100. EX1004, 4:27-37, 17:13-15, 17:47-63. For example, “On startup of flash ... system 100, mapping data ... requires reconstruction.” EX1004, 17:13-15. This includes “scan[ning] and group[ing] ... blocks” into various “categories,” including “bad blocks.” EX1004, 17:47-52. Bad blocks are detected if (1) FTL 208 is “unsuccessful” “read[in]g metadata that is stored in the ... first page of the block” and (2) the following “erase operation of the [same] block was unsuccessful, the block cannot be erased and has failed.” EX1004, 17:59-67. “[T]he block address is [then] added to the bad block list.” EX1004, 18:4-8.

Remap Each Logical Erase Unit Which Corresponds To One Of The Physical Erase Units For Which The Failure Condition Is Detected: When a block is added to the bad block list, a POSITA would be motivated to replace the bad block with a working block—remapping logical block number 406, and block index 412, to a new block. EX1003, ¶¶155-58. Data from a bad block cannot be trusted, especially given the unsuccessful read during the startup scan and subsequent failed erasure of the block. 4:27-37, 17:13-15, 17:47-63. Keeping a bad block around risks reading erroneous data. EX1003, ¶155. As a first remapping step, Reiter discloses,

upon detecting a bad block, adding the “block address ... to the bad block list.” EX1004, 18:4-8.

A POSITA would be motivated to reconstruct the lost data through Reiter’s RAID functionalities, and change data pointer 423 from the old bad block to a new working block with the reconstructed data. EX1003, ¶156. Reiter discloses configuring flash system 100 “analogously to hard drives in a redundant array of independent disks (RAID).” EX1004, 8:46-50; EX1003, ¶156. RAID allows data lost on one disk to be reconstructed with data from the other redundant disks. EX1003, ¶156.

Because the previous block is an unreliable bad block, a POSITA would understand that a new block is needed for the reconstructed data. EX1003, ¶157. And a POSITA would be motivated to change data pointers 423 in summary page 420 pointing to the old bad block to the new block with the reconstructed data. EX1003, ¶157. This allows the LBA addressing, including logical block number 406 and block index 412, to be unchanged despite the data loss. EX1003, ¶157. With the new working block, data at logical block number 406 and block index 412 is trustworthy and usable for later reads (*utilize the remapped correspondence to service one or more of the incoming data access requests*). EX1003, ¶157; EX1004, 17:13-21.

10. Claim 15

(a) 15[pre]

EX1003, ¶159; Section VIII.D.1.

(b) 15[a]

A POSITA would understand, or find it obvious, that Reiter's flash system 100's flash media 118 (*flash memory*) can *have* multiple SCSI logical units (*block devices*). EX1003, ¶¶160-61; Section VIII.D.1.e. ZBC write and erase commands target one of those logical units, volumes, or partitions (*including the addressed block device*). Section VIII.D.1.f.

(c) 15[b]

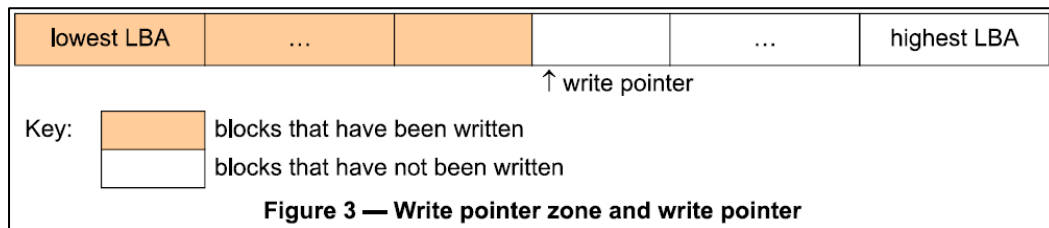
Sections VIII.D.10.d-e.

(d) 15[b][i]

A POSITA would understand to implement Reiter's flash controller 104 to *store* a RESET bit (*metadata*) for each zone of a block device (*on an independent basis for flash memory associated with each one of the block devices*). EX1003, ¶¶163-64; A RESET bit is stored for each zone because a ZBC report zones command is optionally set to return the "[l]ist [of] zones for which the RESET bit is set to one." EX1006, 25. A POSITA would understand that flash controller 104 returning the list of zones with RESET bits set to one means storing a RESET bit for each zone.

(e) 15[b][ii]-[iii]

Reiter's flash controller 104 *receives* a ZBC report zones command (*query*), that is sent by the host (*issued by a host device*). ZBC discloses that a ZBC report zones command is optionally set to return the "[l]ist [of] zones for which the RESET bit is set to one." EX1006, 25. Following this report zones command (*responsive to the query*), each zone's RESET bit (*identify specific metadata*) is checked if equal to one (*compare metadata to at least one threshold*), and a list of zones with the RESET bit equal to one sent to the host (*transmit information to the host device*). EX1003, ¶¶165-66. EX1006 25, Tables 12-13. The list of zones is sent back via parameter data, which includes a list of zone descriptors. EX1006 25, Tables 12-13 The zone descriptors include the zone's write pointer LBA, which details how much of the zone has user data (*such transmitted information conveys address information for data stored at one of the one or more physical storage locations associated with the specific metadata*). See EX1006, Figure 3:



11. Claim 19-21

Claims 19-21 are substantively identical to claim 1 incorporated with claims 9 and 22. EX1003, ¶¶167-168.

Claim 19	Claim 20	Claim 21	Corresponding Claim 1 Limitation(s)
19[pre]	20[pre]	21[pre]	1[pre]
19[a]	20[a]	21[a]	1[a] (1[c][iii]-[iv] for claims 20-21)
19[b]	20[b]	21[b]	1[b]
19[c]	20[c]	21[c]	1[c]
19[c][i]	20[c][i]	21[c][i]	1[c][i]
19[c][ii]	20[c][ii]	21[c][ii]	1[c][ii]
19[c][iii]	20[c][iii]	21[c][iii]	1[c][iii], 9[a], 22[a]
19[c][iv]	20[c][iv]	21[c][iv]	1[c][iv], 9[a], 22[a]
19[d]	20[d]	21[d]	1[d]
19[e]	20[e]	21[e]	1[e]

Limitations 20[a] and 21[a] also add that “block devices,” “segments,” and “storage locations” are at “hierarchically-superior,” “-intermediate,” and “-inferior level[s],” respectively, and that block devices and segments contain mutually exclusive sets of segments and logical erase units, respectively. But Ground I discloses logical units, each with its own LBA range, thereby its own zones/superblocks, where each zone/superblock contains its own logical blocks and pages. Sections VIII.D.1.e-h. Therefore, claims 20 and 21 are rendered obvious for the same reasons as claim 1.

12. Claim 22

(a) 22[pre]:

EX1003, ¶169; Section VIII.D.1.

(b) 22[a]

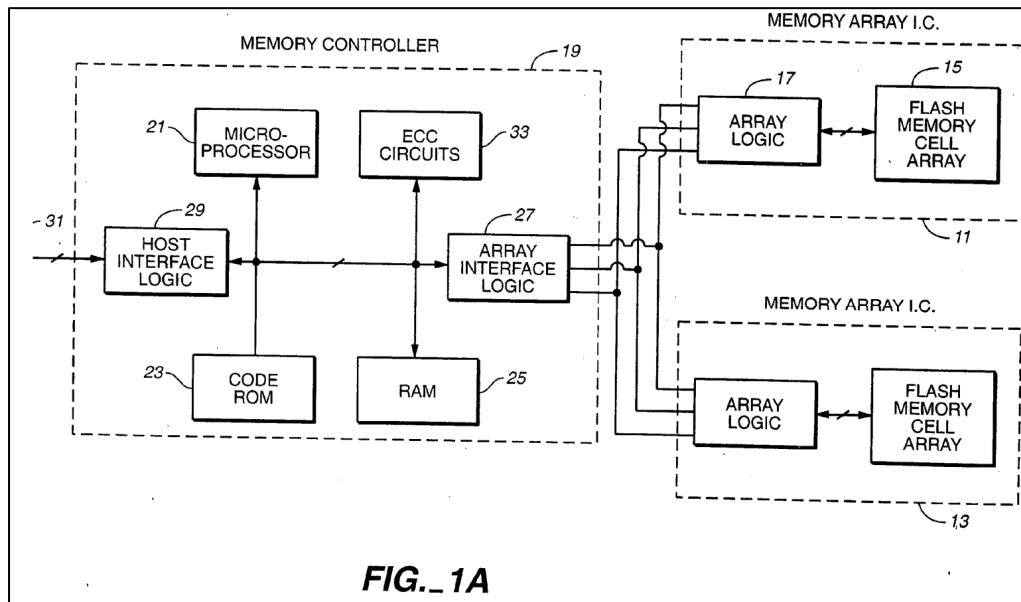
EX1003, ¶170; *See* Section VIII.D.1.g.

IX. GROUND II: CLAIMS 5, 10-13, AND 18 ARE RENDERED OBVIOUS BY REITER IN LIGHT OF ZBC, SBC-4, SPC-4, AND SINCLAIR-367

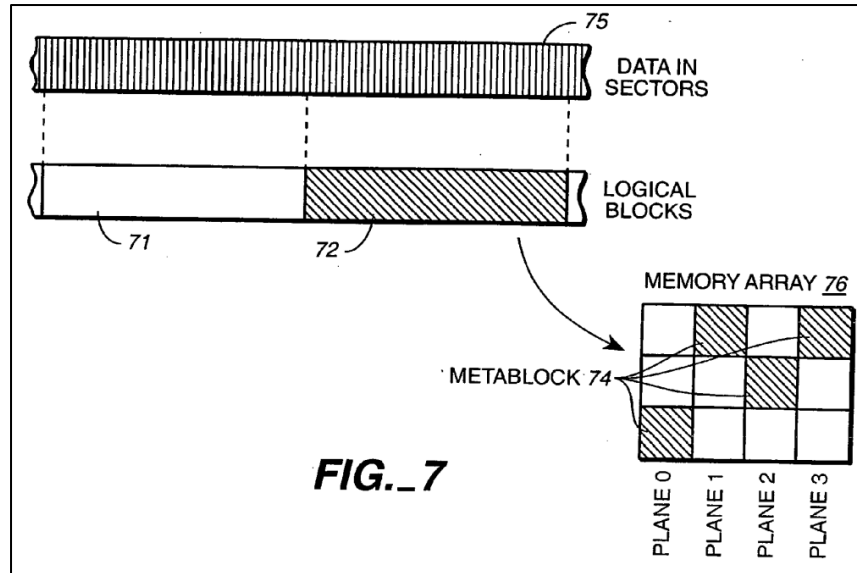
A. Sinclair-367

U.S. Patent Publication No. 2005/0144367 (“Sinclair-367”) was filed on December 16, 2004, published on June 30, 2005 and is prior art under AIA 35 U.S.C. § 102.

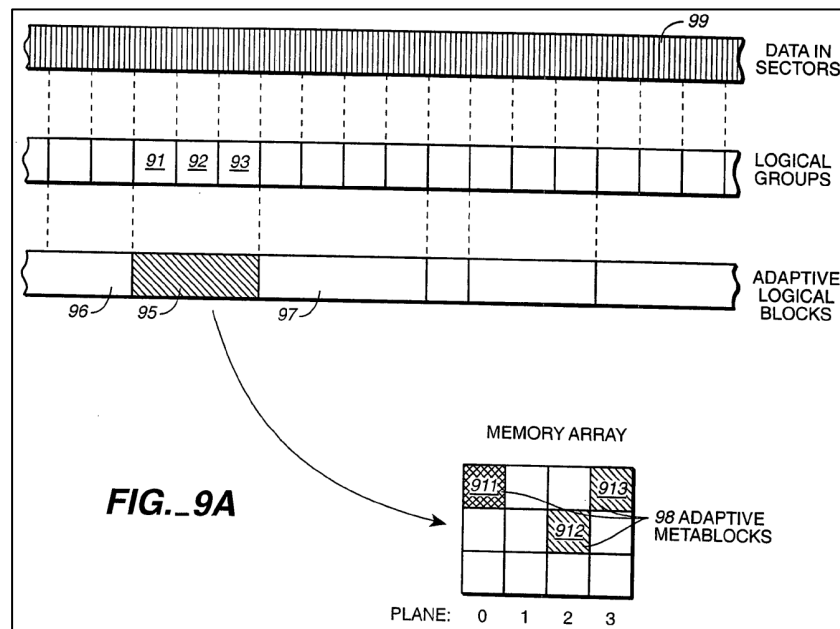
Sinclair-367 discloses a non-volatile memory array (*i.e.*, flash memory) via adaptive metablocks. EX1003, ¶¶51-54; EX1009, Abstract, FIG. 1A:



Metablocks are a logical grouping of erase units with each erase unit in a different plane so that reading, writing, and erasing across the metablock occurs simultaneously. EX1009, ¶[0117], FIG. 7:



Adaptive metablocks are also a logical grouping of erase units, but do not have a fixed size. EX1009, ¶[0117]. Adaptive metablocks might not have one erase unit in each plane (*e.g.*, two erase units in one plane), limiting the extent of simultaneous operation. EX1009, ¶¶[0117], [0119]-[0120], FIG. 9A:



B. Motivation to Combine

A POSITA would have been motivated to combine Reiter's flash system executing ZBC commands (Section VIII.C.) with Sinclair-367's bad block management, metadata storage, and planes and metablocks. EX1003, ¶171.

Sinclair-367 manages bad blocks—identifying “defective cells within the erase block,” and remapping the “logical address” with “the address of any substitute” block, where the “primary erase block is defective.” EX1003, ¶172; EX1009, ¶¶[0107]-[0109]. This improves system resiliency. EX1003, ¶172. Because Reiter contemplates bad block management, (*see* Section VIII.D.1.j.) a POSITA would have a reasonable expectation of success implementing bad block management in Reiter's flash device. EX1003, ¶172.

Further, Sinclair-367 discloses storing in each erase unit a sector of metadata (*e.g.*, error correcting codes (“ECC”) and number of program/erase cycles). EX1009, ¶¶[0106]-[0107], FIG. 3. ECCs recognize and repair damaged data. EX1009, ¶¶[0007], [0102]. Program/erase cycles determine the wear and tear on a physical block. EX1009, ¶¶[0107]-[0108]. Cycle count “is used in a wear-leveling algorithm ... even[ing] out the usage (wear) of all the erase blocks,” increasing the resiliency and lifespan of the flash system. EX1003, ¶173; EX1009, ¶¶[0107]-[0108]. Because a single sector is a small overhead for a block, yet reduces data

loss, a POSITA would have a reasonable expectation of success storing a metadata sector in physical blocks.

Further, Sinclair-367 discloses dividing flash memory into planes, and forming metablocks by linking erase units in each plane. EX1009, ¶¶[0009]-[0012], [0111]-[0112]. This improve speed and throughput. Each plane operates independently—concurrently executing reading/writing. EX1009, ¶[0111]. And “to increase ... parallelism[,] erase blocks ... are logically linked to form metablocks.” EX1009, ¶[0112]. Because Sinclair-367 and Reiter disclose analogous flash systems, and planes and metablocks are common in the flash memory art, a POSITA would have a reasonable likelihood of success implementing them in Ban’s flash device. EX1003, ¶174.

Reiter, ZBC, and Sinclair-367 are analogous art—relating to storage devices, including accessing blocks of memory. EX1003, ¶174.

C. Claim-by-Claim Analysis

1. Claim 5

(a) 5[pre]

EX1003, ¶175; Section VIII.D.1.

(b) 5[a]

A POSITA would be motivated to implement Reiter’s flash system 100 (*storage device*) to accept and execute ZBC commands. Section VIII.D.1.c. The ZBC command set includes a report zones command. The report zones command

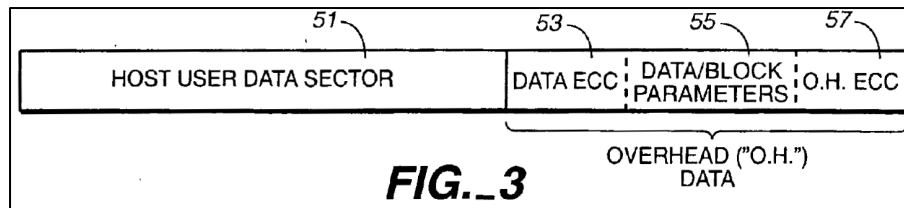
optionally “return[s]” (*transmit*) a “[l]ist [of] zones in the zoned block device” back *to [the] host* via “parameter data” that includes the length of the zone list and zone descriptors indicating the starting LBA and length of each zone (*the information identifying the number of structures to a host*). EX1003, ¶176; EX1006, 25, Tables 11-13:

Table 12 — REPORT ZONES parameter data								
Bit Byte	7	6	5	4	3	2	1	0
1	(MSB)							
...	ZONE LIST LENGTH (n-63)							
3	(LSB)							
4	Reserved							SAME
5	Reserved							
...								
63								
Zone descriptors list								
64	Zone descriptor [first]							
...								
127								
	⋮							
n-63	Zone descriptor [last]							
...								
n								

Table 13 — Zone descriptor format								
Bit Byte	7	6	5	4	3	2	1	0
0	Reserved				ZONE TYPE			
1	ZONE CONDITION				Reserved			RESET
2	Reserved							
...								
7								
8	(MSB)							
...	ZONE LENGTH							
15	(LSB)							
16	(MSB)							
...	ZONE START LBA							
23	(LSB)							
24	(MSB)							
...	WRITE POINTER LBA							
31	(LSB)							
32	Reserved							
...								
63								

Because flash system 100 transmits the zone list information back to the host, a POSITA would understand that flash system 100 also stores that information (*store information identifying a number of structures in the flash memory*). Or a POSITA would be motivated to store such information. EX1003, ¶177. As opposed to wasting time re-calculating such information with every command requiring zone metadata, flash system could store zone metadata and update it as zone settings are changed.

Sinclair-367's erase blocks use a "single sector of data" to *store* overhead data, including error correction codes (ECC) 53 for data and data/block parameters 55 (*metadata*). EX1009, FIG. 3:



Therefore, every block in a logical unit, volume, and partition includes this single sector of overhead data (*independent basis for flash memory associated with each one of the block devices*). EX1003, ¶¶178-80. Sinclair-367 also includes the “time of programming ... as a field in [the] sector or page overhead data” (*metadata representing a time since one or more physical storage locations associated with the one or more block devices has been a subject of a write operation*). EX1009, ¶[0111].

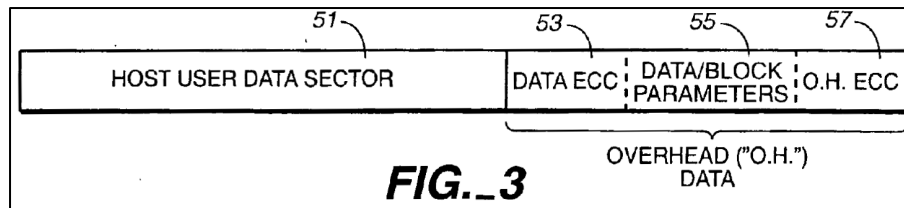
2. Claim 10

(a) 10[pre]

EX1003, ¶181; Section VIII.D.1.

(b) 10[a]

A POSITA would be motivated to incorporate Reiter’s flash system 100 (*storage device*) to *store* physical blocks with Sinclair-367’s metadata sector (*for each segment having an erase unit which stores valid data*). EX1003, ¶182; Section IX.B. Sinclair-367’s erase blocks use a “single sector of data” to *store* overhead data, including ECC 53 (*metadata*). EX1009, FIG. 3:



Therefore, every block in a logical unit, volume, and partition includes this single sector of overhead data (*independent basis for flash memory associated with each one of the block devices*). EX1003, ¶¶183-84. Sinclair-367 also includes the “time of programming ... as a field in [the] sector or page overhead data” (*metadata representing a time since one or more physical storage locations associated with the one or more block devices has been a subject of a write operation*). EX1009, ¶[0107].

(c) 10[b]

A POSITA would be motivated to implement flash controller 104 of Reiter’s flash memory storage 104 (*storage device*) to *compare* the time of programming data in a physical block (*values representing time since data was programmed*) against a retention period (*at least one threshold*). EX1003, ¶185. A retention period is the period of time where flash memory reliably stores a value, and flash memory does not reliably store the value after that period of time. EX1003, ¶185. Checking a time-since-programming alerts flash controller 104 to physical blocks becoming unreliable, allowing flash controller 14 to remedy the situation and reduce data loss. EX1003, ¶185.

Sinclair-367 recognizes that flash memory has a retention period problem. EX1003, ¶¶186-87. For instance, Sinclair-367 discloses how a memory cell relies on “threshold voltage window ... divided into a number of ranges” to create storage states. EX1009, ¶[0007]. But the maintaining the voltage level within one of the slim ranges associated with certain storage states is difficult. “[S]torage levels do shift” shift over usage of the memory cell and surrounding memory cells. EX1009, ¶[0007]. One way to predict and stop an intolerable shift in voltage levels is to check if the time since programming is nearing the average retention period of a memory cell. EX1003, ¶186. This can alert flash controller 104 of potential data loss, and provide an opportunity to save the data. EX1003, ¶186.

(d) 10[c]

Upon nearing or reaching the average retention period (*upon detection that a specific one of the values has satisfied one or more thresholds*), a POSITA would be motivated to copy data out of the unreliable block past its retention period (*copy data stored at one of the one or more physical storage locations associated with the specific one of the values*) and into a working physical block (*from a first physical erase unit to a second physical erase unit*). EX1003, ¶188-89. The data is saved by copying data away to reset the voltage levels, and copying data away from a bad block to a working block is a common, simple solution. EX1003, ¶188. Copying data into a new block, such as through garbage collection is faster than

reading, erasing, and rewriting the same block. *See also* EX1009, ¶¶[0108]-[0109], [0114], [0132], [0177].

(e) 10[d]

Sinclair-367 discloses that copying data away from a failing block to a new block also includes remapping the logical address from the old physical address to the new physical address (*remap a logical address for the data being copied from the first physical erase unit to the second physical erase unit*). EX1003, ¶190-91; EX1009, ¶[0109]. For example, the “logical address of the data ... mapped into th[e] ... address of any substitute erase block” if “the primary erase block is defective.” EX1009, ¶[0109].

3. Claim 11

(a) 11[pre]

EX1003, ¶192; Section IX.C.2.

(b) 11[a]

EX1003, ¶193; Section VIII.D.10.b.

(c) 11[b]-[c]

Sinclair-367 discloses that copying data away from a failing block to a new block also includes remapping the logical address from the old physical address to the new physical address (*remap a logical address for the data being copied from the first physical erase unit to the second physical erase unit*). EX1003, ¶¶194-95; EX1009, ¶¶[0109], [0114], [0132], [0177]. For example, the “logical address of the

data ... mapped into th[e] ... address of any substitute erase block” if “the primary erase block is defective.” EX1009, ¶[0109].

4. Claim 12

(a) 12[pre]

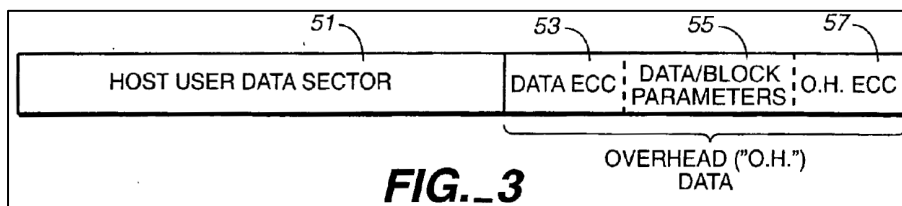
EX1003, ¶196; Section VIII.D.1.

(b) 12[a]

EX1003, ¶197; Section IX.C.3.

(c) 12[b]

Sinclair-367’s erase blocks use a “single sector of data” to *store* overhead data, including error correction codes (ECC) 53 for data and data/block parameters 55 (*metadata*). EX1009, ¶[0107], FIG. 3:



Therefore, every block in a logical unit, volume, and partition includes this single sector of overhead data (*independent basis for flash memory associated with each one of the block devices*). EX1003, ¶¶198-200. Sinclair-367 also discloses that “parameter 55 may include a quantity related to the number of program/erase cycles experiences by the erase block” (*wherein the stored metadata is dependent on an erase count of associated physical memory space*). EX1009, ¶[0108].

(d) 12[c]-[e]

Sinclair-367 discloses that the “number of program/erase cycles” is “used in a wear leveling algorithm,” where “[LBA]s are regularly re-mapped to a different physical block addresses in order to even out the usage.” EX1009, ¶[0108].

A POSITA would understand that the remapping hinges on the number of program/erase cycles (*metadata*) being high enough to indicate a heavily used erase block (*at least one threshold*). EX1003, ¶¶201-04; *see* EX1009, ¶¶[0108]-[0109].

Moreover, detection that an erase block is heavily used (*upon detection that specific metadata has satisfied the one or more thresholds*) causes LBAs to be remapped. A POSITA would understand, or be motivated to modify so, that remapping a logical address to a different physical block means the data of the old block is copied to the new block prevent data loss (*copy data stored at one of the one or more physical storage locations associated with the specific metadata from a first one of the physical erase units for which the failure condition has not been detected by said storage device to a second one of the physical erase units for which the failure condition has not been detected by said storage device*). EX1003, ¶203.

5. Claim 13

(a) 13[pre]-[b]

EX1003, ¶¶205-06; Sections VIII.D.9.a-c.

(b) 13[c]

A POSITA would be motivated to modify Reiter’s flash system 100 (*storage device*) to implement Sinclair-367’s bad block management. Section IX.B. Sinclair-367 discloses “identification of any defective cells within the erase block,” (*detect a failure condition of the physical erase units*) and *remapping* the “logical address” (*logical erase unit which corresponds to one of the physical erase units for which the failure condition is detected*) with the “address of any substitute erase block in case the primary erase block is defective” (*instead correspond to one of the physical erase units for which the failure condition has not been detected*) for later use (*utilize the remapped correspondence to service one or more of the incoming data access requests*). EX1003, ¶207; EX1009, ¶¶[0107]-[0108].

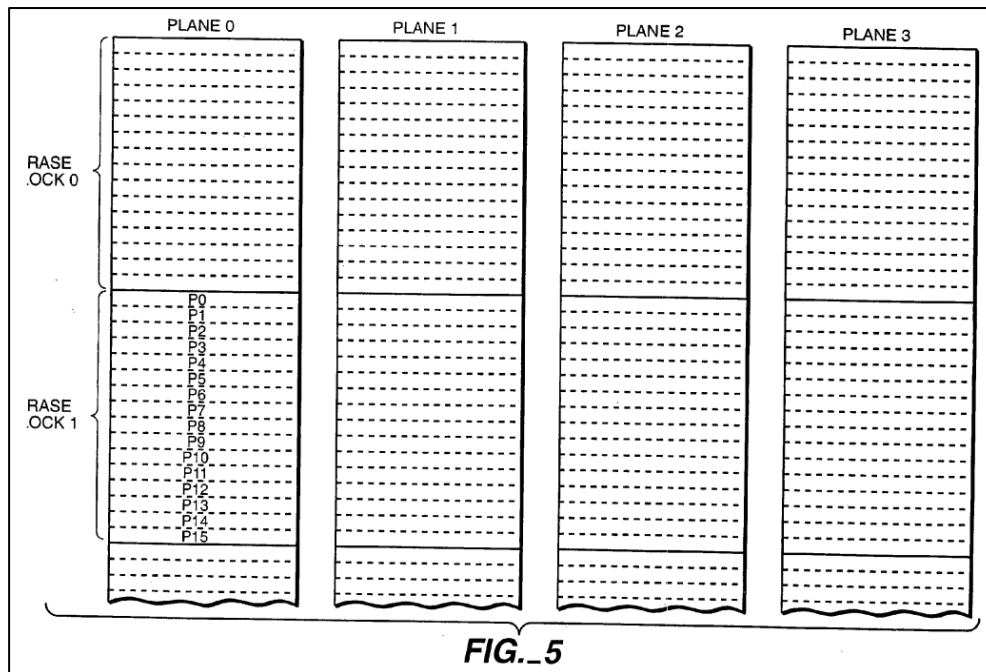
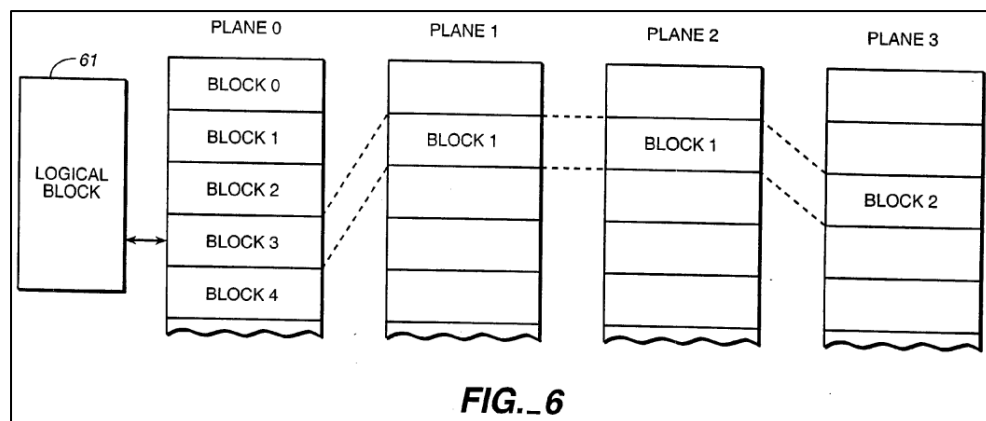
6. Claim 18

(a) 18[pre]

EX1003, ¶208; Section VIII.D.1.

(b) 18[a]

A POSITA would be motivated to implement Reiter’s *flash* media 118 as “divided into two or more planes” (*multiple planes*) per Sinclair-367’s teachings. EX1003, ¶¶209-10; EX1009, ¶[0111], FIGs. 5-6 (below); Section IX.B.



(c) 18[b]

A ZBC write command includes an LBA and “transfer length.” EX1007, Table 112.

Table 112 — WRITE (32) command									
Byte	Bit	7	6	5	4	3	2	1	0
0		OPERATION CODE (7Fh)							
1		CONTROL							
2		Reserved							
...									
5									
6		Reserved			GROUP NUMBER				
7		ADDITIONAL CDB LENGTH (18h)							
8	(MSB)	SERVICE ACTION (000Bh)							
9									(LSB)
10		WRPROTECT			DPO	FUA	Reserved	Obsolete	Reserved
11		Reserved							
12	(MSB)	LOGICAL BLOCK ADDRESS							
...									
19									(LSB)
20	(MSB)	EXPECTED INITIAL LOGICAL BLOCK REFERENCE TAG							
...									
23									(LSB)
24	(MSB)	EXPECTED LOGICAL BLOCK APPLICATION TAG							
25									(LSB)
26	(MSB)	LOGICAL BLOCK APPLICATION TAG MASK							
27									(LSB)
28	(MSB)	TRANSFER LENGTH							
...									
31									(LSB)

“TRANSFER LENGTH ... specifies the ... logical blocks ... that shall be transferred ... starting [from] ... the [LBA]” EX1007, 198. With a large transfer length, a ZBC *write* command causes several write operations across *sequential addresses*. EX1003, ¶212.

To increase throughput for these large write commands, a *POSITA* would be motivated to implement Reiter’s flash media 118 with multiple planes, and metablocks spanning the multiple planes, as disclosed in Sinclair-367. EX1009, ¶¶[0111]-[0112]; Sections VII.B.4.a, VII.B.1. “All blocks of the metablock are erased together, and pages of each block are generally programmed and read

simultaneously” (*identify a commonly addressed page and concurrently write data accompanying the set of incoming write requests into respective ones of the multiple planes in a manner corresponding to the commonly addressed page*).

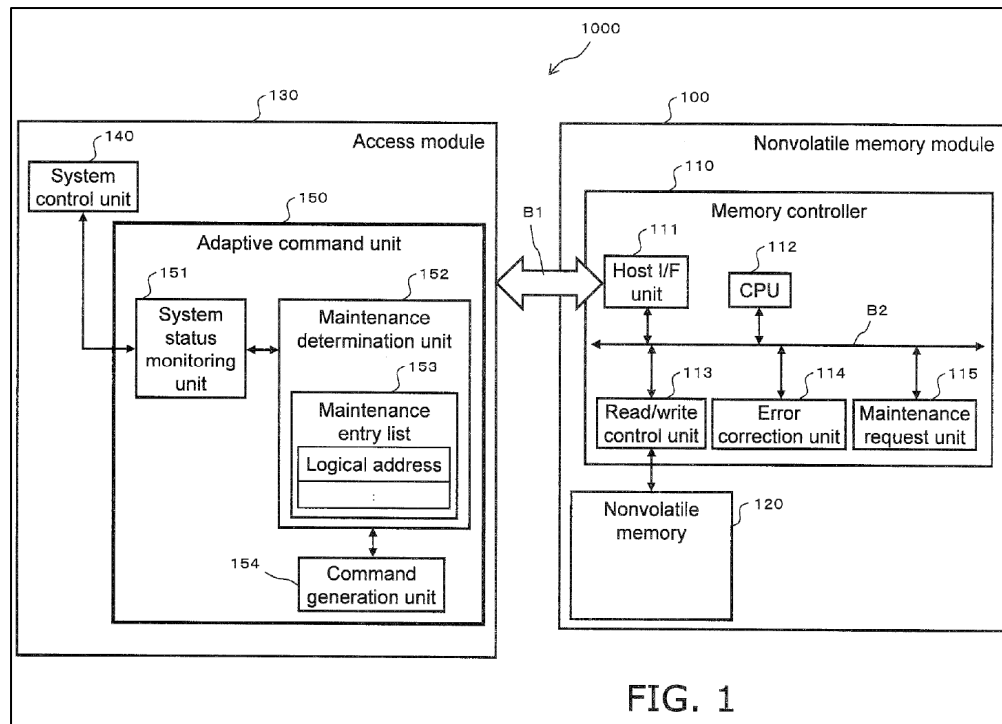
EX1003, ¶¶213-14; EX1009, ¶[0112].

X. GROUND III: CLAIMS 14 AND 16-17 ARE RENDERED OBVIOUS BY REITER IN LIGHT OF ZBC, SBC-4, SPC-4, AND YAMADA

A. Yamada

U.S. Publication No. 2011/0271032 (“Yamada”) was filed on July 27, 2010, was published on November 3, 2011, and is prior art under AIA 35 U.S.C. § 102(a)(1).

Yamada discloses a nonvolatile memory module with flash memory and memory controller. EX1003, ¶¶55-58; EX1010, FIG.1. Read and write commands are issued to the nonvolatile memory module by an access module. EX1010, FIG. 1:



Yamada complains about maintenance operations, such as garbage collection or bad block management, delaying normal storage operations for flash memory devices. EX1010, Abstract. Yamada tries to perform maintenance operations during the lulls between sets of normal storage operations. EX1010, Abstract. To do so, Yamada implements a maintenance request unit on the nonvolatile memory module, and a maintenance determination unit on the access module. EX1010, ¶¶[0081]-[0082], [0089], FIG. 1. The maintenance request unit determines when certain parts of flash memory needs garbage collection or bad block management, and transmits a request for maintenance to the maintenance determination unit. EX1010, ¶¶[0081]-[0082]. The maintenance determination unit, by being on the access module, monitors the access module's status. EX1010, Abstract, ¶[0089]. When

the maintenance determination unit finds downtime in normal processing, the maintenance determination unit performs requested maintenance operations. EX1010, Abstract, ¶[0089].

B. Motivation to Combine

A POSITA would have been motivated to implement Reiter's flash system to execute ZBC commands (Section VIII.B.2.) and with Yamada's maintenance request and determination units in Reiter's flash system and host processor, respectively. EX1003, ¶215-16. Reiter complains that "[g]arbage collection routines impact performance of the flash memory system by utilizing processor resources and potentially delaying write operations." EX1004, 4:23-26. In Yamada, "maintenance is performed without being ... performed together with normal system processing," but "with sufficiently long processing time." EX1010, Abstract. The maintenance request unit on the flash device determines and requests maintenance operations from the maintenance determination unit. EX1010, Abstract, ¶¶[0032], [0081]-[0082], [0089]. The maintenance determination unit monitors status of the host, "determin[es] that the access device is not engaged in normal processing," and performs maintenance. EX1010, Abstract, ¶¶[0018]-[0019], [0022]. Because maintenance scheme is implemented on a nonvolatile system analogous like that in Reiter, a POSITA would have a reasonable expectation of success implementing Reiter's flash system with this scheme. EX1003, ¶215.

Reiter, ZBC, Sinclair-367, and Yamada are analogous art—relating to storage devices. EX1003, ¶216.

C. Claim-by-Claim Analysis

1. Claim 14

(a) 14[pre]

EX1003, ¶217; Section VIII.D.1.

(b) 14[a]

EX1003, ¶218; Section IX.C.2.b.

(c) 14[b]

EX1003, ¶219; Section IX.C.2.c.

(d) 14[c]

Yamada discloses maintenance determination unit 152 in the host that receives a “logical address included in [a] maintenance request transmitted from ... maintenance request unit 115” on the flash storage device (*transmit information to a host device wherein such transmitted information conveys address information*). EX1010, ¶[0089]. A POSITA would be motivated to incorporate Yamada’s maintenance determination unit and maintenance request unit scheme because Yamada’s implementation reduces the delay caused by garbage collection procedures. EX1003, ¶220. Reiter makes this complaint expressly, stating that “[g]arbage collection routines impact performance of the flash memory system by utilizing processor resources and potentially delaying write operations to the flash

media.” EX1004, 4:23-26. The maintenance determination unit, by being on the host can determine the “system status” of the host, and is able to issue “data maintenance ... without being required to be performed together with normal system processing.” EX1010, Abstract.

Given the use of Yamada’s maintenance determination and request units, a POSITA would be motivated to trigger a maintenance request upon detecting a physical block time since programming nears or reaches the average retention period (*responsive to detection that specific metadata has satisfied the one or more thresholds*). EX1003, ¶¶221-22. As indicated by the name of Yamada’s units, maintenance requests should be triggered for garbage collections, including operations reduce the risk of lost data. As stated with Section IX.C.1.d, after the average retention period, a memory cell can be unreliable, risking loss of the data stored in the cell. Saving that data from loss is done through garbage collection or maintenance operations, such as those issued from a maintenance request.

2. Claim 16

(a) 16[pre]

EX1003, ¶223; Section VIII.D.1.

(b) 16[a]

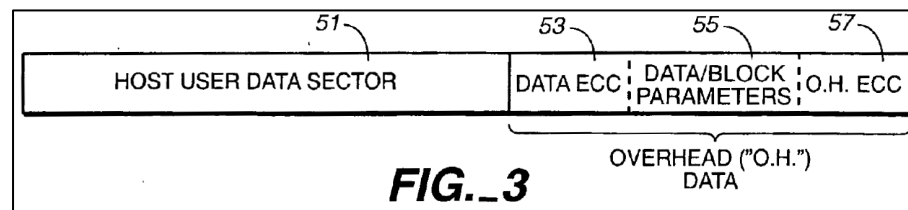
EX1003, ¶224; Section VIII.D.10.b.

(c) 16[b]

EX1003, ¶225; Section XI.A.1.d-e.

(d) 16[b][i]

With the writing capabilities of Reiter's flash controller 104 (*circuitry*), Sinclair-367's erase blocks use a "single sector of data" to *store* overhead data, including error correction codes (ECC) 53 for data and data/block parameters 55 (*metadata*). EX1009, FIG. 3:



Every block in a logical unit, volume, and partition includes this single sector of overhead data (*independent basis for flash memory associated with each one of the block devices*). EX1003, ¶¶226-28. Sinclair-367 also discloses that "parameter 55 may include a quantity related to the number of program/erase cycles experiences by the erase block" (*wherein the stored metadata is dependent on an erase count of associated physical memory space*).

(e) 16[b][ii]

Yamada discloses maintenance determination unit 152 in the host that receives a "logical address included in [a] maintenance request transmitted from ... maintenance request unit 115" on the flash storage device. EX1010, ¶[0089]. A POSITA would be motivated to incorporate Yamada's maintenance determination unit and maintenance request unit scheme because Yamada's implementation

reduces the delay caused by garbage collection procedures. EX1003, ¶¶229-31. Reiter makes this complaint expressly, stating that “[g]arbage collection routines impact performance of the flash memory system by utilizing processor resources and potentially delaying write operations to the flash media.” EX1004, 4:23-26. The maintenance determination unit, by being on the host can determine the “system status” of the host, and is able to issue “data maintenance ... without being required to be performed together with normal system processing.” EX1010, Abstract.

Given the use of Yamada’s maintenance determination and request units, a POSITA would find a query for specific metadata obvious in Yamada. EX1003, ¶230. Yamada’s “non-volatile memory 1020” (*e.g.*, Reiter’s flash media on flash system 100) “stores management information including the information about the importance level of data to enable the management information to be shared between different access modules” (*hosts*). EX1010, ¶[0190]. A POSITA would understand that sharing such metadata requires a *query, issued by a host device, wherein said query is specific to one of the block devices*, where the query causes (*responsive to the query*) the storage device to *identify specific metadata, associated with the specified one of the block devices, and transmit information to the host device that conveys the specific metadata*. EX1003, ¶230.

3. Claim 17

(a) 17[pre]

EX1003, ¶232; Section VIII.D.1.

(b) 17[a]

EX1003, ¶233; Section XI.A.1.d.

(c) 17[b]-[c]

Yamada discloses maintenance determination unit 152 in the host that receives a “logical address included in [a] maintenance request transmitted from ... maintenance request unit 115” on the flash storage device (*transmit information to a host wherein such transmitted information*). EX1010, ¶[0089]. A POSITA would be motivated to incorporate Yamada’s maintenance determination unit and maintenance request unit scheme because Yamada’s implementation reduces the delay caused by garbage collection procedures. EX1003, ¶234. Reiter makes this complaint expressly, stating that “[g]arbage collection routines impact performance of the flash memory system by utilizing processor resources and potentially delaying write operations to the flash media.” EX1004, 4:23-26. The maintenance determination unit, by being on the host can determine the “system status” of the host, and is able to issue “data maintenance ... without being required to be performed together with normal system processing.” EX1010, Abstract.

Given the use of Yamada’s maintenance determination and request units, a POSITA would be motivated to trigger a maintenance request upon detecting a high

number of program/erase cycles (*responsive to detection that specific metadata has satisfied the one or more thresholds*). EX1003, ¶¶235-36. As indicated by the name of Yamada’s units, maintenance requests should be triggered for garbage collections, including wear-level management for improving data retention. EX1003, ¶235. Program/erase counts for physical blocks are “used in a wear leveling algorithm ... in order to even out the usage (wear) of all erase blocks.” EX1009, ¶[0107]. Excessive wear causes erase blocks to become unreliable, risking loss of data stored in the erase block. Spreading out wear to reduce data loss is done through garbage collection or maintenance operations, such as those issued from a maintenance request.

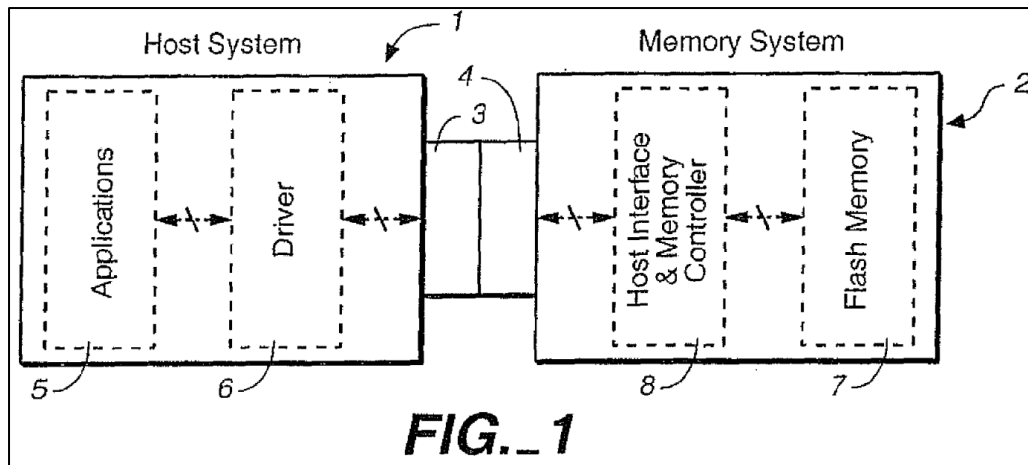
XI. GROUND IV: CLAIMS 1-2, 8-9, 18-22 ARE RENDERED OBVIOUS BY SINCLAIR-233

A. Overview of the Prior Art

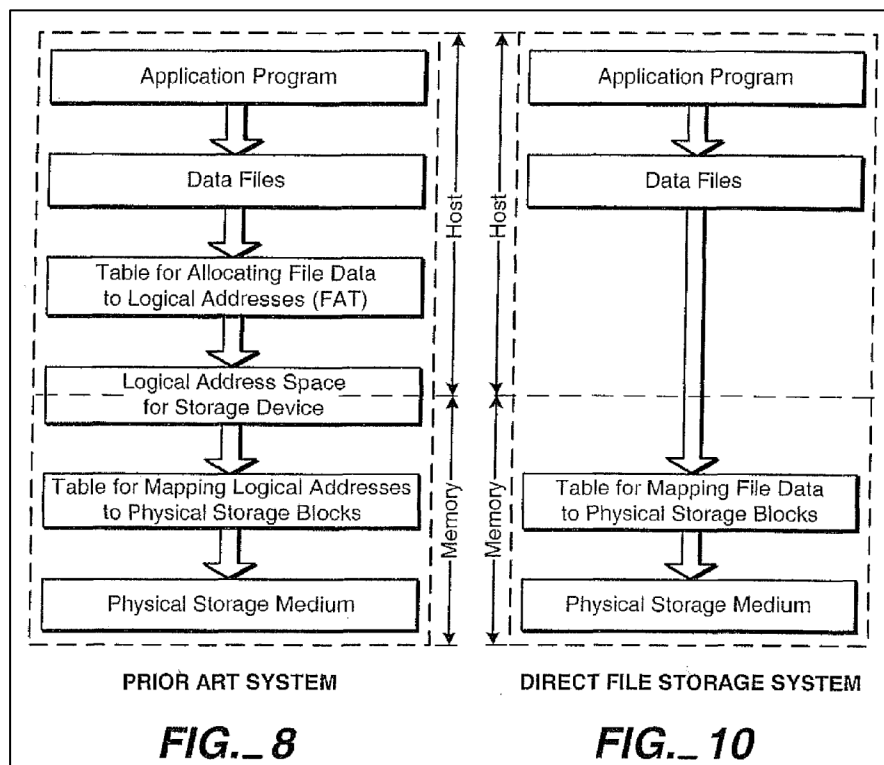
1. Sinclair-233

U.S. Patent 7,984,233 (“Sinclair-233”) was filed on May 5, 2010, was published on August 26, 2010, and issued on July 19, 2011. It is prior art under AIA 35 U.S.C. § 102.

Sinclair-233 discloses flash memory system 2, where flash memory system 2 has a direct file interface with the host. EX1003, ¶¶48-50; EX1005, Abstract, 13:24-34, FIG. 1:



A direct file interface means that commands operate on a filename or an offset address within a linear address space unique to a file. EX1005, 13:24-34. This is as opposed to conventional logical-to-physical address system, where the host commands target an LBA translated to a physical block address by the storage device. EX1005, FIGS. 8, 10:

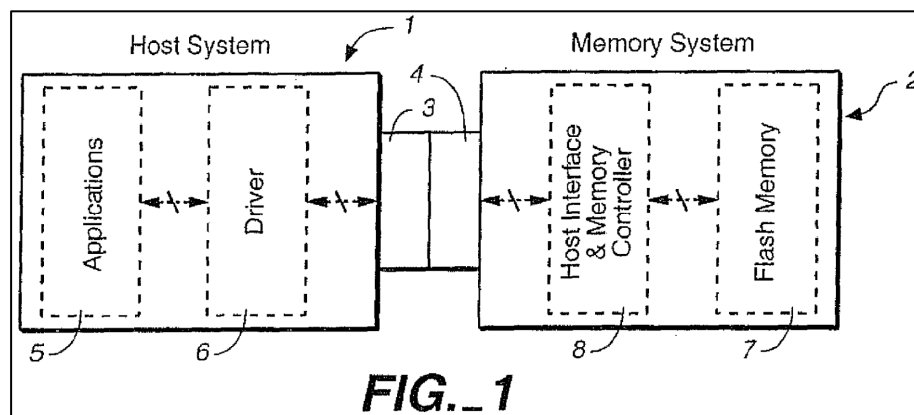


B. Claim-by-Claim Analysis

1. Claim 1

(a) 1[pre]

Sinclair-233 discloses a “memory system 2” (*storage device*) that operates on a “file based interface between the host and memory system[.]” EX1003, ¶¶237-38; EX1005, Abstract, 5:7-9, 5:26-28, FIG. 1:

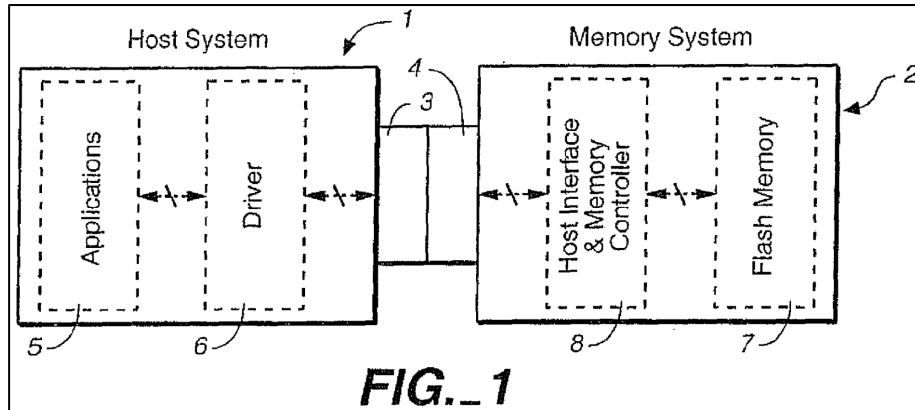


(b) 1[a]

Sinclair-233 discloses flash memory 7 (*flash memory*) made of memory cell groups called “*erase units*” where “such group[s] are the minimum number of memory cells that are erasable together.” EX1003, ¶¶239-40; EX1005, 1:30-34.

(c) 1[b]

Sinclair-233 discloses memory system 2 including a “Host Interface & Memory Controller 8” (*circuitry to receive from a host*) for “convert[ing] between logical addresses of data used ... [for] data programming and reading” (*and control execution of, incoming requests*). EX1003, ¶241; EX1005, 5:19-25, FIG. 1.



“A set of direct file interface commands from the host system supports the operation of the memory system.” EX1005, 13:36-45. The direct file interface commands (*incoming requests*) include an insert command and erase command (*data write requests and erase requests*). EX1003, ¶242; EX1005, FIG. 12A-B:

DATA COMMANDS		
Command	Parameters	Description
Write	< fileID >	Appends data to end of the specified file.
Insert	< fileID > < offset >	Inserts data within specified file at < offset > from start.
Update	< fileID > < offset >	Overwrites data in specified file at < offset > from start.
Read	< fileID > < offset >	Reads data from specified file at < offset > from start.
Remove	< fileID > < offset 1 > < offset 2 >	Removes data within specified file between < offset 1 > and < offset 2 > from start.

FIG._12A

FILE COMMANDS		
Command	Parameters	Description
Open	< fileID >	Requests resources to be made available for writing to the specified file.
Close	< fileID >	Resources may be removed immediately from the specified file, and the file may be scheduled for garbage collection.
Close_after	< fileID > < length >	Resources may be removed from the specified file after further transfer of < length> of file data from host, at which time the file may be scheduled for garbage collection.
Delete	< fileID >	Indicates that table entries and data for the specified file should be deleted. File data may be erased.
Erase	< fileID >	Indicates that table entries and data for the specified file should be deleted and that file data should be erased immediately.

FIG. 12B

The insert and erase command include a “fileID” and/or “offset” parameter (*incoming requests is accompanied by incoming address information*), “designat[ing] [a] file (<fileID>) at a particular offset (<offset>) within the file [to be] written ... or read.” EX1003, ¶¶243-45; EX1005, 13:36-45.

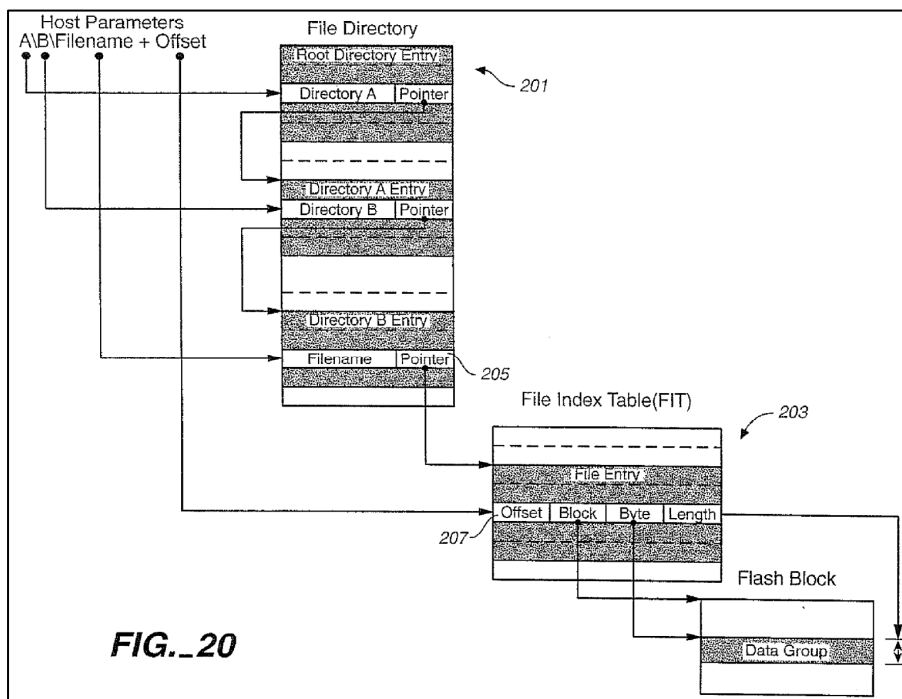
(d) 1[c]

EX1003, ¶246; Sections XI.B.1.e-h.

(e) 1[c][i]

Sinclair-233 discloses that insert and erase commands include a fileID. EX1005, FIGs. 12A-B. “The <fileID> parameter can be ... a full pathname for the file parameter.” EX1005, 14:47-61. Sinclair-233 discloses the pathname as including (1) a storage ID and (2) a directory pathname for the file (*derive, from the accompanying incoming address information, a first address portion and a second address portion*). EX1003, ¶¶247-48; EX1005, 14:47-61, 34:50-58. “The pathname that is used for files at the Direct-File Interface may identify the ID of the storage

device itself.” EX1005, 34:50-58. “The pathname [also] describes a hierarchy of directories and a file within the lowest level of directory.” EX1005 at 14:47-61. An example of the pathname detailing the directories and filename is provided in FIG. 20:



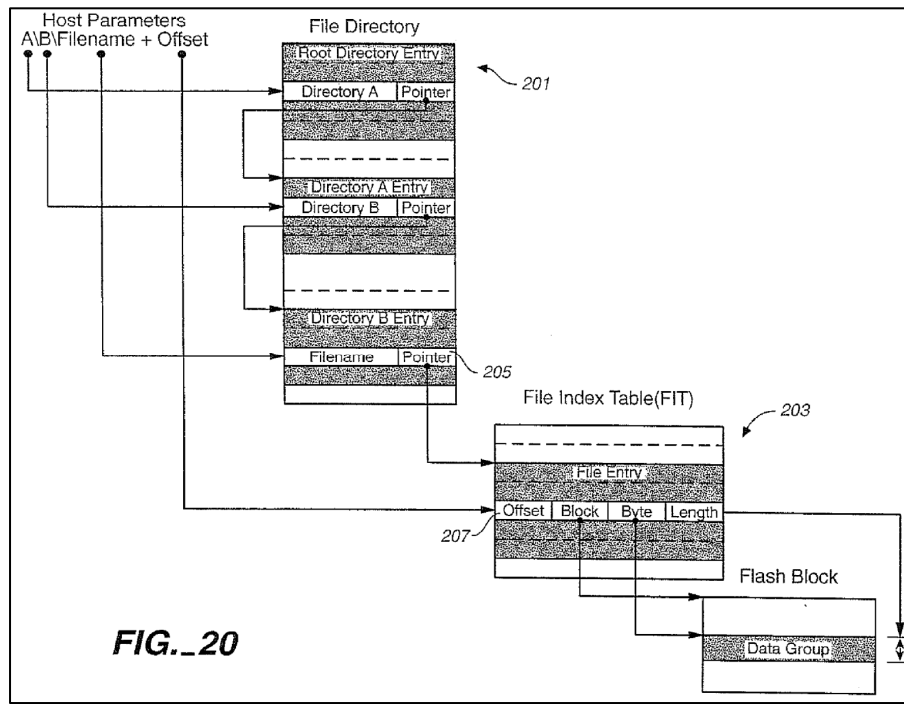
(f) 1[c][ii]

The storage ID constitutes the *first address portion*. Section XI.B.1.e. The storage ID *identifies* “the storage device itself” (*addressed block device from the first address portion*). EX1003, ¶¶249-50; EX1005, 34:50-58.

(g) 1[c][iii]

The directory-file pathname constitutes the *second address portion*. Section XI.B.1.e. “The pathname describes a hierarchy of directories and a file within the lowest level of directory.” EX1005, 14:47-61. Sinclair-233 discloses that the

“[p]ath segments may be delimited by ‘\’” (*use a first division operation on the second address portion*). EX1003, ¶¶251-52; EX1005, 14:56-59; *see also* EX1213, 56:25-57:10. This separates and extracts the directory names and the filename in order for traversing a file directory 201. EX1003, ¶251; EX1005, 30:40-60, FIG. 20:



Separating out the filename allows use of the filename for searching the corresponding directory entry to identify the file to be accessed (*an addressed segment within the addressed block device*). EX1003, ¶252; EX1005, 30:40-47.

A file *corresponds to a subset of the erase units of the flash memory*. EX1003, ¶¶253-54; EX1005, Abstract, FIG. 19 “Host system data files are written directly to large erase block flash memory system.” EX1005, Abstract. The “file

index table (FIT)” lists the set of blocks associated with a file. *See* EX1005, 29:35-52, FIG. 19. For instance, “FIG. 19 illustrates a sequence of index entries in a file index table (FIT) for one file.” EX1005, 29:35-38, FIG. 19:

File Index Table (FIT)				
	OFFSET	BLOCK	BYTE	LENGTH
Time 0 (Fig. 14A)	F0	001	D0	—
	F1	002	D1	—
Time 2 (Fig. 14C)	F0	001	D0	—
	F5	003	D6	—
	F6	004	D7	—
	F7	001	D5	—
	F8	002	D3	—
	F9	003	D4	—
	F10	002	D2	—
Time 4 (Fig. 14E)	F0	001	D0	—
	F5	003	D6	—
	F6	004	D7	—
	F7	001	D5	—
	F8	002	D3	—
	F9	003	D4	—
	F12	004	D10	—
	F13	005	D11	—
Time 5 (Fig. 15)	F0	001	D0	—
	F1	002	D1	—
	F2	003	D2	—
	F3	004	D3	—

FIG. 19

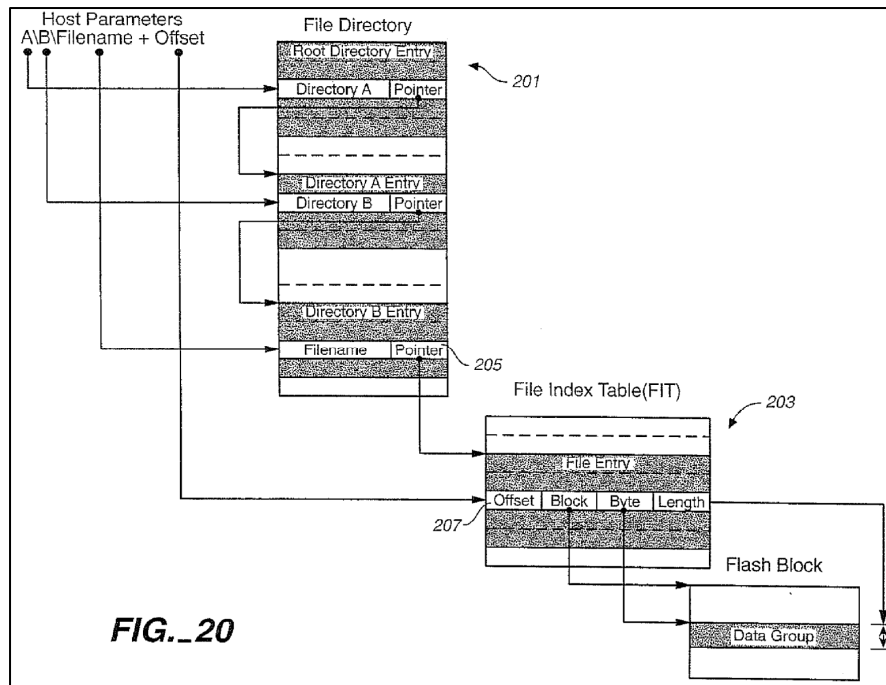
(h) 1[c][iv]

Sinclair-233 discloses insert commands (*data write requests*). Section XI.B.1.c. Each insert command includes a file pathname and an offset. EX1005, FIG. 12B. A delimiter operation on the file pathname (*first division operation*) with the offset *identifies* the file entry 207 (*third address portion*). EX1003, ¶255; *see also* EX1213, 56:25-57:10 Via the delimiter operation “[t]he pathname ... is used to traverse the hierarchy of the file directory” 201, where the “directory [is] then searched to find an entry 205 for the filename supplied by the host.” EX1005, 30:40-47. “When found, the entry 205 provides a pointer to a group of contiguous entries

in the FIT 203 for that file.” EX1005, 30:48-49, FIG. 20. “The offsets of these entries are then compared with the offset address supplied by the host in order to identify a correct entry 207.” EX1005, 30:50-53.

A POSITA would understand, or find it obvious, to incorporate apply integer division to the bitstring in entry 207 (*use a second division operation on the third address portion*) in order to *identify* a block number (*address value to select one of the erase units from the subset which corresponds to the addressed segment*) and a byte number (*storage location within the selected one of the erase units*). EX1003, ¶256.

A POSITA would understand entry 207 to be a bitstring with bitfields for the starting block number, starting byte number, and length of the data group. EX1003, ¶¶90-91, 257. “The selected entry 207 of the FIT 203, in this example, contains the physical block and byte of the memory location containing data to which access is being sought by the host.” See EX1010, 30:53-60, FIG. 20:



A POSITA would be motivated to use integer division operation on the obtained the entry 207 bitstring to extract the block number and byte number. EX1003, ¶¶258-59. Integer division by a certain power-of-two extracts bitfields out of bitstrings. EX1003, ¶258. Specifically, the quotient of integer division by certain power-of-two will extract the bitfields of the most significant bits. EX1003, ¶258. Integer division is preferable because it is fast, as division by a power-of-two on a bitstring is implement with a simple bit shift operation. EX1003, ¶258. The block number is the higher order or most significant bitfield (*i.e.*, leftmost bits in the bitstring) of entry 207 bitstring. *See* EX1005, FIG. 20. Applying integer division on the bitstring causes the block number to be extracted through the quotient. The remainder leaves a byte-length bitstring where the byte number is extracted with a another division operation. A POSITA would be motivated to implement integer

division via a bit shift operation to perform the extraction of these two bitfields quickly. EX1003, ¶¶258-59.

(i) 1[d]

Sinclair-233 discloses “Host Interface & Memory Controller 8” (*circuitry*) for “convert[ing] between logical addresses of data used ... [for] data programming and reading” (*execute each one of the data write requests*). EX1005, 5:19-25. Sinclair-233 describes the insert command (*each one of the data write requests*) as “[i]nserting data within specified file at <offset> from start” (*programming accompanying data into the storage location that was identified for that one of the data write requests*). EX1003, ¶¶260-61; EX1005, FIG. 12A:

DATA COMMANDS		
Command	Parameters	Description
Write	< fileID >	Appends data to end of the specified file.
Insert	< fileID > < offset >	Inserts data within specified file at < offset > from start.
Update	< fileID > < offset >	Overwrites data in specified file at < offset > from start.
Read	< fileID > < offset >	Reads data from specified file at < offset > from start.
Remove	< fileID > < offset 1 > < offset 2 >	Removes data within specified file between < offset 1 > and < offset 2 > from start.

FIG. 12A

(j) 1[e]

Sinclair-233 discloses “Host Interface & Memory Controller 8” (*circuitry*) is *also to execute each one of the incoming* erase commands (*erase requests*).

EX1005, 5:19-25. Sinclair-233 describes the erase command as “[i]ndicat[ing] ... data for the specified file should be deleted and that file data should be erased immediately” (*controlling erasure in the flash memory of one or more of the erase units in the subset corresponding to the addressed segment that was identified from the incoming address information accompanying the one of the incoming erase requests*). EX1003, ¶¶262-63; EX1005, FIG. 12B:

FILE COMMANDS		
Command	Parameters	Description
Open	< fileID >	Requests resources to be made available for writing to the specified file.
Close	< fileID >	Resources may be removed immediately from the specified file, and the file may be scheduled for garbage collection.
Close_after	< fileID > < length >	Resources may be removed from the specified file after further transfer of < length> of file data from host, at which time the file may be scheduled for garbage collection.
Delete	< fileID >	Indicates that table entries and data for the specified file should be deleted. File data may be erased.
Erase	< fileID >	Indicates that table entries and data for the specified file should be deleted and that file data should be erased immediately.

FIG. 12B

2. Claim 2

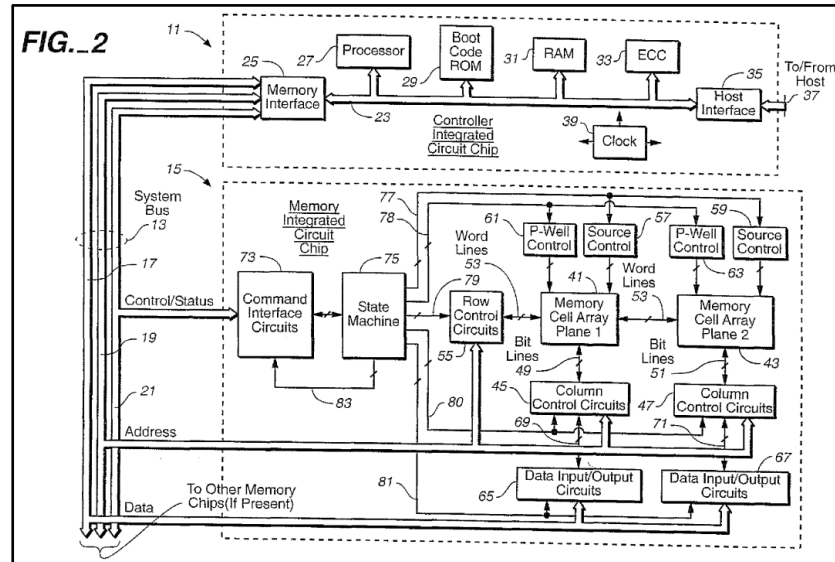
(a) 2[pre]

EX1003, ¶264; Section XI.B.1.

(b) 2[a]

Sinclair-233 discloses Host Interface & Memory Controller 8” (*circuitry*) “is usually implemented on a single integrated circuit chip 11,” which includes

processor 27 (*comprises at least one processor*). EX1003, ¶¶265-66; EX1005, 5:26-40, FIG. 2:



(c) 2[b]

Sinclair-233 discloses the controller on memory system's 2 (*storage device*) “interfaces with system bus 13,” where the “functions normally connected to the bus are a processor 27[] [and] ... read-only-memory (ROM) 29 containing code to initialize (‘boot’) the system” (*comprises instructions stored on at least one physical storage medium, said instructions when executed to control the at least one processor*). EX1003, ¶¶267-68; EX1005, 5:41-56.

3. Claim 8

(a) 8[pre]

EX1003, ¶269; Section XI.B.1.

(b) 8[a]

The bitstring of entry 207 constitutes the *third address portion*. Section XI.B.1.h. Sinclair-233 discloses a file containing block numbers within a range of sequentially numbered blocks (*sequential range of locations that correspond to the erase units in the subset*). EX1003, ¶¶270-71; EX1005, 29:35-52, FIG. 19. For example, “FIG. 19 illustrates a sequence of index entries in a file index table (FIT) for one file.” EX1005, 29:35-38, FIG. 19:

File Index Table (FIT)				
	OFFSET	BLOCK	BYTE	LENGTH
FIG. 19	Time 0 (Fig. 14A)	F0	001	D0
		F1	002	D1
		F0	001	D0
		F5	003	D6
	Time 2 (Fig. 14C)	F6	004	D7
		F7	001	D5
		F8	002	D3
		F9	003	D4
		F10	002	D2
		F0	001	D0
	Time 4 (Fig. 14E)	F5	003	D6
		F6	004	D7
		F7	001	D5
		F8	002	D3
		F9	003	D4
		F12	004	D10
		F13	005	D11
		F14	004	D9
	Time 5 (Fig. 15)	F0	001	D0
		F1	002	D1
		F2	003	D2
		F3	004	D3

The FIT entries show the file spanning a range of five blocks—blocks “001” to “005.” EX1005, FIG. 19. Entry 207 refers to a location within that sequential range of blocks. EX1005, FIG. 19.

4. Claim 9

(a) 9[pre]

EX1003, ¶272; Section XI.B.3.

(b) 9[a]

A POSITA would be motivated to implement the delimiter operation (*first division operation*) as a *subdividing* the file pathname (*value obtained from the incoming address information accompanying each of the write requests*) into directory names and the filename. EX1003, ¶¶273-75. Integer division by a power-of-two extracts bitfields out of bitstrings. EX1003, ¶273. Integer division is preferable because it is fast, as division by a power-of-two on a bitstring is implement with a simple bit shift operation. EX1003, ¶273. A file pathname is a long bitstring where each directory name, file name, and ‘\’ character is a bitfield, thereby requiring multiple operations to identify and extract each bitfield. EX1003, ¶273. Therefore, the speed of integer division is preferable. EX1003, ¶273. Additionally, simple operations like bit shifting are more ideal on external devices, such as storage devices, where processing and storage resources are limited. EX1003, ¶273. A POSITA would be motivated to implement delimiting with integer division by a power-of-two to subdivide the file pathname *into respective bitfields*. EX1003, ¶274.

A POSITA would understand, or find it obvious, to apply integer division by a power-of-two (*second division operation*) to the entry 207 bitstring (*value*

obtained from the incoming address information accompanying each of the write requests), subdividing it into block-number, byte-number, and length bitfields (into respective bitfields). EX1003, ¶¶274-75; Section XI.B.1.h.

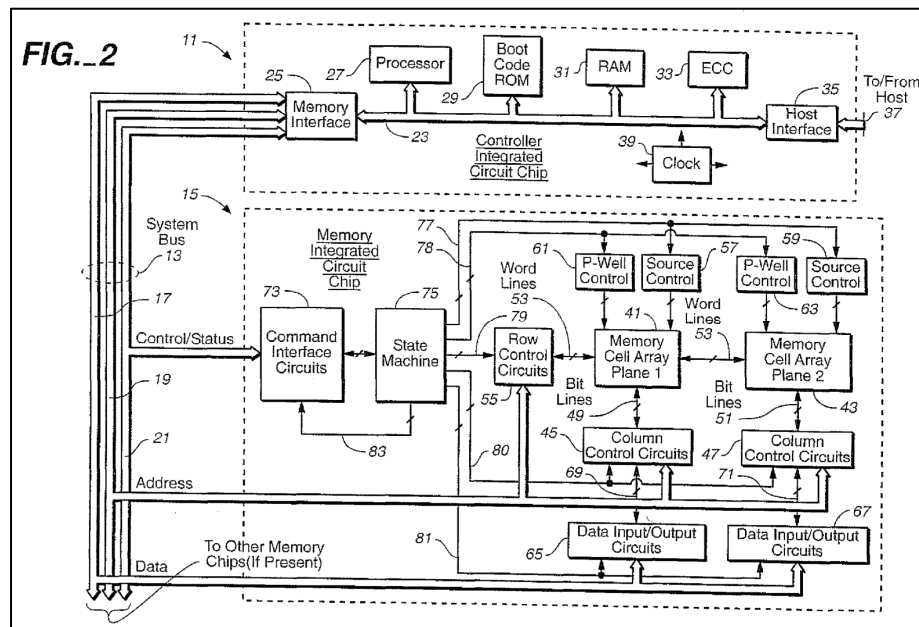
5. Claim 18

(a) 18[pre]

EX1003, ¶276; Section XI.B.1.

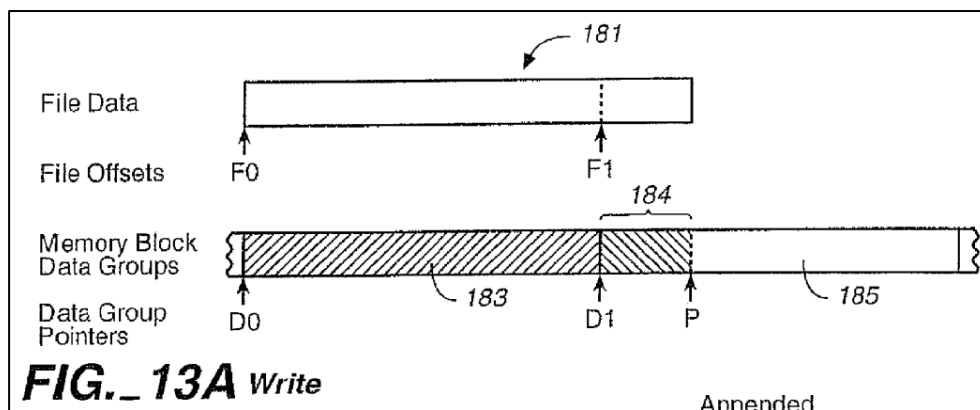
(b) 18[a]

Sinclair-233 discloses “memory chip 15 ... typically contains an array of memory cells organized into multiple sub-arrays or planes, two such planes 41 and 43 being illustrated for simplicity but more, such as four or eight such planes, may instead be used.” EX1003, ¶277-78; EX1005, 5:57-61, FIG. 2:

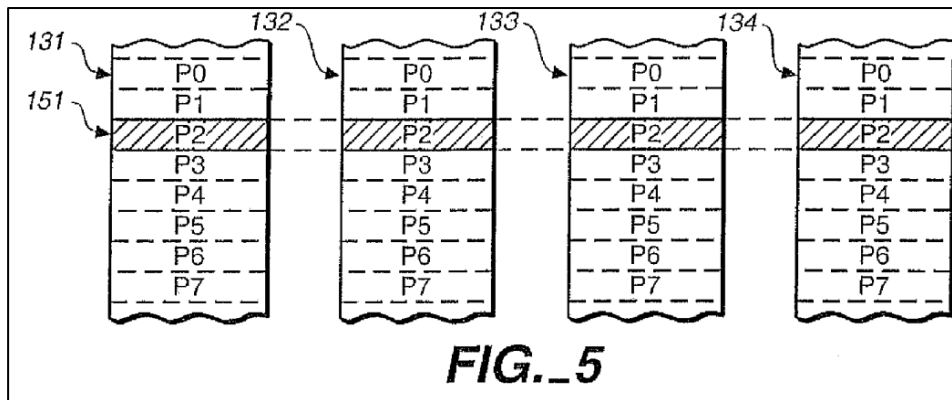


(c) 18[b]

A POSITA would understand that Sinclair-233 discloses data write commands, such as the insert command, where the data to be written spans across several sequential addresses (*for each one of the set of data write requests having sequential addressing*). EX1003, ¶279. For instance, “[r]eferring to FIG. 13A,” “data file 181 ... is larger than the storage capacity of one block.” EX1005, 16:46-50. “A portion 184 of ... data file 181 is therefore also written into ... second block 185.” EX1005, 16:50-51, FIG. 13A:



To increase throughput for these large write commands, a POSITA would be motivated to use metapages as disclosed in Sinclair-233. EX1003, ¶280. Per Sinclair-233, “in order to increase the memory system operational parallelism, ... pages within two or more blocks may be logically linked into metapages.” EX1005, 7:62-64, FIG. 5:



“The metapage 151, for example, includes page P2 in of each of the four blocks” (*identify a commonly addressed page*). EX1005, 7:66-8:3. “[M]etapages” include “pages in separate blocks in different planes.” EX1005, 8:3-7. This allows “for high system performance” due to “program[ming] and read[ing] the maximum amount of data in parallel across all four planes” (*write data accompanying the set of incoming data access requests concurrently into respective ones of the multiple planes*). EX1003, ¶¶280-81; EX1005, 83-12. A POSITA would take advantage of this parallelism large write commands or multiple write commands spanning several sequential addresses. EX1003, ¶¶280-81.

6. Claim 19-21

Claims 19-21 is substantively identical to claim 1 incorporated with claims 9 and 22. EX1003, ¶282; Sections VIII.D.11, XI.B.1.

7. Claim 22

(a) 22[pre]

EX1003, ¶283; Section XI.B.1.

(b) 22[a]

The file pathname constitutes the *second address portion*. Section XI.B.1.e. The file pathname is a series of bitfields corresponding to directory names, the filename, and the “\” character. EX1003, ¶¶284-85; EX1005, 14:47-61. When dividing the pathname into these bitfields with a delimiter operation (*first division operation*), (Section XI.B.1.g.) the filename bitfield (*first set of bits*) is *separated* from the next “\” character bitfield (*mutually exclusive bits of the second set of bits*). EX1003, ¶¶284-85; EX1005, 30:40-49, FIG. 20; Section XI.B.1.g-h. The separate filename *identifies* the file (*addressed segment*). EX1005, 30:40-49, FIG. 20; Section XI.B.1.g.

XII. MANDATORY NOTICES AND FEES

A. Real Party-In-Interest

The real party-in-interest for Petitioner is Samsung Electronics Co., Ltd. and Samsung Electronics America, Inc.

B. Related Matters

Radian Memory Systems LLC, v. Samsung Electronics Co., Ltd., AND Samsung Electronics America, Inc., 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 5th Avenue, 9th Floor New York, New York 10016 Tel: (212) 849-7000 Fax: (212) 849-7100 Nima Hefazi (Reg. No. 63,658) QUINN EMANUEL URQUHART & SULLIVAN, LLP 865 S. Figueroa St., 10 th Floor Los Angeles, California 90017 nimahefazi@quinnemanuel.com Tel: (213) 443-3000 Fax: (213) 443-3100

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.

XIII. CONCLUSION

For these reasons, Petitioner requests *inter partes* review of the '657 patent.

DATED: August 5, 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 13,179. 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: August 5, 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 '657 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: August 6, 2025

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

Attorney for Petitioners