

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SANDISK TECHNOLOGIES, INC. AND WESTERN DIGITAL
TECHNOLOGIES, INC.,
Petitioner

v.

POLARIS POWERLED TECHNOLOGIES, LLC,
Patent Owner.

IPR2025-00516
U.S. Patent No. 8,601,346

**PETITION FOR *INTER PARTES* REVIEW
UNDER 35 U.S.C. § 312 AND 37 C.F.R. § 42.104**

TABLE OF CONTENTS

PETITIONER’S EXHIBIT LIST	5
I. INTRODUCTION	6
II. GROUNDS FOR STANDING.....	7
III. NOTE.....	7
IV. BACKGROUND	7
V. SUMMARY OF THE ’346 PATENT	7
VI. PROSECUTION HISTORY OF THE ’346 PATENT.....	11
VII. LEVEL OF ORDINARY SKILL IN THE ART.....	11
VIII. CLAIM CONSTRUCTION	12
IX. RELIEF REQUESTED AND THE REASONS FOR RELIEF	15
X. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE....	15
A. Challenged Claims	15
B. Statutory Grounds for Challenges	15
C. Ground 1: Claims 1-3, 5-11, 13-16, and 18 are obvious in view of Jeddeloh, Kleiman, and Ma.....	16
1. Summary of Jeddeloh.	16
2. Summary of Kleiman.....	20
3. Reasons to Combine Jeddeloh and Kleiman.	22
4. Summary of Ma.	25
5. Reasons to Combine Ma with Jeddeloh and Kleiman.....	25

6.	Claim 1	28
7.	Claim 2	49
8.	Claim 3	50
9.	Claim 5	50
10.	Claim 6	53
11.	Claim 7	60
12.	Claim 8	60
13.	Claim 9	61
14.	Claim 10	63
15.	Claim 11	64
16.	Claim 13	67
17.	Claim 14	67
18.	Claim 15	69
19.	Claim 16	70
20.	Claim 18	70
D.	Ground 2: Claims 4 and 17 are obvious in view of Jeddeloh, Kleiman, Ma, and Yashiro.	70
1.	Summary of Yashiro	70
2.	Reasons to Combine Yashiro with Jeddeloh, Kleiman, and Ma	70
3.	Claim 4	72
4.	Claim 17	75

XI.	DISCRETIONARY DENIAL IS NOT WARRANTED.....	76
A.	The Fintiv factors favor institution.	76
1.	No evidence regarding a stay	76
2.	Parallel proceeding trial date	77
3.	Investment in the parallel proceeding.....	77
4.	Overlapping issues with the parallel proceeding.....	77
5.	Petitioner is a defendant.....	78
6.	Other circumstances.....	78
B.	Advanced Bionics Test Favors Institution.	78
XII.	CONCLUSION.....	78
XIII.	MANDATORY NOTICES	80
A.	Real Party-in-Interest	80
B.	Related Matters.....	80
C.	Lead and Back-up Counsel and Service Information	80
	CERTIFICATE OF WORD COUNT.....	82
	CERTIFICATE OF SERVICE	83

PETITIONER’S EXHIBIT LIST

Ex.1001	U.S. Patent No. 8,601,346
Ex.1002	Prosecution History of U.S. 8,601,346
Ex.1003	Declaration of Paul Min under 37 C.F.R. § 1.68
Ex.1004	Curriculum Vitae of Paul Min
Ex.1005	U.S. Patent Publication No. 2011/0078496A1 to Jeddeloh (“Jeddeloh”)
Ex.1006	U.S. Patent No. 5,950,225 to Kleiman (“Kleiman”)
Ex.1007	U.S. Patent No. 8,341,332 to Ma et al. (“Ma”)
Ex.1008	U.S. Patent No. 5,787,460 to Yashiro et al. (“Yashiro”)
Ex.1009	U.S. Patent No. 6,161,165 to Solomon et al. (“Solomon”)
Ex.1010	Reserved
Ex.1011	The Authoritative Dictionary of IEEE Standards Terms: Seventh Edition; (“IEEE”)
Ex.1012	U.S. Patent No. 8,074,149 to Stolowitz (“Stolowitz”)
Ex.1013	U.S. Patent No. 8,819,699 to Cota-Robles (“Cota-Robles”)
Ex.1014	U.S. Patent No. 6,092,215 to Hodges et al. (“Hodges”)
Ex.1015	Complaint, <i>Polaris PowerLED Technologies, LLC v. Western Digital Corporation et al.</i> 2:24-cv-02864 (C.D. CA)
Ex.1016	District Court Trial Statistics
Ex.1017	Director Vidal Memorandum re: Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation, June 22, 2022

I. INTRODUCTION

The prior art presented in this Petition describes the same parity calculation techniques that are claimed in U.S. Patent No. 8,601,346 (“the ’346 patent”). The ’346 patent relates to a “method for generating parity data” that includes “distributing commands for performing a data stripe operation among a number of command processing units.” Ex.1001, 2:65-3:1. The ’346 patent was allowed after amending the claims to cover the concept in which “the parity calculator generates the parity block **on the fly** as the command processing units and the parity calculator are receiving the data blocks for the data stripe operation.” Ex.1001, 4:7-11; Ex.1002, 146.

However, it was already known to calculate parity data in this manner. The Jeddeloh reference describes using “RAID XOR circuitry [to] calculate parity data” and distribute commands among a plurality of “channel memory control circuit[s] 338-1, ...338-N.” Ex.1005, [0027], [0038]. And the Kleiman reference (entitled “Fly-by XOR for Generating Parity for Data Gleaned from a Bus) shows that it was known to use RAID XOR circuitry “for performing **fly-by** XOR.” Ex.1006, 3:38. The Ma reference describes further details recited in the claims.

As shown in this Petition, and supported by the declaration of Dr. Paul Min, Jeddeloh, Kleiman, and Ma render obvious claims 1-11 and 13-18 (hereinafter, the “Challenged Claims”) of the ’346 patent.

Accordingly, pursuant to 35 U.S.C. §§ 311, 314(a), and 37 C.F.R. § 42.100, Petitioner respectfully requests that the Board review and cancel as unpatentable under (pre-AIA) 35 U.S.C. § 103(a) the Challenged Claims of the '346 patent.

II. GROUNDS FOR STANDING

Petitioner certifies that the '346 patent is eligible for IPR and that Petitioner is not barred or estopped from requesting IPR challenging the patent claims. 37 C.F.R. § 42.104(a).

III. NOTE

Petitioner cites to exhibits' original page numbers. Emphasis in quoted material has been added.

IV. BACKGROUND

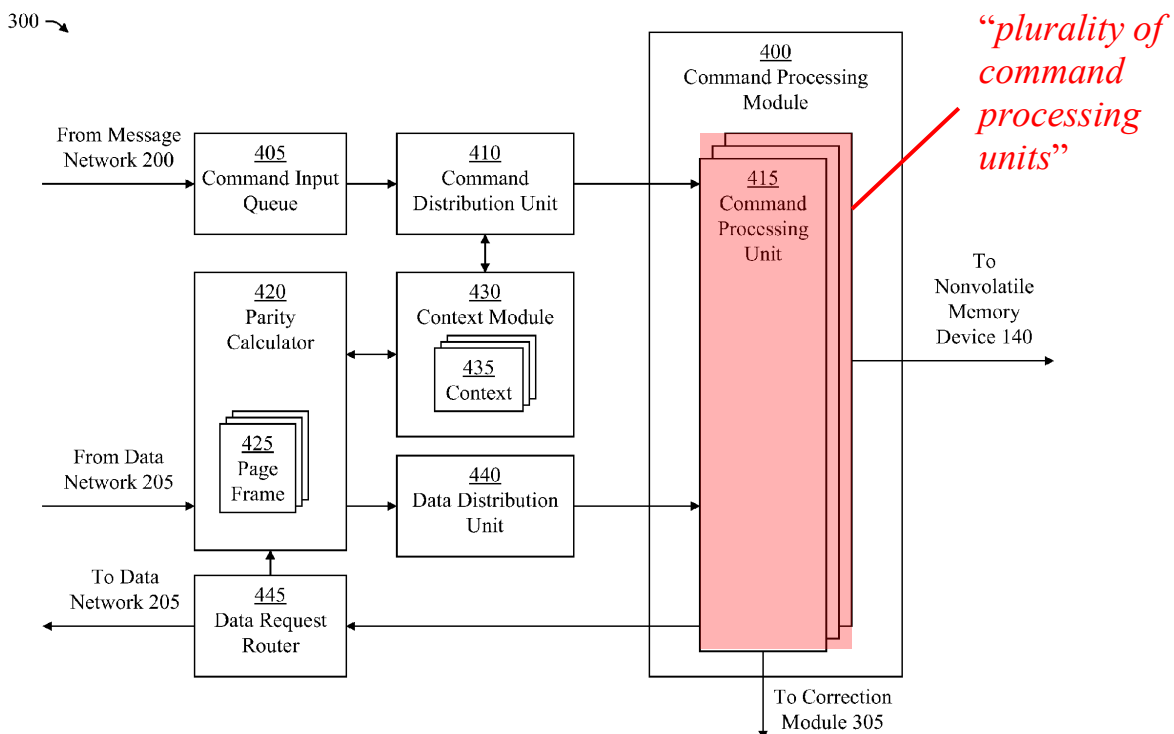
Dr. Paul Min's declaration provides a brief background section describing memory systems, Redundant Array of Independent Disks (RAID), and parity data. Ex.1003, ¶¶29-34.

V. SUMMARY OF THE '346 PATENT

The '346 patent describes and claims nothing more than well-known techniques to calculate and store parity data for a data stripe operation. The '346 relates to a "nonvolatile memory controller performs a data stripe operation on data blocks." Ex.1001, abstract. "The nonvolatile memory controller includes command processing units, each of which processes a command of the data stripe operation

to store a data block into a nonvolatile memory device.” Ex.1001, abstract. The non-volatile memory controller also includes a “parity calculator [that] generates a parity block.” Ex.1001, abstract.

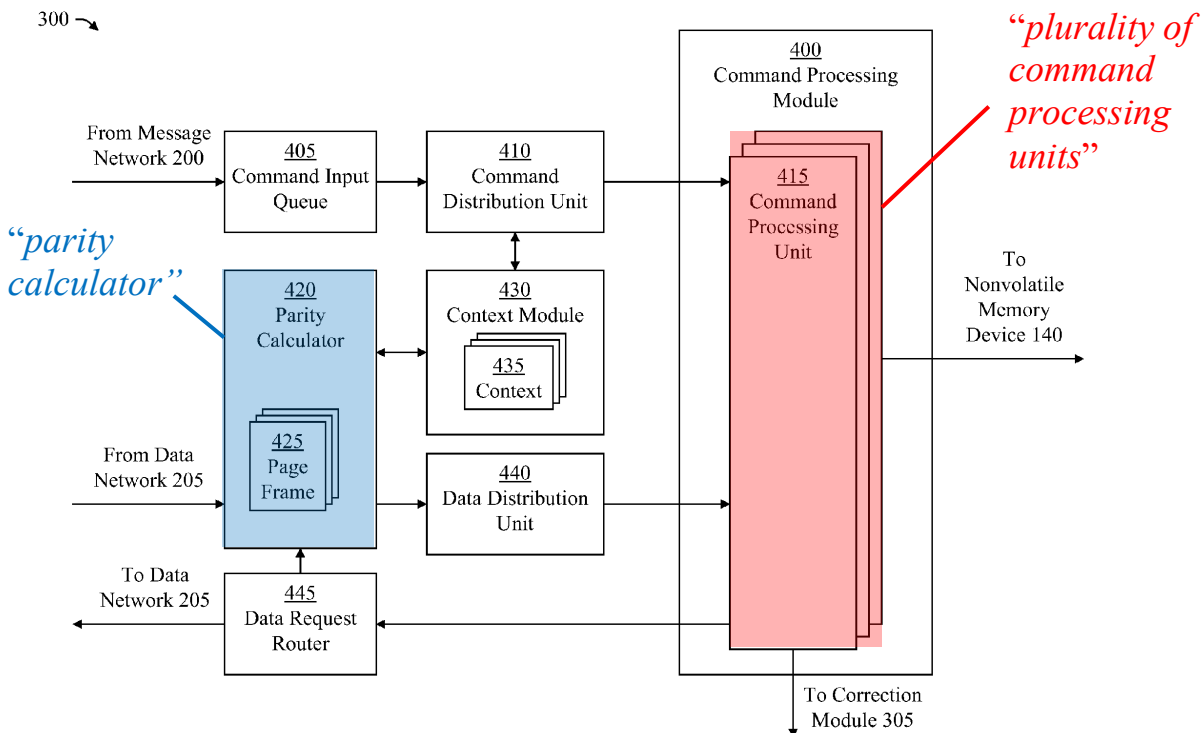
One purported novelty of the '346 patent is that commands are distributed across a plurality of command processing units. To process these commands, the claimed “nonvolatile memory controller includes a number of command processing units, each of which receives a command in the collection of commands.” Ex.1001, 1:53-55. The plurality of command processing units are shown in Fig. 4 below.



Ex.1001, Fig. 4 (annotated); Ex.1003, ¶36.

The '346 patent also purports to address the complexity of calculating parity data. The '346 patent acknowledges that “generation of parity data in a RAID operation is often a computationally intensive task.” Ex.1001, 1:33-35. In purported conventional systems, each of the “data blocks of a data stripe operation are stored in individual data buffers when computing a parity block.” Ex.1001, 2:16-17. These “data buffers consume a considerable amount of area and power in integrated circuit implementations of the disk drive controller.” Ex.1001, 1:44-46.

Accordingly, rather than storing each of the data blocks in a buffer at the same time, the “parity calculator generates the parity block **on the fly** as the command processing units are receiving the data blocks for the data stripe operation.” Ex.1001, 2:1-3. To do this, the “parity calculator generates a parity block by storing a first data block of the sequence of data blocks into a page frame and updating the data block stored in the page frame with each remaining data block in the sequence of data blocks.” Ex.1001, 1:63-67. The parity calculator is shown in Fig. 4 below.



Ex.1001, Fig. 4 (annotated); Ex.1003, ¶38.

The claims of the '346 patent were allowed after the Applicant made amendments directed to performing parity calculations on the fly: “by storing a first data block of the sequence of data blocks into the page frame and updating the data block stored in the page frame with each data block following the first data block in the sequence of data blocks, without storing each data block in a data buffer.” Ex.1002, 146.

However, as will be explained in detail below, calculating parity data “on-the-fly,” as well as distributing the commands among multiple command processors, were already well-known concepts at the time the '346 patent was filed. Ex.1003, ¶¶35-40.

VI. PROSECUTION HISTORY OF THE '346 PATENT

The '346 patent was filed on March 21, 2011 and claims priority to a provisional application filed August 16, 2010. In a first Office Action, the claims were rejected by the Examiner and found obvious over U.S. Patent Publication No. 2009/0077302 to Fukuda et al. ("Fukuda") and U.S. Patent Publication No. 2010/0185808 to Yu ("Yu").

To overcome the rejection, the Applicant added amendments directed to performing parity calculation on-the-fly. The claims were amended to recite that the parity block is generated "by storing a first data block of the sequence of data blocks into the page frame and updating the data block stored in the page frame with each data block following the first data block in the sequence of data blocks, without storing each data block in a data buffer." Ex.1002, 146. The Examiner then allowed the claims and the '346 patent issued on December 3, 2013.

However, as will be explained in detail below, calculating parity data "on the fly" was already well-known concepts at the time the '346 patent was filed. Ex.1003, ¶¶41-43.

VII. LEVEL OF ORDINARY SKILL IN THE ART

A Person of Ordinary Skill in The Art ("POSITA") in August of 2010 would have had a working knowledge of the memory system art that is pertinent to the '346 patent, including memory systems and memory controllers. A POSITA would

have had a bachelor's degree in electrical engineering computer science, or an equivalent, and approximately two years of professional experience working in the field of memory systems. Lack of professional experience can be remedied by additional education, and vice versa. Ex.1003, ¶¶22-24.

VIII. CLAIM CONSTRUCTION

Claim terms in IPR are construed according to their “ordinary and customary meaning” to those of skill in the art. 37 C.F.R. § 42.100(b). Petitioner submits that, for the purposes of this proceeding and the Grounds presented herein, the following phrase should be construed.

“without storing each data block in a data buffer.”

Based on the structure of the claims as well as the specification and the prosecution history, the limitation “*without storing each data block in a data buffer*” should be construed as “*without storing each data block in a data buffer at the same time.*” In other words, the claims do **not** require that each data block never pass through any buffer during the parity calculation process (as that would not be possible). Rather, the claims merely require that there is no single point in time in which each of the data blocks involved in the parity calculation are stored in a buffer together. That is, as each data block passes through the buffer, the parity calculation is updated using that block.

The claim language itself suggests this interpretation because the claims

recite that the first block is stored in a page frame, which is a type of buffer. Claim 1 recites “to generate a parity block by **storing a first data block** of the sequence of data blocks **into the page frame** and updating the data block stored in the page frame with each data block following the first data block in the sequence of data blocks, without storing each data block in a data buffer.” The page frame itself is a temporary storage for the first block and is thus a type of buffer. It would thus contradict the claim language to interpret “without storing each data block in a data buffer” to mean that a data block can never pass through a buffer during parity calculation. Accordingly, a POSITA would have understood that this term to mean that the parity block is generated *without storing each data block in a data buffer at the same time*.

The specification of the '346 patent supports this interpretation as well. The specification of the '346 patent states that conventionally, a “dedicated processor stores data blocks into individual data buffers of the disk drive controller, executes computing instructions to generate parity data, and stores the parity data into yet another data buffer.” Ex.1001, 1:37-40. The '346 patent further states that “data buffers consume a considerable amount of area and power in integrated circuit implementations of the disk drive controller.” Ex.1001, 1:44-46.

The '346 patent purports to address the data buffer issue by using an on-the-fly technique:

The parity calculator generates a parity block by storing a first data block of the sequence of data blocks into a page frame and updating the data block stored in the page frame with each remaining data block in the sequence of data blocks. In this way, **the parity calculator generates the parity block on the fly as the command processing units are receiving the data blocks for the data stripe operation.**

Ex.1001, 1:61-2:3.

Moreover, because the parity calculator generates the parity block in the page frame as the parity calculator is receiving the data blocks for the data stripe operation, the parity calculator need not store all the data blocks of the data stripe at the same time when generating a parity block.

Ex.1001, 2:9-13.

As a result of using this on-the-fly technique, the '346 patent purports that “the nonvolatile memory controller consumes less power and area than other nonvolatile memory controllers in which data blocks of a data stripe operation are stored in individual data buffers when computing a parity block.” Ex.1001, 2:13-17. Accordingly, a POSITA would have understood that what the '346 patent seeks to capture, is that the parity block is generated on-the-fly rather than by storing each of the data blocks in an individual buffer at the same time to perform the parity calculation.

For this reason as well, a POSITA would have understood this term to mean

that the parity block is generated *without storing each data block in a data buffer at the same time*. Ex.1003, ¶¶44-50.

IX. RELIEF REQUESTED AND THE REASONS FOR RELIEF

Petitioner asks that the Board institute a trial for *inter partes* review and cancel the Challenged Claims in view of the analysis below.

X. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE

A. Challenged Claims

Petitioner challenges claims 1-11 and 13-18.

B. Statutory Grounds for Challenges

Grounds	Claims	Basis
#1	1-3, 5-11, 13-16, and 18	Obvious under 35 U.S.C. § 103 (Pre-AIA) over Jeddeloh, Kleiman, and Ma
#2	4 and 17	Obvious under 35 U.S.C. § 103 (Pre-AIA) over Jeddeloh, Kleiman, Ma, and Yashiro

U.S. Patent Publication No. 2011/0078496 to Jeddeloh (“Jeddeloh”) was filed September 29, 2009 and published on March 31, 2011. Jeddeloh is thus prior art to the ’346 patent under pre-AIA 35 U.S.C. 102(e).

U.S. Patent No. 5,950,225 to Kleiman (“Kleiman”) was filed February 28, 1997 and issued Sept. 7, 1999. Kleiman is thus prior art to the ’346 patent under at least pre-AIA 35 U.S.C. 102(b).

U.S. Patent No. 8,341,332 to Ma et al. (“Ma”) was filed August 5, 2008 and

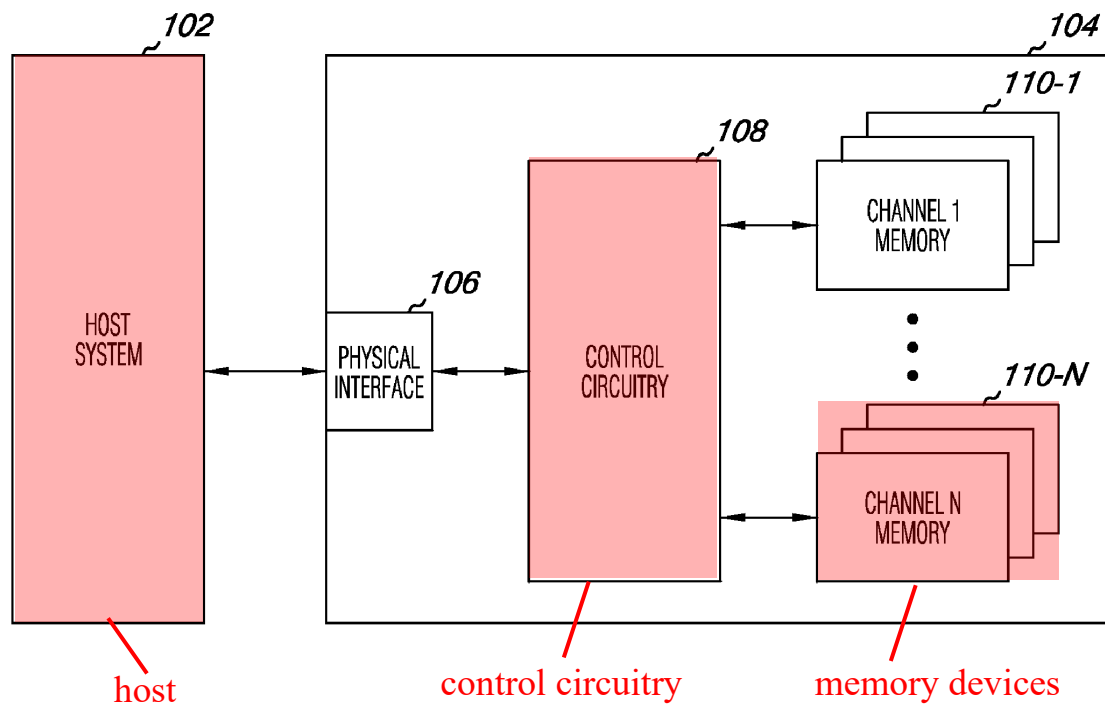
issued on December 25, 2012. Ma is thus prior art to the '346 patent under pre-AIA 35 U.S.C. 102(e).

U.S. Patent No. 5,787,460 to Yashiro et al. ("Yashiro") was filed Nov. 6, 1996 and issued on July 28, 1998. Yashiro is thus prior art to the '346 patent under at least pre-AIA 35 U.S.C. 102(b).

C. Ground 1: Claims 1-3, 5-11, 13-16, and 18 are obvious in view of Jeddeloh, Kleiman, and Ma.

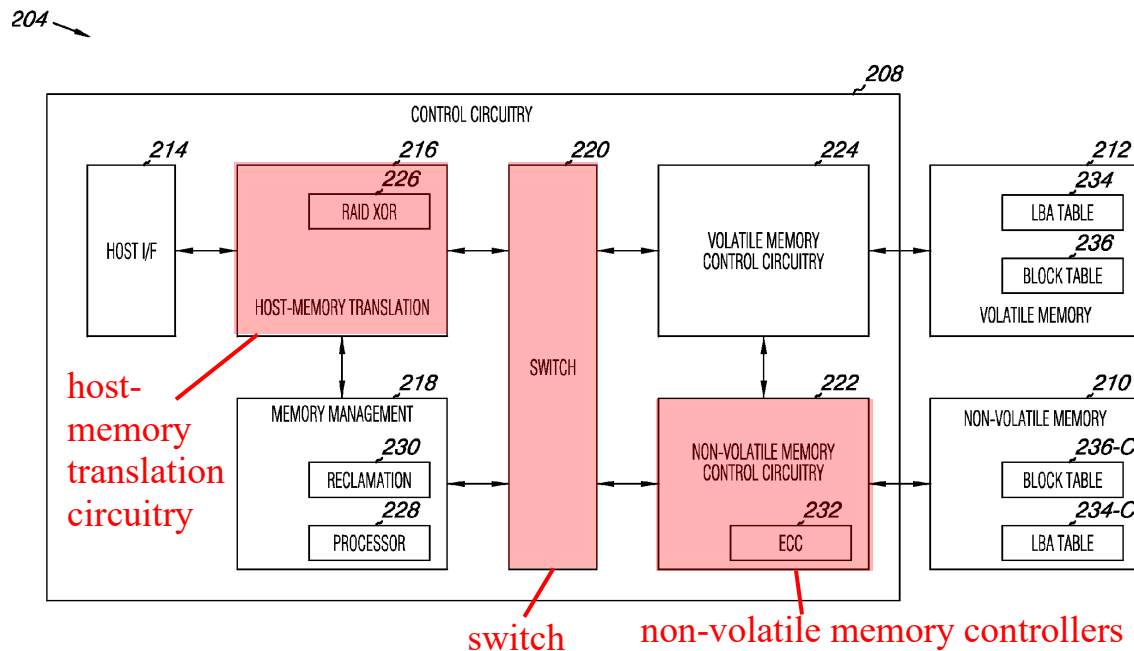
1. Summary of Jeddeloh.

Jeddeloh, like the '346 patent, "relates generally to semiconductor memory devices." Ex.1005, [0001]. Consistent with its title, Jeddeloh describes a "stripe-based memory operation." Ex.1005, [0001]. To perform stripe-based memory operations, Jeddeloh's system includes memory control circuitry 108 for managing memory operations between a host 102 and the multiple memory devices 110 of the stripe operation. Ex.1005, [0015].



Ex.1005, Fig. 1 (annotated); Ex.1003, ¶54.

Jeddeloh illustrates the control circuitry in more detail with respect to Fig. 2. “The memory system control circuitry 208 can include host interface circuitry 214, host-memory translation circuitry 216, memory management circuitry 218, a switch 220, non-volatile memory control circuitry 222, and/or volatile memory control circuitry 224.” Ex.1005, [0024].



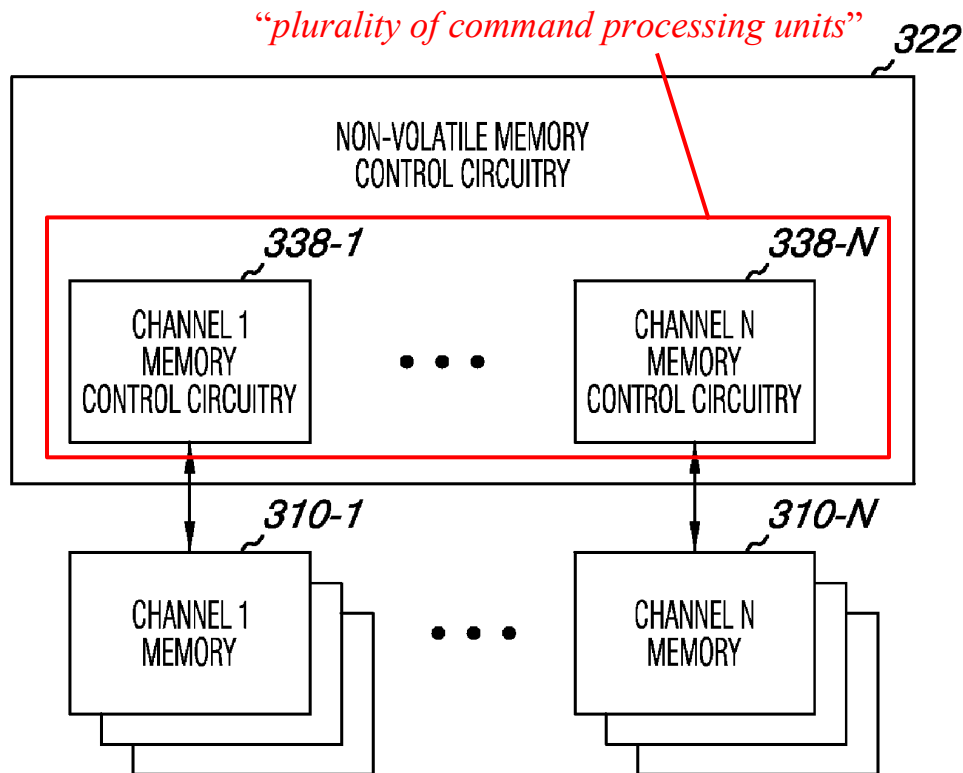
Ex.1005, Fig. 2 (annotated); Ex.1003, ¶55.

As shown in Fig. 2, Jeddelloh’s “switch 220 can be coupled to the host-memory translation circuitry 216, the memory management circuitry 218, [and] the non-volatile control circuitry 222.” Ex.1005, [0033]. Jeddelloh further explains that “the switch 220 can be a direct memory access (DMA) module.” Ex.1005, [0033].

Like the ’346 patent, Jeddelloh’s memory control circuitry 208 includes a parity calculator. In particular, “[t]he host-memory translation circuitry 216 can include second level error detection/correction circuitry, such as RAID exclusive or (XOR) circuitry 226. The RAID XOR circuitry 226 can calculate parity data based on data received from the host interface circuitry 214.” Ex.1005, [0027]; *see also* Ex.1005, [0044].

Also like the ’346 patent, Jeddelloh utilizes a plurality of command

processing units to perform the stripe operation. Jeddeloh's non-volatile memory control circuitry 222 includes a plurality of discrete channel memory control circuits 338-1 ... 338-N: "In one or more embodiments, each solid state memory device 310-1, . . . , 310-N can be coupled to the non-volatile memory control circuitry 322 [222 in Fig. 2] by a separate channel. Each channel can be associated with discrete channel memory control circuitry 338-1, . . . , 338-N." Ex.1005, [0038]. The channel memory control circuitry performs "a number of operations on the one or more solid state memory devices 310-1, . . . , 310-N including page read, stripe write, and stripe erase." Ex.1005, [0038]. The discrete channel memory controllers are shown in further detail with respect to Fig. 3, shown below.



Ex.1005, Fig. 3 (annotated); Ex.1003, ¶58.

Thus, Jeddelloh provides evidence that it was known to distribute the processing of commands for a data stripe operation among multiple command processing units, as well as to calculate parity data. Ex.1003, ¶¶54-59.

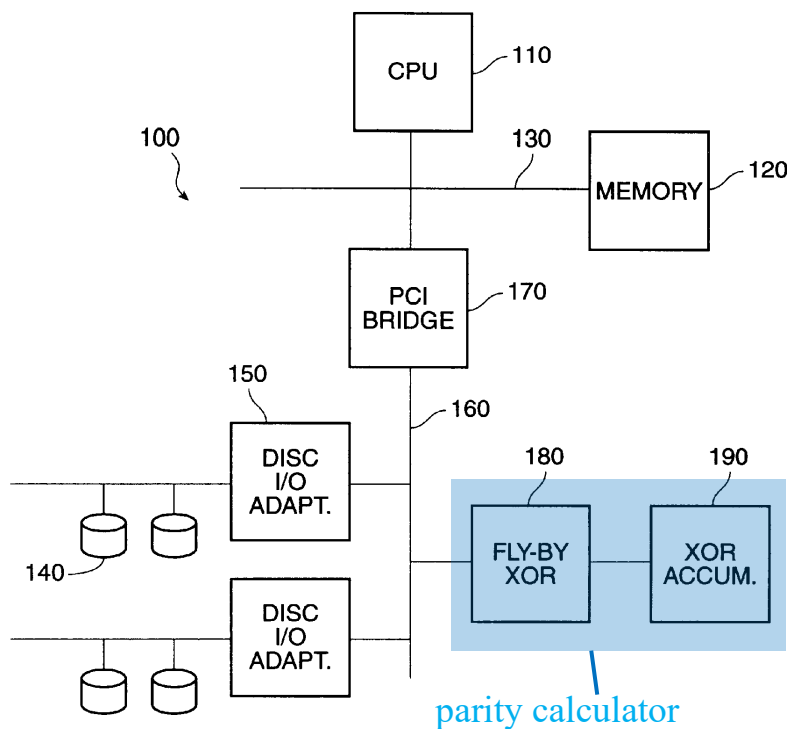
2. Summary of Kleiman.

Kleiman provides evidence that it was known to perform parity calculations “on-the-fly.” Ex.1006, abstract. Like Jeddelloh and the ’346 patent, Kleiman relates to a “method and system for performing XOR operations without consuming substantial computing resources.” Ex.1006, abstract. Kleiman, which is entitled “Fly-by XOR for Generating Parity for Data Gleaned from a Bus,” describes the

same on-the-fly parity calculation technique described by the '346 patent. *See* Ex.1006, abstract.

Kleiman describes a “fly-by XOR element 180” that “uses an XOR buffer 190 for saving partial and final results of XOR operations.” Ex.1006, 3:3-5.

Kleiman’s fly-by XOR element 180 is shown in Fig. 1 below.



Ex.1006, Fig. 1 (annotated); Ex.1003, ¶62.

Kleiman’s fly-by XOR element detects data transfers to the storage disks. “When the fly-by XOR element 180 recognizes such a transfer, it copies the data being transferred and adds it (by performing an XOR operation) to the accumulated parity information in the associated XOR accumulator buffer.”

Ex.1006 3:60-64. “When all data for the recognized memory buffer has been transferred, the associated XOR accumulator buffer holds the computed parity for the RAID stripe.” Ex.1006, 3:65-67; *see also* Ex.1006, 4:1-6. Thus, Kleiman provides evidence that it was known to utilize an on-the-fly technique for parity calculation in which the data blocks are not stored in a buffer at the same time. Ex.1003, ¶¶60-64.

3. Reasons to Combine Jeddeloh and Kleiman.

A POSITA would have found it obvious for Jeddeloh’s parity calculator to use an on-the-fly technique as taught by Kleiman. Ex.1003, ¶65. Jeddeloh describes a parity calculator generally: “RAID XOR circuitry 226 can calculate parity data based on data received from the host interface circuitry 214.” Ex.1005, [0027]. A POSITA would have looked to known techniques to implement the RAID XOR circuitry. Ex.1003, ¶65.

Kleiman provides an example of one known way to implement RAID circuitry. Kleiman implements a parity calculation technique that calculates parity data on-the-fly: “method of operating a system for performing fly-by XOR.” Ex.1006 3:37-38. In particular, “the fly-by XOR element 180 ... copies the data being transferred and adds it (by performing an XOR operation) to the accumulated parity information in the associated XOR accumulator buffer.” Ex.1006 3:60-65. Then, “[w]hen all data for the recognized memory buffer has

been transferred, the associated XOR accumulator buffer holds the computed parity for the RAID stripe.” Ex.1006, 3:65-67; Ex.1003, ¶66.

A POSITA would have been motivated to utilize Kleiman’s on-the-fly technique because it provides various benefits.

- **First**, Kleiman’s on-the-fly technique “performs XOR operations on data transferred to and from the disk drives without requiring separate transfers.” Ex.1006, 1:67-2:2.
- **Second**, the on-the-fly technique “dynamically computes a set of parity information for the RAID stripe, without requiring additional resources from any part of the system 100.” Ex.1006 3:17-20; Ex.1003, ¶67.

Accordingly, the motivation to combine is explicitly suggested. *KSR Int’l v. Teleflex Inc.*, 550 U.S. 398, 417 (2007).

The results of the proposed combination would have been predictable because Kleiman’s on-the-fly parity calculation is done for the same reason parity data is calculated in Jeddeloh—creating parity data for a RAID stripe operation. Jeddeloh explains that “write data can be striped across multiple solid state memory devices in a RAID operation.” Ex.1005, [0028]. Then, “[t]he memory system, e.g., RAID XOR circuitry 226 ... can calculate parity data for the write data.” Ex.1005, [0044]; Ex.1003, ¶68.

Kleiman likewise performs parity calculation for a RAID scheme. Kleiman explains that “When a RAID stripe is read or written, either entirely or partially, the fly-by XOR element 180 thus dynamically computes a set of parity information for the RAID stripe.” Ex.1006, 3:14-17. Accordingly, a POSITA would have reasonably predicted Kleiman’s on-the-fly technique to be successful within Jeddeloh’s RAID scheme. Ex.1003, ¶69.

A POSITA would have had a reasonable expectation of success because Kleiman’s on-the-fly technique was a well-established method for calculating parity data. *See e.g.*, Ex.1009, abstract (“A high performance data path for performing XOR on the fly.”); Ex.1012, 10:34-37 (“Segments of these data words, e.g. see data paths 342, 344, are written to each of the drives. At this point, a logical XOR operation can be performed between the corresponding bits of the segments ‘on the fly.’”). Ex.1003, ¶70. Accordingly, the combination is nothing more than “the predictable use of prior art elements according to their established functions.” *KSR* 550 U.S. at 417.

Accordingly, the combination of Jeddeloh and Kleiman is merely applying a known technique (Kleiman’s on-the-fly parity calculation) to a known device (Jeddeloh’s RAID XOR circuitry) ready for improvement to yield predictable results (performing parity calculations on-the-fly). *See KSR*, 550 U.S. at 417. Ex.1003, ¶¶65-71.

4. Summary of Ma.

Ma provides additional details as to the processing of write commands. Dr. Min's declaration provides a summary of how Ma provides additional details related to DMA operations of a switch connected to multiple memory controllers. Ex.1003, ¶¶72-75.

5. Reasons to Combine Ma with Jeddeloh and Kleiman.

A POSITA would have found it obvious for Jeddeloh's switch 220—which may be a DMA module—to use known DMA techniques as taught by Ma to communicate with the channel memory controllers 338 and the host. Ex.1003, ¶76.

As explained above, Jeddeloh's memory control circuitry 208 includes a “switch 220 [that] can be coupled to the host-memory translation circuitry 216, the memory management circuitry 218, [and] the non-volatile control circuitry 22.” Ex.1005, [0033]. Jeddeloh further explains that “the switch 220 can be a direct memory access (DMA) module.” Ex.1005, [0033]; Ex.1003, ¶77.

Jeddeloh's switch passes data between the host and the channel memory controllers 338 of the non-volatile memory circuitry 222. “The switch 220 can be coupled to the host-memory translation circuitry 216 [and] the non-volatile control circuitry 222.” Ex.1005, [0033]. Accordingly, Jeddeloh's channel memory controller receives write data through the switch 220. Jeddeloh's channel memory controllers 338 are configured to queue write requests. “A page write command

can be sent, **e.g., queued, to the desired channel memory controller 338-1, . . . , 338-N** and executed on the one or more solid state memory devices 310-1, . . . , 310-N.” Ex.1005, [0040]. Jeddeloh then explains that the data may be obtained from the host. In one example, “Write data for a stripe write operation **can be received from a host system**, e.g., host system 102 in FIG. 1.” Ex.1005, [0040]; Ex.1003, ¶78.

Given that Jeddeloh’s switch may be a DMA module, a POSITA would have found it obvious to look to known techniques for DMA operations between a switch and a plurality of memory controllers. Ex.1003, ¶79. In other words, by referring to DMA techniques, Jeddeloh explicitly suggests the combination. *See In re Kahn*, 441 F.3d 977, 986 (Fed. Cir. 2006) (obviousness may be established by combining or modifying prior art elements when there is a teaching, suggestion, or motivation to do so). Ma provides an example of such known techniques. In particular, Ma provides evidence that it was known for a memory controller to request data from the host when processing write commands in a queue. Like both Jeddeloh and the ’346 patent, Ma “relates to flash-memory solid-state-drive (SSD) devices.” Ex.1007, 1:52-53.

Ma explains that when one of the memory controllers on a flash device is ready to write data, it “sends a DMA activate back to host 10 through smart storage switch 30 to indicate that it is ready for the DMA transfer to begin.” Ex.1007,

10:61-63. In response to the DMA activate request, the “host 10 sends one or more data-out packets to smart storage switch 30, which takes the data and forms one or more data-out packets to send to endpoint-1, with the data to write into flash memory blocks in first single-chip flash-memory device 73.” Ex.1007, 11:45-49. “NVM controller 76 in first single-chip flash-memory device 73 receives these data packets and then writes this data into its flash memory blocks 68 and responds with a status packet back to smart storage switch 30.” Ex.1007, 11:51-54; Ex.1003, ¶80.

A POSITA would have thus found it obvious to implement Jeddelloh’s system such that the channel memory controller sends a DMA request to the host (through the switch) when processing a command from its queue. Ex.1003, ¶81. Jeddelloh’s switch would then send back the requested data as a data-out packet so that the channel memory controller can write the data within that packet to its channel memory. Ex.1003, ¶81. Indeed, the combination is nothing more than “the predictable use of prior art elements according to their established functions.” *KSR* 550 U.S. at 417.

A POSITA would have had a reasonable expectation of success when combining the teachings of Ma with those of Jeddelloh. Both Jeddelloh and Ma relate to solid-state storage devices that stripe data across multiple storage devices. *See e.g.*, Ex.1005, [0036] (“The non-volatile memory control circuitry 222 can be

configured to operate a number of non-volatile memory devices associated with the number of memory channels as a RAID.”); Ex.1007, 2:56-58 (“It is further desired that the smart storage switch interleaves and stripes data accesses to the multiple channels of single-chip flash-memory devices.”). Accordingly, a POSITA would have expected Ma’s DMA techniques to work similarly within Jeddeloh’s system. Ex.1003, ¶82.

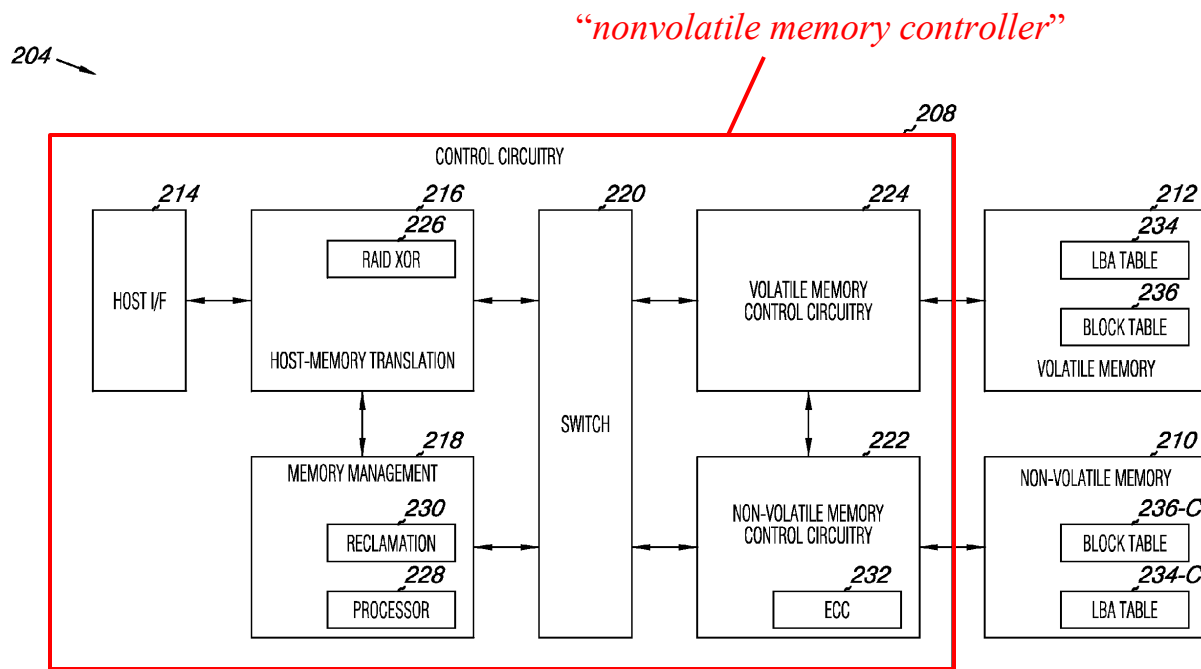
Accordingly, the combination of Jeddeloh and Ma is merely applying a known technique (Ma’s DMA technique) to a known device (Jeddeloh’s memory control circuitry) ready for improvement to yield predictable results (obtaining write data from the host when ready). Ex.1003, ¶¶76-83.

6. Claim 1

[1.0] *A nonvolatile memory controller for performing a data stripe operation on a plurality of data blocks, the nonvolatile memory controller comprising:*

To the extent the preamble is limiting, it is rendered obvious by Jeddeloh. Ex.1003, ¶84.

First, Jeddeloh describes memory system control circuitry 208 (“*nonvolatile memory controller*”) for controlling non-volatile memory. “The memory system 204 can include memory system control circuitry 208. The memory system control circuitry 208 can be coupled to one or more solid state memory devices, e.g., non-volatile memory 210.” Ex.1005, [0023].



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶85.

Second, Jeddeloh’s memory system control circuitry performs a stripe-based memory operation (“*data stripe operation*”). See Ex.1005, title (“Stripe Based Memory Operation”). “The present disclosure includes methods and devices for **stripe-based memory operation**. One method embodiment includes writing data in **a first stripe across a storage volume of a plurality of memory devices.**” Ex.1005, [0012].

Third, Jeddeloh’s stripe-based memory operation is performed on a plurality of pages (“*a plurality of data blocks*”). “A stripe write operation can include a plurality of page write operations, e.g., one page write operation per each channel associated with the non-volatile memory control circuitry 322.” “For

example, a page write operation can support 4 kilobyte (KB), 8 KB, or another volume of data, depending upon solid state memory device 310-1, . . . , 310-N page size, plus meta-data.” Ex.1005, [0040].

By performing the stripe operation on a plurality of memory pages, Jeddeloh teaches performing a stripe operation on “*a plurality of data blocks*” under the various ways a POSITA would have understood the term “block.” For example, a memory page is a structured form of data and is thus a “block” as claimed. Ex.1003, ¶88.

Additionally, Jeddeloh’s memory pages include logical blocks (the smallest unit of addressable memory). “In one or more embodiments of the present disclosure, and as shown in FIG. 5, a page can store data in a number of sectors.” Ex.1005, [0062]. “As one of ordinary skill in the art will appreciate, **logical block addressing** is a scheme that can be used by a host for identifying a sector of information, e.g., each sector can correspond to a unique LBA. In one or more embodiments, a sector is the smallest addressable portion of a storage volume.” Ex.1005, [0062]. Accordingly, because the stripe operation is performed on memory pages, which are comprised of sector/LBAs (logical *block* addresses), Jeddeloh teaches performing a stripe operation on “*a plurality of data blocks*.”

As another example, Jeddeloh explains that its memory pages can be organized into a larger structure referred to as a block. “The memory arrays can

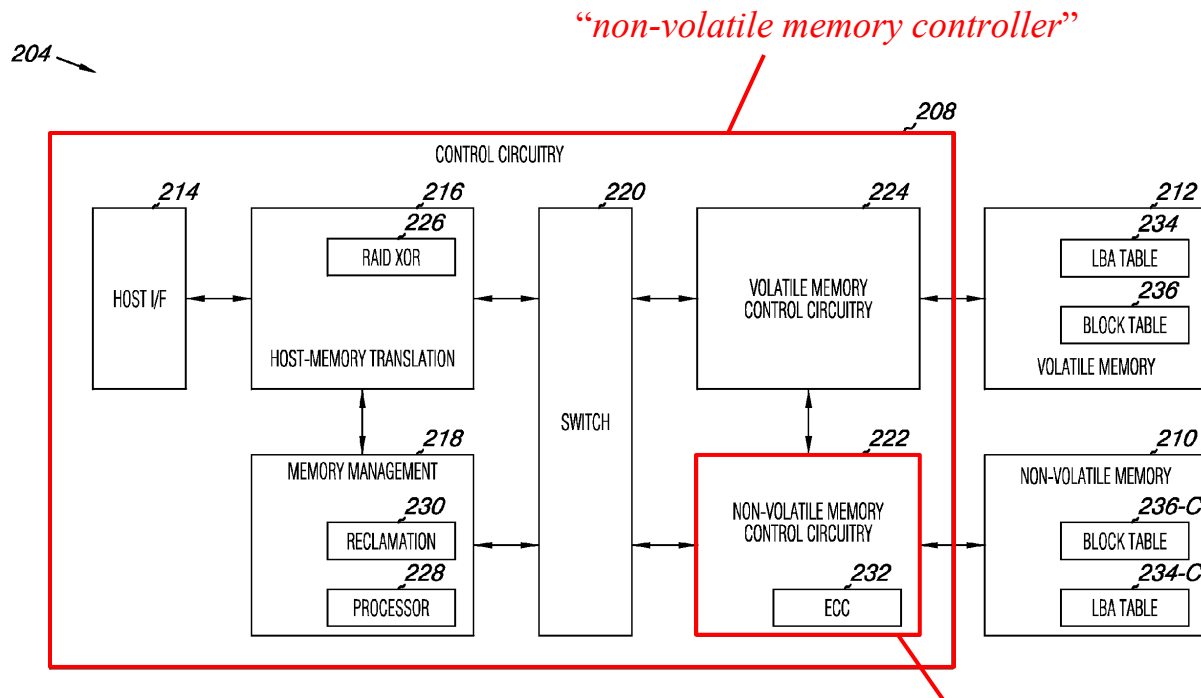
include a number of memory cells organized into a number of physical pages, and the physical pages can be organized into a number of blocks.” Ex.1005, [0005].

Accordingly, when Jeddeloh’s system performs a stripe operation on memory pages, which can be organized into a number of blocks, the stripe operation is performed on “*a plurality of data blocks*” regardless of how the term block is interpreted. Ex.1003, ¶90.

Thus, because Jeddeloh describes memory control circuitry for performing a data stripe operation on a plurality of memory pages, which are comprised of LBAs and together form larger blocks, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶84-92.

[1.1] *a plurality of command processing units, each command processing unit of the plurality of command processing units configured to receive a command of a plurality of commands for performing the data stripe operation,*

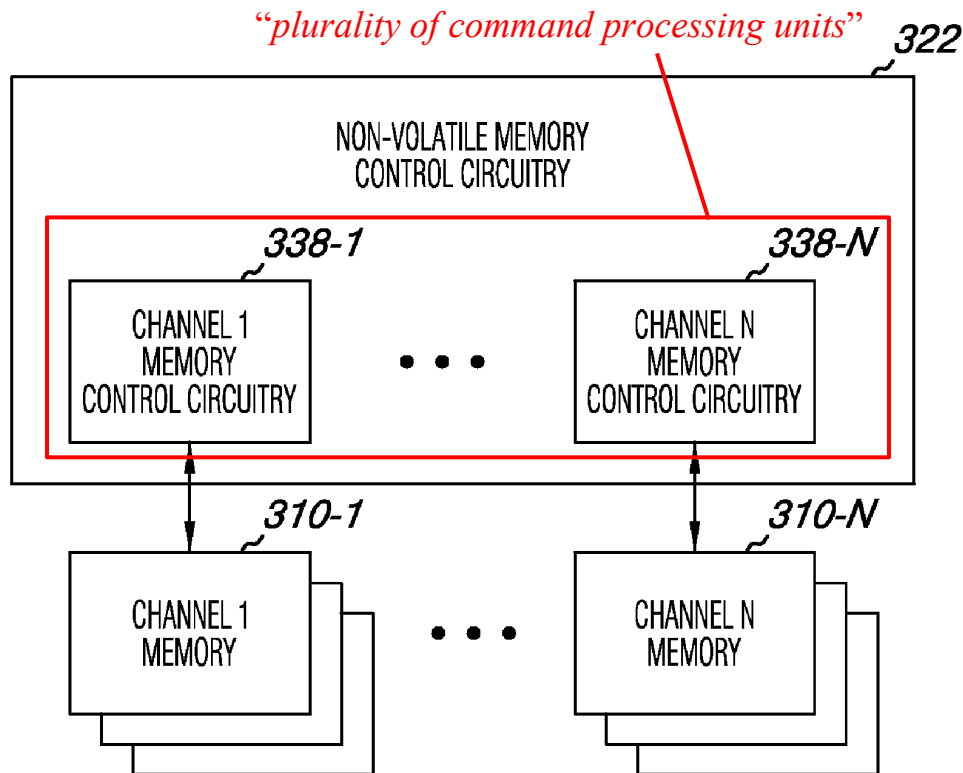
First, Jeddeloh’s control circuitry 208 includes non-volatile memory control circuitry 222, which includes a plurality of channel memory control circuits 338 (“*plurality of command processing units*”). “The memory system control circuitry 208 can include ... non-volatile memory control circuitry 222.” Ex.1005, [0024]. “The non-volatile memory control circuitry 222 can be configured to write data in a first stripe across a number of memory channels as described herein.” Ex.1005, [0036].



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶93.

non-volatile memory
control circuitry

Jeddeloh’s non-volatile memory control circuitry 222 includes a plurality of discrete channel memory control circuits 338-1 ... 338-N (*“plurality of command processing units”*). “In one or more embodiments, each solid state memory device 310-1, . . . , 310-N can be coupled to the non-volatile memory control circuitry 322 by a separate channel. Each channel can be associated with discrete channel memory control circuitry 338-1, . . . , 338-N.” Ex.1005, [0038]; *see also* Ex.1005, [0038].



Ex.1005, Fig. 3 (annotated); Ex.1003, ¶94.

Jeddeloh’s channel memory control circuits 338 are “*command processing units*” because they process various operations “including page read, stripe write, and stripe erase.” Ex.1005, [0038]. “A stripe write operation can include a plurality of page write operations, e.g., one page write operation per each channel associated with the non-volatile memory control circuitry 322.” Ex.1005, [0040].

Second, in a data stripe operation that includes a plurality of commands, the different channel memory control circuits receive different commands (“*each command processing unit of the plurality of command processing units configured to receive a command of a plurality of commands for performing the data stripe*

operation”). For example, during a stripe operation across each of the channel memory devices 310-1 ... 310-N, each of the corresponding channel control circuits will process its respective command for that stripe operation. “A stripe write operation can include a plurality of page write operations, e.g., one page write operation per each channel associated with the non-volatile memory control circuitry 322. In one or more embodiments, the plurality of page write operations in the stripe write operation can be performed substantially simultaneously across the plurality of channels.” Ex.1005, [0040]. “Write data can be striped across a storage volume of a plurality of solid state memory devices 310-1, . . . , 310-N.” Ex.1005, [0041].

Thus, because Jeddeloh’s memory control circuitry includes a plurality of channel memory circuits to process commands, and each of those channel memory circuits receives a command that is part of a data stripe operation, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶93-97.

[1.2] the plurality of commands including a plurality of data update commands and a parity write command,

First, Jeddeloh explains that the commands provided to the channel memory circuits (“*command processing units*”) include “*data update commands.*” “A memory system can receive ... **a write command and updated write data associated with a particular one or more of the LBAs associated with the data**

written in the first stripe 444-1, e.g., an LBA that is the same as at least one LBA associated with data written in the first stripe 444-1.” Ex.1005, [0050];

see also [0051]. Accordingly, because some write commands are used to write “updated write data associated with a particular one or more of the LBAs,” they are “*data update commands*” as claimed. Ex.1005, [0050]; Ex.1003, ¶98.

Second, the commands provided to the channel memory circuits also include parity write commands. “Storing parity data can allow data associated with an effectively removed portion of a memory stripe to be recreated, e.g., a portion that experiences a UECC error.” Ex.1005, [0047]. “In some embodiments that include **writing parity data** along with the write data, at least one fragment of write data can be written to one or more memory devices associated with a first subset of the channels ... and parity data can be written to one or more memory devices associated with a second subset of the channels.” Ex.1005, [0049]; *see also* Ex.1005, [0028] (“write data can be striped across seven of eight memory devices, **where parity data can be stored in an eighth memory device**”), [0044] (“a memory system can include eight channels where seven channels are used for writing write data and **one channel is used for writing parity data**”).

Jeddeloh explains that one memory channel and its associated memory device may be dedicated to parity data. “At least one of the number of memory channels can be operated as a parity channel and accordingly, one or more memory

devices associated with the at least one of the number of memory channels can be operated as a parity memory device(s).” Ex.1005, [0036]. Accordingly, a write command processed by the channel memory controller 338 for the dedicated parity device is a “*parity write command*” as claimed.

Additionally, a POSITA would have found it obvious to use parity write commands when writing parity data. It was known in the art to use a parity write command when writing parity data to memory. *See e.g.*, Ex.1014, 3:3-5 (“The current proposed ANSI standard for “XOR commands” on SCSI disk drives includes read (READ), data update (XDWRITE), parity update (XPWRITE)”); Ex.1003, ¶103. Accordingly, it would have been obvious to use a parity update command, such as XPWRITE, when writing parity data to a dedicated parity channel.

Thus, because Jeddeloh describes that the commands processed by the memory channel circuits include data update commands and parity write commands, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶98-104.

[1.3a] *each command processing unit of the plurality of command processing units receiving a data update command of the plurality of data update commands configured to request a data block of the plurality of data blocks based on the data update command,*

A POSITA would have found it obvious for Jeddeloh’s channel memory circuits 328 to request and receive pages for writing to memory according to

known DMA techniques, as taught by Ma.

First, as explained above at [1.2], channel memory control circuits receive data update commands (“*each command processing unit of the plurality of command processing units receiving a data update command of the plurality of data update commands*”).

Second, when implementing the DMA techniques mentioned by Jeddeloh, the channel memory circuits (“*command processing units*”) are configured to “*request a data block*” as claimed. Jeddeloh explains that write commands are queued to the channel memory controllers, and that the channel memory controllers then write data received from the host system. “A page write command can be sent, e.g., queued, to the desired channel memory controller 338-1, . . . , 338-N and executed on the one or more solid state memory devices 310-1, . . . , 310-N.” Ex.1005, [0040]. “Write data for a stripe write operation can be received from a host system, e.g., host system 102 in FIG. 1.” Ex.1005, [0040].

As explained above at X.C.5, it would have been obvious for Jeddeloh’s memory control circuitry 208 to utilize DMA operations. *See e.g.*, Jeddeloh, [0033]; Ex.1003, ¶107. A POSITA would have understood that when a particular channel memory control circuit processes the next command in the queue, it would request that data when ready, as taught by Ma: “flash-memory device 73 ... sends a DMA activate back to host 10 to indicate that it is ready for the DMA transfer to

begin.” Ex.1007, 11:40-44.

Accordingly, by sending a DMA activate message to the host, the DMA host is requesting the write data from the host (“*to request a data block of the plurality of data blocks based on the data update command*”). Accordingly, a POSITA would have found it obvious for Jeddeloh’s channel memory controllers 338 to similarly request the write data associated with the data update command (“*based on the data update command*”) from the host using a DMA activate message. *See supra* X.C.5; Ex.1003, ¶108.

Thus, because Jeddeloh’s channel memory circuits queue commands that when processed, request data from the host, and Ma provides an example of using DMA operations to do so, Jeddeloh and Ma render this limitation obvious.

Ex.1003, ¶¶105-09.

[1.3b] *[each command processing unit ... configured to] receive the data block in response to the request, and write the data block to a nonvolatile memory device; and*

First, Jeddeloh explains that the channel memory controllers 338 write the data blocks to non-volatile memory. *See* Ex.1005, [0040]-[0041]. As explained above at [1.3a], Jeddeloh’s channel memory controllers 338—implementing Ma’s DMA technique—request a data block when ready. Similarly, Jeddeloh’s channel memory controllers—implementing Ma’s DMA technique—receive the requested data block (as a data-out packet) in response to the request. “[H]ost 10 sends one or

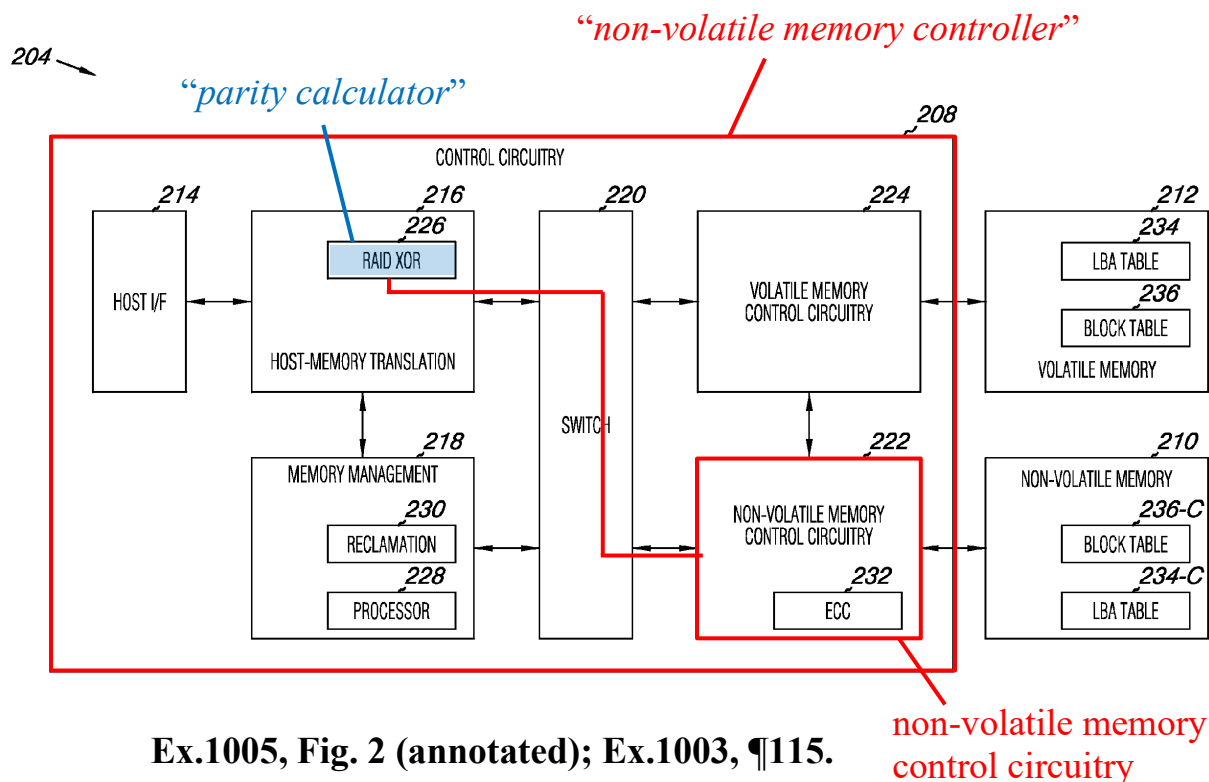
more data-out packets to smart storage switch 30, which takes the data and forms one or more data-out packets to send to endpoint-1, with the data to write into flash memory blocks in first single-chip flash-memory device 73.” Ex.1007, 11:45-50. Accordingly, Jeddelloh’s channel memory controllers—implementing Ma’s DMA technique—would similarly receive from the DMA switch the data-out packets that include the data to write to memory (“*receive the data block in response to the request*”).

Second, Jeddelloh’s channel memory controllers write the data to their respective memory devices “*in response*” to receiving that data from the DMA request. Jeddelloh explains that data that is written is “received from [the] host.” Ex.1005, [0041]. A POSITA would have understood that the data cannot be written to the memory devices until it has been received. As explained by Ma: “NVM controller 76 in first single-chip flash-memory device 73 receives these data packets and then writes this data into its flash memory blocks 68.” Ex.1007, 11:51-55.

Thus, because Jeddelloh’s channel memory controllers write to their respective memory devices, and it would have been obvious to do so in response to receiving the write data according to a DMA operation described in further detail by Ma, Jeddelloh and Ma render this limitation obvious. Ex.1003, ¶¶110-14.

[1.4] a parity calculator coupled to the plurality of command processing units,

First, Jeddeloh’s memory control circuit includes RAID XOR circuitry 226 (“a *parity calculator*”). “The host-memory translation circuitry 216 can include second level error detection/correction circuitry, such as RAID exclusive or (XOR) circuitry 226. The RAID XOR circuitry 226 can calculate parity data based on data received from the host interface circuitry 214.” Ex.1005, [0027]; *see also* Ex.1005, [0044]. Jeddeloh’s RAID XOR circuitry 226 is shown in Fig. 2 below.



Second, as shown in Fig. 2 above, the RAID XOR circuitry (“*parity calculator*”) is coupled to the channel memory control circuits 338 (“*plurality of command processing units*”) of the non-volatile memory control circuitry 222. *See*

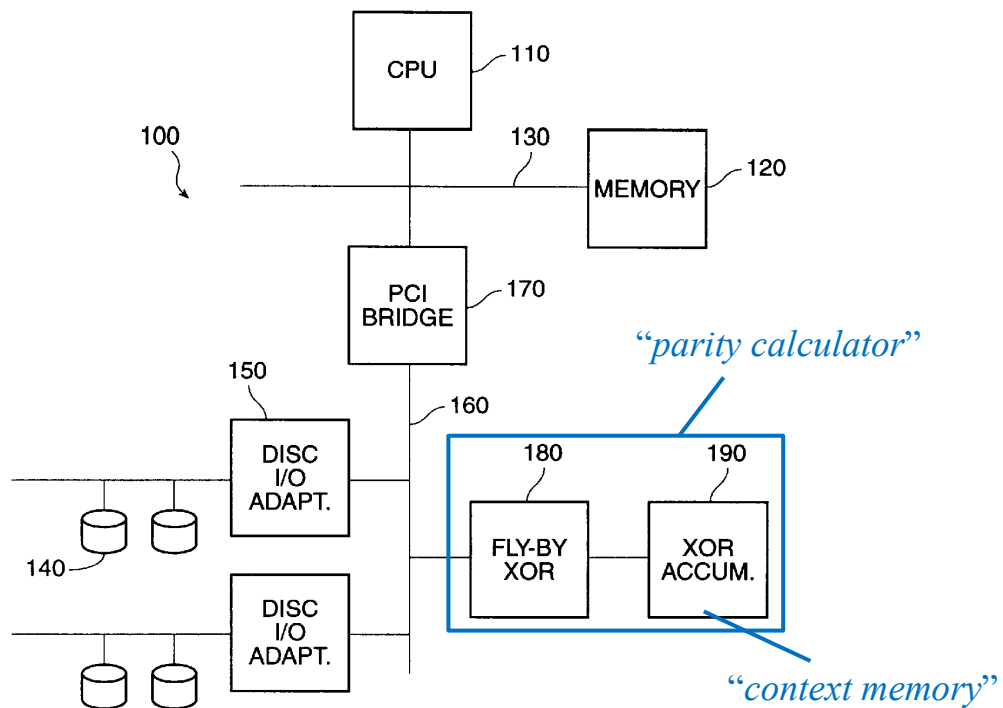
supra [1.1]. Indeed, because the non-volatile memory circuitry (including the channel memory controllers (“*plurality of command processing units*”) receives parity data, they are communicatively coupled to the RAID XOR circuitry that calculates the parity data. *See* Ex.1005, [0028], [0044].

Thus, because Jeddeloh’s memory control circuit 208 includes RAID XOR circuitry for calculating parity data, and the RAID XOR circuitry is connected to the channel memory circuitry 338 of the non-volatile memory control circuitry 222, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶115-17.

[1.5] *the parity calculator further comprising a context memory including a page frame,*

For the reasons explained above at X.C.2, it would have been obvious for Jeddeloh’s RAID XOR circuitry 226 to be implemented according to Kleiman’s on-the-fly technique. Ex.1003, ¶118.

First, Jeddeloh’s parity calculator—implemented according to Kleiman’s techniques—includes a “*context memory*” as claimed. Kleiman’s on-the-fly technique utilizes a XOR buffer 190 (“*context memory*”) to store the partial parity result until the parity block is complete.



Ex.1006, Fig. 1 (annotated); Ex.1003, ¶119.

In Kleiman, a “fly-by XOR element 180 is coupled to the mezzanine bus 160, so as to glean data from data transfers which use the mezzanine bus 160. The fly-by XOR element 180 uses an XOR buffer 190 for saving partial and final results of XOR operations. Ex.1006, 3:1-5. “As the data is transferred using the mezzanine bus 160, the fly-by XOR element 180 reads the data (without interference with the data transfer) and adds it to one of the XOR accumulators (see FIG. 2) in the XOR buffer 190.” Ex.1006, 3:6-22.

Kleiman’s XOR buffer 190 is a “context” memory as claimed because it maintains the context of the parity calculation, similar to the context memory

described in the '346 patent. *See* Ex.1001, 14:24-30. The context of the operation is maintained by storing the partial results of the parity calculation until all data blocks have passed through. “At a step 222, the ... processor 110 also allocates one of the XOR accumulator buffers in the XOR buffer 190 to accumulate the results of the XOR operation.” Ex.1006, 3:49-56. The XOR element 180 “copies the data being transferred and adds it (by performing an XOR operation) to the accumulated parity information in the associated XOR accumulator buffer.” Ex.1006, 3:57-67. Accordingly, because the XOR buffer stores the updated parity block throughout the stripe operation, it is a “*context memory*” as claimed.

Second, Kleiman’s XOR buffer 190 “*context memory*” includes at least one XOR accumulator buffer (“*page frame*”). Kleiman explains that “[t]he XOR buffer 190 includes a random access memory (RAM) 340, which provides a set of 32 XOR accumulator buffers, each of which is 1024 entries long by 4 bytes (32 bits) wide, thus 4K bytes in size.” Ex.1006, 5:33-36.

Kleiman’s XOR accumulator buffers, as they would have been implemented within Jeddeloh’s system would have been sized to store a memory page. Jeddeloh performs its operations on pages by writing different pages to different memory devices. *See* Ex.1005, [0040]. Accordingly, Jeddeloh’s RAID circuit, implemented according to Kleiman’s technique, would include a XOR accumulator buffer sized for a page. A POSITA would have recognized that a XOR accumulator buffer

sized for a page is a “*page frame*” as claimed. *See e.g.*, Ex.1011, 789 (“page frame ... A block of main storage having the size of, and used to hold, a page ... An area of main storage used to hold a page.”) Accordingly, Jeddeloh’s system as implemented with Kleiman’s on-the-fly technique would use a XOR accumulator with enough memory space to hold a page and would thus include a “*page frame*” as claimed.

Thus, because Jeddeloh describes RAID XOR circuitry, and a POSITA would have found it obvious for the RAID XOR circuitry to include a XOR buffer with XOR accumulators, Jeddeloh and Kleiman render this limitation obvious.

¶¶118-25

[1.6] *the parity calculator configured to receive the plurality of data blocks as a sequence of data blocks,*

First, as explained above at [1.4], Jeddeloh describes that “RAID XOR circuitry 226 can calculate parity data based on data received from the host interface circuitry 214.” Ex.1005, [0027].

Second, when using Kleiman’s on-the-fly technique, the data being received from the host interface circuitry is received in sequence. “The timing and control element 320 sequences each word in the FIFO queue 350 ... through the XOR operator 354, so as to perform an XOR operation on data in the associated XOR accumulator buffer in the RAM 340 with data in the FIFO queue 350.” Ex.1006,

7:26-32. “The XOR buffer 190 also includes a FIFO queue 350, which provides for inputting sequential words of bus data values and performing the XOR operation on those **sequential words**.” Ex.1006, 5:39-44. Accordingly, it would have been obvious for Jeddeloh’s RAID XOR circuitry, as it implements Kleiman’s on-the-fly technique, to receive pages (“*plurality of data blocks*”) in sequence.

Thus, because Kleiman’s fly-by XOR element receives data in sequence, Jeddeloh and Kleiman render this limitation obvious. Ex.1003, ¶¶126-29.

[1.7] [the parity calculator configured] to generate a parity block by storing a first data block of the sequence of data blocks into the page frame and updating the data block stored in the page frame with each data block following the first data block in the sequence of data blocks,

First, to generate a parity block, Kleiman’s on-the-fly technique “*stor[es] a first data block of the sequence of data blocks into the page frame.*” Kleiman’s fly-by XOR element “copies the data being transferred and adds it (by performing an XOR operation) to the accumulated parity information in the **associated XOR accumulator buffer**.” Ex.1006, 3:62-64.

Kleiman also explains that before the parity block operation, the XOR accumulator is “reset to a uniform zero value.” Ex.1006, 10:47-48. Accordingly, because a bit value XORed with zero is the original bit value, the first data block stored in the XOR accumulator will simply be that first data block. Accordingly, Kleiman’s technique “*stor[es] a first data block of the sequence of data blocks into*

the page frame.” Using this technique in Jeddeloh, a first page (*“first data block”*) is stored in the XOR accumulator (*“page frame”*) of the XOR buffer.

Second, for each subsequent piece of data passing through, Kleiman’s fly-by XOR element replaces (*“updat[es]”*) the data in the XOR accumulator with an XOR of the subsequent data and the data in the XOR buffer (*“updating the data block stored in the page frame with each data block following the first data block in the sequence of data blocks”*). “As the data is transferred using the mezzanine bus 160, the fly-by XOR element 180 reads the data (without interference with the data transfer) and adds it to one of the XOR accumulators (see FIG. 2) in the XOR buffer 190.” Ex.1006, 3:7-21. “When the fly-by XOR element 180 recognizes such a transfer, it copies the data being transferred and adds it (by performing an XOR operation) to the accumulated parity information in the associated XOR accumulator buffer.” Ex.1006, 3:57-67.

This process, described by Kleiman, was a well-known process for on-the-fly parity calculation. For example, Solomon similarly describes this process. “In performing an XOR on data transfers, **the first data being passed between the host-side memory and the disk-side memory is copied into the XOR buffer 48.**” Ex.1009, 4:22-41. “In subsequent data transfers between the two memories, data words are XORed by the DMA/XOR engine with the corresponding data words in the XOR buffer 48. Each time the result of the XORing is replaced into

the XOR buffer 48.” Ex.1009, 4:22-41.

Using Kleiman’s technique in Jeddeloh, the RAID XOR circuitry 222 updates the data stored in a buffer with each subsequent page passing through. In particular, the RAID XOR circuitry XORs each passing page with the data stored in the XOR buffer and replaces the data in the XOR buffer with the result.

Ex.1003, ¶34.

Thus, because it would have been obvious for Jeddeloh’s RAID XOR circuitry to use Kleiman’s on-the-fly technique to calculate parity data, and that technique updates the XOR buffer with an XOR result from each subsequent page passing through, Jeddeloh and Kleiman render this limitation obvious. Ex.1003, ¶¶130-35.

[1.8] *without storing each data block in a data buffer,*

Jeddeloh’s RAID XOR circuitry, implemented with Kleiman’s on-the-fly technique, does not have to store each data block in a buffer at the same time in order to perform the XOR calculation. Instead, it performs the XOR calculation “on-the-fly” and thus does so “*without storing each data block in a data buffer.*”

Kleiman’s technique is consistent with the on-the-fly technique described in the ’346 patent. As explained above at VIII, a POSITA would have understood that the term “*without storing each data block in a data buffer*” would be interpreted as “*without storing each data block in a data buffer at the same time.*” “**In this way,**

the parity calculator generates the parity block on the fly as the command processing units are receiving the data blocks for the data stripe operation.”

Ex.1001, 1:61-2:3; Ex.1003, ¶137.

Thus, because it would have been obvious for Jeddeloh’s RAID XOR circuitry to utilize Kleiman’s on-the-fly technique, Jeddeloh and Kleiman render this limitation obvious. Ex.1003, ¶¶136-38.

[1.9] the command processing unit receiving the parity write command configured to write the parity block to a nonvolatile memory based on the parity write command.

Jeddeloh’s memory controller writes parity data based on the parity write command. “A page write command can be sent, e.g., queued, to the desired channel memory controller 338-1,..., 338-N and executed on the one or more solid state memory devices 310-1,..., 310-N.” Ex.1005, [0040]. “As an example, write data can be striped across seven of eight memory devices, where parity data can be stored in an eighth memory device, e.g., memory system 204 can include multiple non-volatile memory devices 210, each coupled to the memory system control circuitry 208 by a separate channel.” Ex.1005, [0028].

The parity data is written to a specific memory channel and thus processed by one of the channel memory controllers (“*command processing units*”). “At least one of the number of memory channels can be operated as a parity channel and accordingly, one or more memory devices associated with the at least one of the

number of memory channels can be operated as a parity memory device(s).”

Ex.1005, [0036]; *see also* Ex.1005, [0048].

Thus, because Jeddeloh describes using channel memory controllers to write parity data to memory, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶139-41.

7. Claim 2

[2.1] *The nonvolatile memory controller of claim 1, wherein the parity calculator is further configured to update the data block stored in the page frame by performing a logical operation on the data block stored in the page frame and a data block following the first data block in the sequence of data blocks.*

As explained above at [1.7], Jeddeloh’s RAID XOR circuitry, utilizing Kleiman’s on-the-fly technique, updates the parity block by performing an XOR operation on the data stored in the XOR buffer and a subsequent data block passing through. The XOR function is a “*logical operation*.” The “XOR operation” described by Kleiman is a Boolean logical operation that returns a “1” if the two bit values are different and a “0” if both bit values are the same. *See e.g.*, Ex.1011, 403.

Thus, because Kleiman’s on-the-fly technique uses a XOR operation, which is a Boolean logical operation, Jeddeloh and Kleiman render this limitation obvious. Ex.1003, ¶¶142-43.

8. Claim 3

[3.1] *The nonvolatile memory controller of claim 2, wherein the logical operation includes an exclusive-or operation.*

As explained above at [1.7] and [2.1], Kleiman’s on-the-fly technique utilizes an XOR (exclusive-OR) operation. Thus, Jeddeloh and Kleiman render this limitation obvious. Ex.1003, ¶144.

9. Claim 5

[5.1] *The nonvolatile memory controller of claim 1, wherein the nonvolatile memory controller further comprises a command distribution unit configured to distribute the plurality of data update commands to command processing units of the plurality of command processing units,*

First, Jeddeloh’s memory control circuitry 208 includes host memory translation circuitry 216 (“*command distribution unit*”) that distributes commands to the channel memory controllers 338 of the non-volatile memory control circuitry 222. “The host-memory translation circuitry 216 might, for example, convert host sector read and write operations to commands directed to specific portions of the non-volatile memory 210.” Ex.1005, [0027].

As explained above at [1.1], different portions of memory (i.e., different channels) are managed by different memory channel control circuits 338 (“*command processing units*”). Thus, by translating commands and directing them to specific portions of memory and thus the memory channel control circuits associated therewith, the host-memory translation circuitry “*distribute[s] the*

plurality of data update commands to command processing units of the plurality of command processing units.”

Additionally, the non-volatile memory control circuitry 222/322 distributes commands across the different memory channels. “The non-volatile memory control circuitry 222 can be configured to write data in a first stripe across a number of memory channels as described herein.” Ex.1005, [0036]. “At least one of the number of memory channels can be operated as a parity channel and accordingly, one or more memory devices associated with the at least one of the number of memory channels can be operated as a parity memory device(s).” Ex.1005, [0036].

Again, as explained above at [1.1], each of the memory channels is controlled by memory channel control circuits 338. Thus, by sending commands to the different memory channels and thus the memory channel control circuits associated therewith, the host translation circuitry 216 “*distribute[s] the plurality of data update commands to command processing units of the plurality of command processing units.”*

Thus, because the host-memory translation circuitry 216 distributes commands to different memory channel controllers, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶145-49.

[5.2] [the command distribution unit configured to] determine generation of the

parity block is complete, and distribute the parity write command to a command processing unit of the plurality of command processing units in response to determining the parity block is complete.

First, as explained above at [1.4], the host-memory translation circuit includes RAID XOR circuitry 226. The RAID XOR circuitry—implemented with Kleiman’s on-the-fly technique—determines that the parity block is complete. Kleiman explains that the system updates the XOR accumulator with each passing data block. Then, when the last of the data blocks has been XORed, the contents of the XOR buffer are written to memory as the parity data. “When all data for the recognized memory buffer has been transferred, the associated XOR accumulator buffer holds the computed parity for the RAID stripe.” Ex.1006, 3:65-67. Then, “the processor 110 directs the fly-by XOR element 180 to readout the accumulated parity information from the associated XOR accumulator buffer, either to a memory buffer in the memory 110 or to a storage block on one of the disk drives 140.” Ex.1006, 4:1-11.

Second, as explained above at [5.1], the host-memory translation circuitry 222 directs commands—including the parity write command—to the channel memory control circuits 338.

Third, as explained above at [1.9], the parity write command is sent to the appropriate channel memory after it is complete.

Thus, because Jeddeloh includes translation circuitry that distributes write

and parity commands to the channel memory controllers, and the RAID XOR circuitry implemented according to Kleiman's technique distributes the parity block once complete, Jeddeloh and Kleiman render this limitation obvious.

Ex.1003, ¶¶150-53.

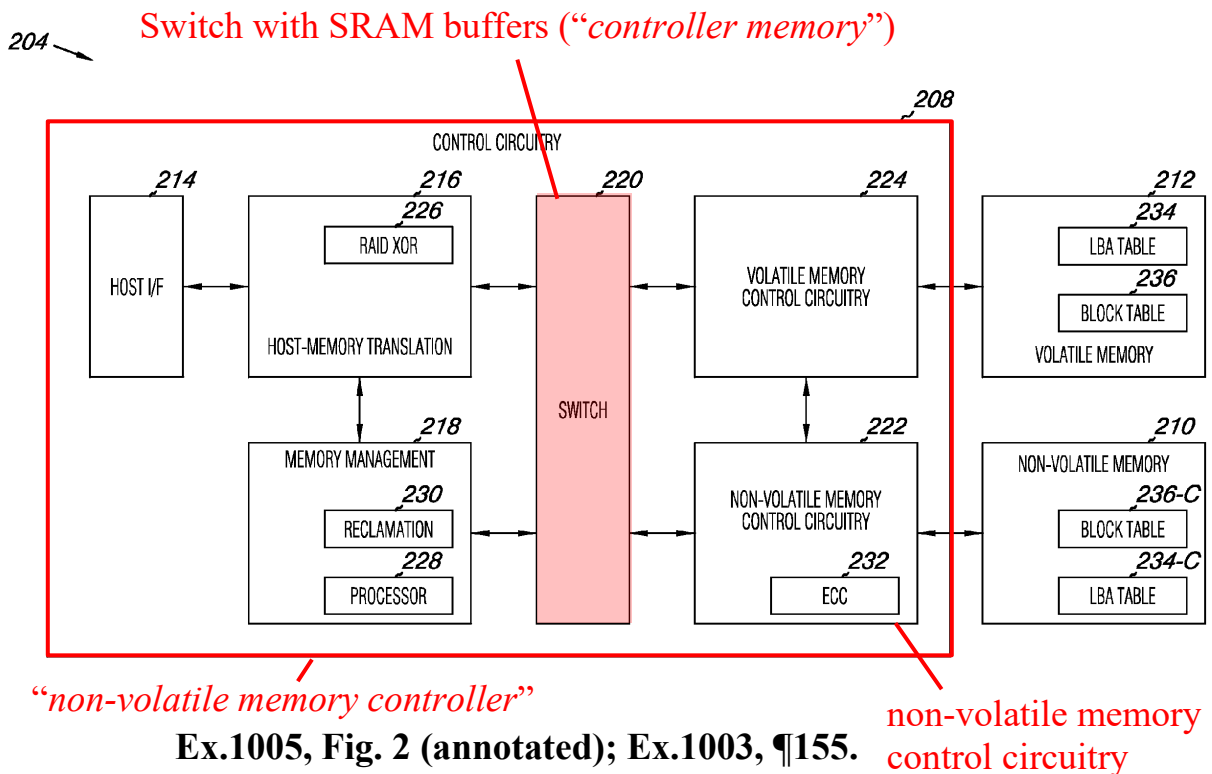
10. Claim 6

[6.0] *The nonvolatile memory controller of claim 1, further comprising.*

See Claim 1. Ex.1003, ¶154.

[6.1] *a controller memory configured to store the plurality of data blocks; and*

First, Jeddeloh's memory controller includes a switch 220 with SRAM buffers ("*controller memory*"). "The switch 220 can be a cross bar switch and can include and/or be coupled to one or more buffers, e.g., static random access memory (SRAM) buffers." Ex.1005, [0033].



Second, because the memory pages (*“data blocks”*) pass through the switch 220 on their way from the host to the memory devices, the switch is configured to store the memory pages in the switch’s SRAM buffers, for example. Indeed, Ma, which provides more details as to a DMA switch’s operation. Ma explains that data blocks being passed to the memory control circuitry are stored in the switch’s buffers: “Host 10 responds with the data-out packet with the **write data**, DATA-1, **which is stored in the SDRAM buffer of smart storage switch 30.**” Ex.1007, 12:44-49. Accordingly, it would have been obvious for Jeddeloh’s switch 220 to similarly buffer incoming write data (*“data blocks”*) in its SRAM buffers.

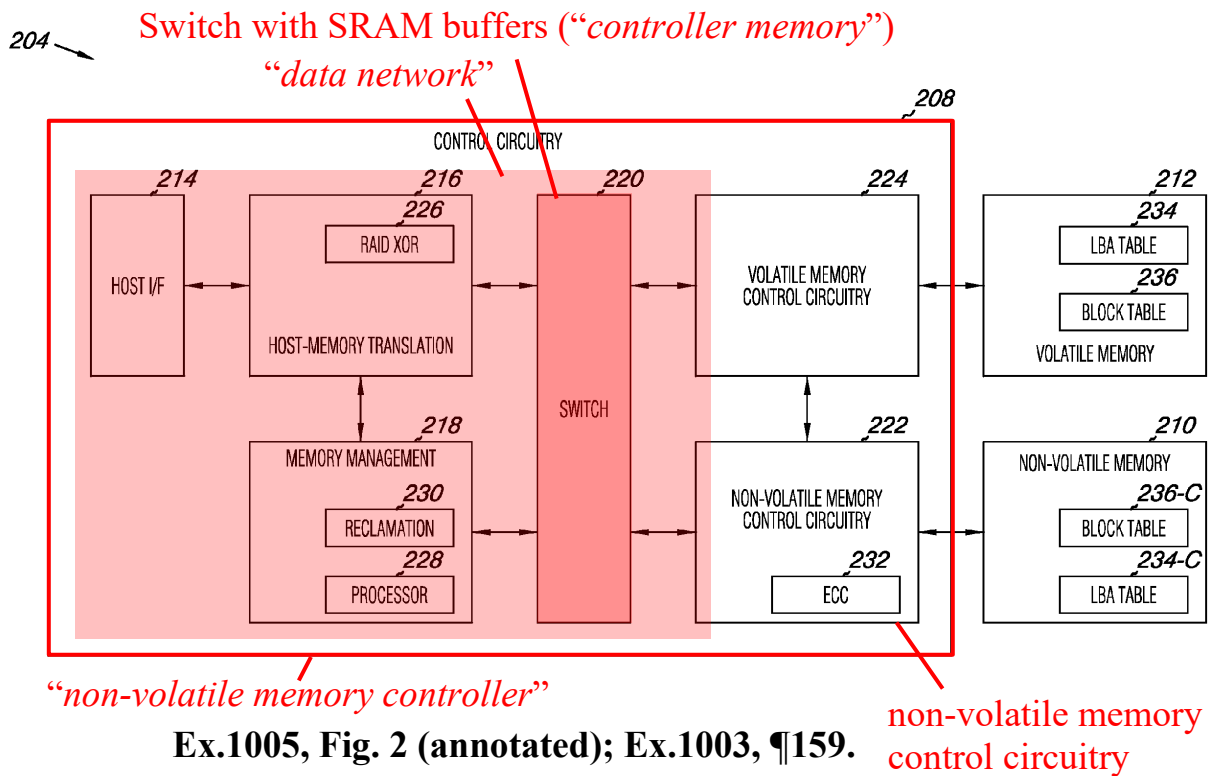
Thus, because page write operation data (*“the plurality of data blocks”*) pass

through the buffers of the switch 220 from the host translation circuitry to the non-volatile memory control circuitry 222, Jeddeloh renders this limitation obvious.

Ex.1003, ¶¶155-58.

[6.2] *a data network coupled to the controller memory, the plurality of command processing units, and the parity calculator,*

First, Jeddeloh’s memory controller includes a “*data network*” that connects the various components such as the host interface 214, host memory translation circuitry 216, switch 220, and non-volatile memory control circuitry 222, as illustrated in Fig. 2 below. “The switch 220 can be coupled to the host-memory translation circuitry 216, the memory management circuitry 218, the non-volatile control circuitry 222, and/or the volatile memory control circuitry 224.” Ex.1005, [0033].



Second, the network is connected to the switch 220 that includes SRAM buffers (“*controller memory*”). “The switch 220 can be a crossbar switch and can include and/or be coupled to one or more buffers, e.g., static random access memory (SRAM) buffers.” Ex.1005, [0033]. Accordingly, the “*data network [is] coupled to the controller memory.*”

Third, the network is connected to the channel memory control circuits 338 (“*the plurality of command processing units*”) of the non-volatile memory control circuitry (222). “The non-volatile memory control circuitry 222, e.g., controller, can be coupled to the switch 220 and to one or more non-volatile memory devices 210.” Ex.1005, [0034]. Accordingly, the “*data network [is] coupled to ... the*

plurality of command processing units” as claimed.

Fourth, the network is connected to the Raid XOR circuitry 226 (“*parity calculator*”) of the host-memory translation circuitry 216. “The switch 220 can be coupled to the host-memory translation circuitry 216.” Ex.1005, [0033].

According, the “*data network [is] coupled to the ... parity calculator*” as claimed.

Thus, because Jeddeloh’s memory controller includes a switch 220 with (or coupled to) SRAM buffers, host translation circuitry 222, and non-volatile memory circuitry 222 all interconnected together, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶159-63.

[6.3] wherein the plurality of command processing units is further configured to request the plurality of data blocks by generating a plurality of data request packets,

First, as described above at [1.3a] and [1.3b], each of the channel memory controllers 338 (“*plurality of command processing units*”) requests the data from the host using, for example, a “DMA activate” request as evidenced by Ma. Accordingly, because a plurality of channel memory controllers send such requests, there are a plurality of data requests.

Second, a POSITA would have understood that the requests are “*request packets*” as claimed. For example, Ma describes a DMA activate request as a “DMA activate packet” that requests the write data from the host: “Smart storage switch 30 generates **a DMA activate packet** back to host 10. Host 10 responds

with the data-out packet with the write data, DATA-1, which is stored in the SDRAM buffer of smart storage switch 30.” Ex.1007, 12:43-46.

Thus, because Jeddeloh’s channel memory circuits—implementing Ma’s DMA technique—send out DMA activate packets to request write data, Jeddeloh and Ma render this limitation obvious. Ex.1003, ¶¶164-66.

[6.4] *the controller memory is further configured to generate a plurality of completion packets including the plurality of data blocks, and*

First, as described above at [6.1], Jeddeloh’s memory controller includes a switch 220 with SRAM buffers (“*controller memory*”). Jeddeloh further explains that “the switch 220 can be a direct memory access (DMA) module.” Ex.1005, [0033].

Second, when performing DMA operations (described in more detail by Ma), the switch 220 (“*controller memory*”) would transmit the data to be written in a data-out packet (“*completion packets*”). “Next host 10 sends one or more data-out packets to smart storage switch 30, which takes the data and forms one or more **data-out packets** to send to endpoint-1 [e.g., Jeddeloh’s channel control circuits 338].” Ex.1007, 11:45-50. Ma’s data-out packets are “*completion packets*” as claimed. In particular, the data-out packets include the “data to write to flash memory blocks” (“*including the plurality of data blocks*”).

Thus, because Jeddeloh’s switch, when implementing Ma’s DMA technique,

generates data-out packets with the data to be written to memory, Jeddeloh and Ma render this limitation obvious. Ex.1003, ¶¶167-70.

[6.5] *both the parity calculator and the plurality of command processing units are further configured to receive the plurality of data blocks through the data network.*

First, as described above at [6.2], Jeddeloh's data network connects the switch 220 to the host-memory translation circuitry 216 (which includes the RAID XOR circuitry ("*parity calculator*")) and the channel memory controllers ("*plurality of command processing units*").

Second, the RAID XOR circuitry 226 ("*parity calculator*") receives the memory pages to be written to memory from the host interface 214 over the data network. "The host-memory translation circuitry 216 can be coupled to the host interface circuitry 214, to the memory management circuitry 218, and/or to the switch 220." Ex.1005, [0027]. "The host-memory translation circuitry 216 can include ... RAID exclusive or (XOR) circuitry 226. The RAID XOR circuitry 226 can calculate parity data based on data received from the host interface circuitry 214." Ex.1005, [0027]. Accordingly, because the RAID XOR circuitry receives data to be written to memory from the host interface, the RAID XOR circuitry is "*configured to receive the plurality of data blocks through the data network.*"

Second, as explained above at [1.3b] the channel memory controllers 338 receive the write data from the host. As explained at [6.1] and [6.2], the channel

memory controllers are connected to the host over a data network between the channel memory controllers and the host interface. Accordingly, the channel memory controllers are “*configured to receive the plurality of data blocks through the data network.*”

Thus, because Jeddeloh’s RAID XOR circuitry and channel memory controllers receive write data over interconnections to the host interface, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶171-75.

11. Claim 7

[7.1] *The nonvolatile memory controller of claim 6, wherein the nonvolatile memory controller further comprises a command distribution unit configured to distribute the plurality of data update commands to command processing units of the plurality of command processing units,*

See [5.1]. Ex.1003, ¶176.

[7.2] *determine generation of the parity block is complete, and distribute the parity write command to a command processing unit of the plurality of command processing units in response to determining the parity block is complete.*

See [5.2]. Ex.1003, ¶177.

12. Claim 8

[8.1] *The nonvolatile memory controller of claim 7, wherein each command processing unit of the plurality of command processing units comprises a command processing unit command queue for storing commands distributed to the command processing unit from the command distribution unit.*

Jeddeloh explains that each of the channel memory control circuits (“*command processing units*”) are capable of queuing commands. “A page write

command can be sent, e.g., queued, to the desired channel memory controller 338-1, . . . , 338-N and executed on the one or more solid state memory devices 310-1, . . . , 310-N.” Ex.1005, [0040].

Thus, because Jeddeloh’s channel memory controllers queue received commands, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶178-79.

13. Claim 9

[9.1] *The nonvolatile memory controller of claim 7, further comprising a context memory including a plurality of page frames, wherein each command of the plurality of commands includes a flow identifier, and the command distribution unit is further configured to allocate a page frame of the context memory to the data stripe operation by associating the flow identifier with the page frame.*

First, for the reasons explained above at [1.5], Kleiman describes a XOR buffer (“*context memory*”) that includes a plurality of XOR accumulators (“*plurality of page frames*”).

Second, Kleiman’s fly-by XOR element allocates certain XOR accumulators to certain stripe operations using 20-bits of a bus address value associated with an address in host memory 120 (“*the command distribution unit is further configured to allocate a page frame of the context memory to the data stripe operation by associating the flow identifier with the page frame*”). “To allocate an XOR accumulator buffer for a new RAID stripe ... the timing and control element 320 causes the CAM 330 to be loaded with a 20-bit memory address for a memory buffer in the memory 120 ... and associates that 20-bit memory address with a 5-bit

identifier for one of the 32 XOR accumulator buffers.” Ex.1006, 6:36-55. In other words, Kleiman uses the 20-bits of the memory address where data for the data stripe operation is stored in host memory 120 as an identifier for that data stripe. That memory address is then used to allocate an XOR buffer (“*page frame*”) to the data stripe operation.

As the data commands pass through the bus, Kleiman’s technique looks at 20-bits of the bus address to determine which XOR accumulator should be used. Kleiman’s “processor 110 allocates a memory buffer in the memory 120, and sets up data to be DMA transferred to or from the memory buffer and one of the disk drives 140.” Ex.1006, 3:45-48. “At a step 223, the DMA transfer is performed. The fly-by XOR element 180 monitors the mezzanine bus 160 and attempts to recognize transfers to or from the memory buffer.” Ex.1006, 3:57-59. “To monitor the mezzanine bus 160 as in the step 223, the timing and control element 320 selects the bus address path 312 for coupling to the tag port 332.” Ex.1006, 6:56-64. “When the high-order 20 bits of a memory address (or an address within a null memory buffer) appearing on the mezzanine bus 160 matches one of the 20-bit tags in the CAM 330, the CAM 330 matches bits 12 through 31 of the bus address and associates a 5-bit value with that 20-bit tag.” Ex.1006, 6:56-64

Then, “[t]he associated 5-bit value is used to select one of the 32 XOR accumulator buffers in the RAM 340.” Ex.1006, 6:65-66. Accordingly, as the write

commands pass through the data bus, the associated bus address is used as a “*flow identifier*” (“*wherein each command of the plurality of commands includes a flow identifier*”).

As explained above at X.C.3, it would have been obvious for Jeddeloh’s RAID XOR circuitry 226 of the host-memory translation circuit (“*command distribution unit*”) to be implemented according to Kleiman’s on-the-fly technique. In doing so, the RAID XOR circuitry 226 of the host-memory translation circuit (“*command distribution unit*”) would use 20 bits of the bus address (“*flow identifier*”) to associate the command of the data stripe operation with a specific XOR accumulator (“*page frame of the context memory*”).

Thus, Jeddeloh’ RAID XOR circuitry—as implemented using Kleiman’s technique uses 20 bits of a bus address to associate write commands with a particular XOR accumulator, Jeddeloh and Kleiman render this limitation obvious. Ex.1003, ¶180-86.

14. Claim 10

[10.1] *The nonvolatile memory controller of claim 1, wherein the plurality of command processing units is further configured to perform a plurality of data stripe operations in parallel.*

Jeddeloh explains that data stripe operations are performed “simultaneously” (“*in parallel*”). “In one or more embodiments, the plurality of page write operations in the stripe write operation can be performed substantially

simultaneously across the plurality of channels.” Ex.1005, [0040]. Additionally, in Kleiman’s on-the-fly technique, “up to 32 different RAID stripes can have XOR accumulation performed simultaneously by the fly-by XOR element 180.” Ex.1006, 5:32-37.

Thus, because Jeddeloh’s channel control circuits perform page write operations substantially simultaneously, and Kleiman’s on-the-fly-technique performs several stripe operations in parallel, Jeddeloh and Kleiman render this limitation obvious. Ex.1003, ¶¶187-89.

15. Claim 11

[11.0] *A nonvolatile memory controller comprising:*

For the reasons explained at [1.0], the prior art teaches a “*non-volatile memory controller*” as claimed. Ex.1003, ¶190.

[11.1] *a command distribution unit configured to receive a plurality of commands for performing a data stripe operation on a plurality of data blocks, the plurality of commands comprising a plurality of data update commands and a parity write command;*

First, for the reasons explained above at [5.1], Jeddeloh’s device includes host-memory translation circuitry that distributes page write commands of a stripe operation to the channel memory controllers 338. The channel memory controllers receive the plurality of commands for the stripe operations as discussed above at [1.1]. Thus, Jeddeloh teaches a “*command distribution unit configured to receive a*

plurality of commands for performing a data stripe operation on a plurality of data blocks” as claimed.

Second, as explained above at [1.2], the plurality of commands include “*a plurality of data update commands and a parity write command.*” Ex.1003, ¶¶191-92.

[11.2] *a plurality of command processing units coupled to the command distribution unit,*

First, for the reasons explained above at [1.1] Jeddeloh includes a plurality of channel memory controllers 338 (“*a plurality of command processing units*”).

Second, for the reasons explained above at [5.1], Jeddeloh includes a host-memory translation circuitry 216 (“*command distribution unit*”).

Third, as shown in Jeddeloh’s Fig. 1, the channel memory controllers 338 are coupled to the command distribution unit through the switch.

Thus, because Jeddeloh includes a plurality of channel memory controllers coupled to the host translation circuitry, Jeddeloh renders this limitation obvious. Ex.1003, ¶¶193-96.

[11.3] *the command distribution unit further configured to distribute the plurality of data update commands among command processing units of the plurality of command processing units,*

See [1.0]. Ex.1003, ¶197.

[11.4] *each command processing unit receiving a data update command configured to request a data block of the plurality of data blocks based on the*

data update command, receive the data block in response to the request, and write the data block to a nonvolatile memory device; and

For the reasons explained above at [1.3a] and [1.3b], Jeddeloh and Ma render this limitation obvious. Ex.1003, ¶198.

[11.5] *a parity calculator coupled to the plurality of command processing units,*

See [1.4]. Ex.1003, ¶199.

[11.6] *the parity calculator including a context memory including a page frame,*

See [1.5]. Ex.1003, ¶200.

[11.7] *the parity calculator configured to receive the data blocks of the plurality of data blocks as a sequence of data blocks,*

See [1.6]. Ex.1003, ¶201.

[11.8] *[the parity calculator configured] to generate a parity block by storing a first data block of the sequence of data blocks into the page frame and updating the data block stored in the page frame by performing an exclusive-or operation on the data block stored in the page frame and a data block following the first data block in the sequence of data blocks,*

First, for the reasons explained above at [1.7], Jeddeloh teaches “[*the parity calculator configured] to generate a parity block by storing a first data block of the sequence of data blocks into the page frame and updating the data block stored in the page frame.*”

Second, for the reasons explained at [1.7], [2.1] and [3.1], updating the block in the page frame is done “*by performing an exclusive-or operation on the data block stored in the page frame and a data block following the first data block*

in the sequence of data blocks.” Ex.1003, ¶202-03.

[11.9] the command distribution unit further configured to determine generation of the parity block is complete and to distribute the parity write command to a command processing unit of the plurality of command processing units in response to determining the parity block is complete,

See [5.1]. Ex.1003, ¶204.

[11.10] the command processing unit receiving the parity write command configured to write the parity block into the nonvolatile memory device.

See [1.9]. Ex.1003, ¶205.

16. Claim 13

[13.1] The nonvolatile memory controller of claim 12, wherein the context memory includes a plurality of page frames for storing a plurality of parity blocks of a plurality of data stripe operations, and wherein the plurality of command processing units is further configured to perform the plurality of data stripe operations in parallel.

See [9.1]. Ex.1003, ¶206.

17. Claim 14

[14.0] A method of generating parity data, the method comprising:

Jeddeloh describes a method for generating parity data during a data stripe operation. “The present disclosure includes methods and devices for stripe-based memory operation. One method embodiment includes writing data in a first stripe across a storage volume of a plurality of memory devices.” Ex.1005, [0012]. The data stripe operation uses “RAID XOR circuitry 226 [to] calculate parity databased on data received from the host interface circuitry 214.” Ex.1005, [0027]; Ex.1003,

¶207.

[14.1] *distributing a plurality of commands for performing the data stripe operation among a plurality of command processing units in the nonvolatile memory controller, the plurality of commands including a plurality of data update commands and a parity write command;*

For the reasons explained above at [1.1], Jeddeloh renders this limitation obvious. Ex.1003, ¶208.

[14.2] *generating a plurality of data requests by the plurality of command processing units based on the plurality of data update commands;*

For the reasons explained above at [1.3a], Jeddeloh and Ma render this limitation obvious. Ex.1003, ¶209.

[14.3] *receiving a plurality of data blocks at the plurality of command processing units and at a parity calculator of the nonvolatile memory controller in response to the plurality of data requests;*

First, for the reasons explained above at [1.3b], Jeddeloh and Ma render obvious “*receiving a plurality of data blocks at the plurality of command processing units ... in response to the plurality of data requests*” as claimed.

Second, because the data is received by the channel memory circuits 338 from the host, the data passes through the RAID XOR circuitry (“*at a parity calculator of the nonvolatile memory controller*”) as well. Ex.1003, ¶¶210-11.

[14.4] *writing the plurality of data blocks to a plurality of nonvolatile memory devices by the plurality of command processing units;*

For the reasons explained above at [1.3b], Jeddeloh and Ma render this

limitation obvious. Ex.1003, ¶212.

[14.5] *generating a parity block in a page frame of a context memory by the parity calculator based on the plurality of data blocks by initializing a parity block in the page frame by storing a first data block of the sequence of data blocks into the page frame and updating the data block stored in the page frame with each data block following the first data block in the sequence of data blocks, without storing each data block in a data buffer;*

For the reasons explained above at [1.4]-[1.8], Jeddeloh renders this limitation obvious. Ex.1003, ¶213.

[14.6] *determining that generation of the parity block is complete; and*

For the reasons explained above at [5.2], Jeddeloh renders obvious “*determining that generation of the parity block is complete*” as claimed. Ex.1003, ¶214.

[14.7] *writing the parity block to a nonvolatile memory device by a command processing unit of the plurality of command processing units based on the parity write command after determining generation of the parity block is complete.*

For the reasons explained above at [5.2] and [1.9], Jeddeloh renders obvious “*writing the parity block to a nonvolatile memory device by a command processing unit of the plurality of command processing units based on the parity write command after determining generation of the parity block is complete*” as claimed. Ex.1003, ¶215.

18. Claim 15

[15.1] *The method of claim 14, wherein ... the sequence of data blocks.*

See [2.1]. Ex.1003, ¶216.

19. Claim 16

[16.1] *The method of claim 15, wherein the operation includes an exclusive-or operation.*

See [3.1]. Ex.1003, ¶217.

20. Claim 18

[18.1] *The method of claim 14, wherein ... associating the flow identifier with the page frame.*

See [9.1]. Ex.1003, ¶218.

D. Ground 2: Claims 4 and 17 are obvious in view of Jeddeloh, Kleiman, Ma, and Yashiro.

1. Summary of Yashiro

Yashiro provides evidence that it was known to use a counter to determine whether a parity block is complete. Dr. Min provides a summary of Yashiro in his declaration. Ex.1003, ¶¶219-21

2. Reasons to Combine Yashiro with Jeddeloh, Kleiman, and Ma

A POSITA would have found it obvious to use a counter for determining that the parity block of the Jeddeloh/Kleiman combination is complete. Ex.1003, ¶222.

As explained above at X.C.2, Kleiman describes a known method in which a parity block is calculated on-the-fly. Ex.1006, title, abstract. In particular, Kleiman describes Kleiman describes a “fly-by XOR element 180” that “uses an XOR

buffer 190 for saving partial and final results of XOR operations.” Ex.1006, 3:3-5. Then, “[w]hen all data for the recognized memory buffer has been transferred, ... the processor 110 directs the fly-by XOR element 180 to readout the accumulated parity information from the associated XOR accumulator buffer ... to a storage block on one of the disk drives 140.” Ex.1006, 3:65-4:6; Ex.1003, ¶223.

A POSITA would have found it obvious to look to known techniques for determining when “all data for the recognized memory buffer has been transferred” as described by Kleiman. Ex.1006, 3:65-67; Ex.1003, ¶224. Yashiro provides one example of a known technique in which a counter is used for determining when the parity block is complete. In particular, Yashiro further describes using a counter to determine when the parity block is complete. “The counter N has been cleared to N=0 in the initial state.” Ex.1008, 14:31-32. As the system processes write commands (described in the text accompanying Fig. 12), the “count value of the count N ... is increased by 1.” Ex.1008, 14:30-31. If the counter N has not yet reached the “set [] number of [] times” (i.e., a threshold), then the method returns to processing the next write command with respect to Fig. 12. However, once the counter reaches the threshold, the parity block is complete and the parity block can be stored on the disk in Step 8. *See* Ex.1008, 14:28-55.

A POSITA would have been motivated to use Yashiro’s counter technique because it provides certain benefits.

- **First**, Yashiro is a tried-and-true technique for determining when the parity block is complete. Ex.1003, ¶225.
- **Second**, given that Kleiman stores the parity block upon completion, a POSITA would have found that using a counter to count data blocks would have predictably resulted in accurately determining completion. Ex.1003, ¶225.

A POSITA would have had a reasonable expectation of success because Yashiro discloses a similar operation for calculating parity data as Kleiman—an XOR operation. Ex.1003, ¶226. Kleiman uses “XOR accumulator buffers in the XOR buffer 190 to accumulate the results of the **XOR operation**.” Ex.1006, 3:53-55. Similarly, in Yashiro, “**the exclusive OR** of the data in a plurality of disk apparatuses is calculated and is set to a parity and is held into the disk apparatus for parities.” Ex.1008, 4:22-25.

Accordingly, the combination of Yashiro with Jeddeloh and Kleiman merely represents applying a known technique (Yashiro’s counter) to a known method (Kleiman’s on-the-fly technique) ready for improvement to yield predictable results (accurately determining when the parity block is complete). Ex.1003, ¶¶222-27.

3. Claim 4

[4.1] *The nonvolatile memory controller of claim 1, further comprising a context*

associated with the page frame and including a stripe count for counting each occurrence of updating the parity block, the nonvolatile memory controller further comprising a command distribution unit configured to determine the parity block is generated by determining the stripe count has reached a threshold value.

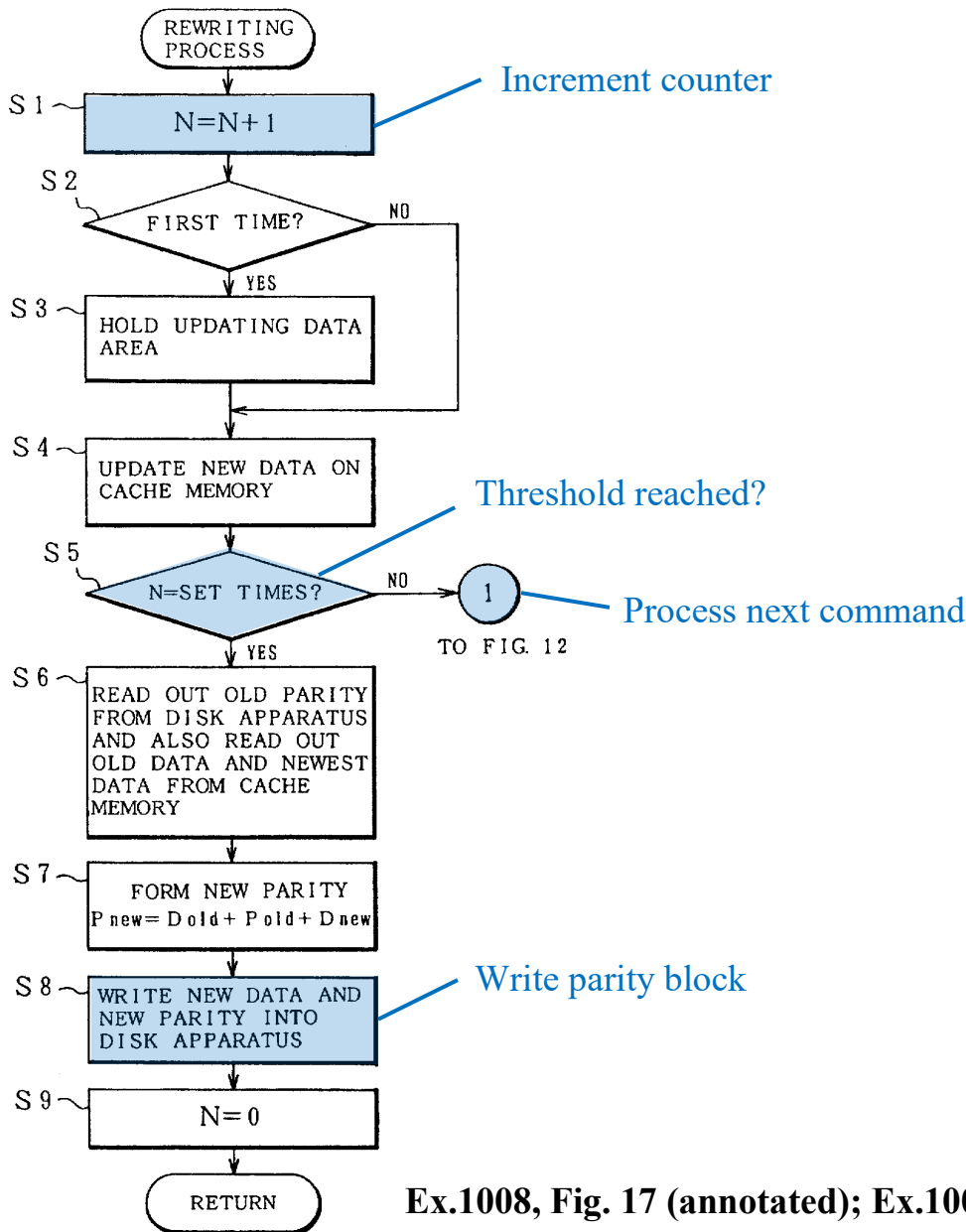
First, Jeddeloh's RAID XOR circuitry as implemented according to Kleiman's teachings maintains a context. The term "context" as used in computer science and processor/memory systems means the minimal set of data needed to complete a task. Ex.1003, ¶228. For example, the "context" may include the partial or final results in the XOR accumulator ("*page frame*") of the XOR buffer. "The fly-by XOR element 180 uses an XOR buffer 190 for saving partial and final results of XOR operations." Ex.1006, 3:3-5.

Second, it would have been obvious for Kleiman's on-the-fly XOR technique to maintain a stripe count as part of that context. Kleiman explains that the system updates the XOR buffer with each passing data sector. "Ex.1006, 3:60-64. Then, "[w]hen all data for the recognized memory buffer has been transferred ... the processor 110 directs the fly-by XOR element 180 to readout the accumulated parity information from the associated XOR accumulator buffer." Ex.1006, 3:65-4:3.

One known way to determine whether each of the sectors has passed through would be to maintain a counter. *See generally*, Ex.1008, Fig. 17. Once the counter reaches the threshold, the parity block is complete and the parity block can be

stored on the disk in Step 8. This process is illustrated below in Fig. 17. See

Ex.1008, 14:28-55.



Ex.1008, Fig. 17 (annotated); Ex.1003, ¶230.

For the reasons explained above at X.D.2. it would have been obvious for the Jeddeloh/Kleiman parity calculator to utilize a counter. In Jeddeloh's example

where there are seven data blocks and one parity block per stripe, the counter would increment to seven. When the counter reaches seven, it is determined that the parity block is complete.

Thus, because it would have been obvious for Kleiman's on-the-fly technique to utilize a counter to determine when the parity block is complete, Jeddeloh, Kleiman, and Yashiro render this limitation obvious. Ex.1003, ¶¶228-32.

4. Claim 17

[17.0] The method of claim 14, further comprising: initializing a stripe count to a predetermined value; and decrementing the stripe count in response to each occurrence of updating the parity block, wherein determining generation of the parity block is complete comprises determining the stripe count has reached a threshold value.

As explained above at [4.1], it would have been obvious to use a counter to determine when the parity block is complete. Claim 17 recites a similar concept to that of claim 4. But rather than incrementing the counter from zero to a threshold, claim 17 recites decrementing the counter from the threshold to zero. This alternative approach is merely an obvious design choice. Indeed, POSITAs understood that incrementing or decrementing in this manner were interchangeable. *See* Ex.1013, 5:9-11 (“alternatively use a decrementing counter rather than an incrementing counter, and could increment or decrement the counter by any quantity per event”).

Accordingly, in the case of Jeddeloh's system, it would have been obvious

to initialize the counter to seven, and then decrement the counter as each successive block passes through. Then, when the counter reaches zero (“*a threshold value*”) it is determined that the parity block is complete. In Jeddeloh’s example where there are seven data blocks and one parity block per stripe, the counter would be initialized to seven. When the decrementing counter reaches zero, it is determined that the parity block is complete.

Thus, because it would have been obvious for Kleiman’s on-the-fly technique to utilize a counter to determine when the parity block is complete, whether incrementing or decrementing, Jeddeloh, Kleiman, and Yashiro render this limitation obvious. Ex.1003, ¶¶233-35.

XI. DISCRETIONARY DENIAL IS NOT WARRANTED

Petitioner respectfully submits that the Board should not exercise its discretion under 35 U.S.C. §§314(a) or 325(d) to deny this Petition.

A. The Fintiv factors favor institution.

Denial would be improper based on factors in *Apple v. Fintiv*, IPR2020-00019, Paper 11 (P.T.A.B. Mar. 20, 2020) (precedential).

1. No evidence regarding a stay

No motion to stay has been filed, so the Board should not infer the outcome of such a motion.

2. Parallel proceeding trial date

This factor weighs against discretionary denial because no trial date has currently been scheduled. Moreover, the projected trial date—based on median time-to-trial statistics—is in January of 2027¹, which is well after the projected date for the Board to issue a Final Written Decision in July of 2026. Because the projected trial date is approximately six months *after* the Board’s expected date for issuing a Final Written Decision, this factor weighs against discretionary denial.

3. Investment in the parallel proceeding

The investment in the co-pending litigation has been minimal. No schedule has been set. Thus, the Markman hearing is unlikely to occur before the Board issues an institution decision. This lack of investment favors institution.

4. Overlapping issues with the parallel proceeding

The co-pending litigation is in its early stages. Petitioner served its preliminary invalidity contentions on December 16, 2024. However, Petitioner has not served its final invalidity contentions, nor has the deadline for doing so been set. The extent of overlap is thus speculative at this point in time.

¹ The current average time-to-trial for the district court hearing the co-pending litigation is 29.1 months. *See* Ex.1015 and Ex.1016.

5. Petitioner is a defendant

Petitioner is a defendant in the co-pending litigation. That is true of most Petitioners in IPR proceedings, making this factor neutral.

6. Other circumstances

“[T]he PTAB will not deny institution of an IPR or PGR under *Fintiv* (i) when a petition presents compelling evidence of unpatentability.” Ex.1017, 9. Here, the petition—along with Dr. Min’s expert testimony—plainly shows that the ’346 patent claims no more than known concepts related to distributed processing and parity calculation on-the-fly.

B. Advanced Bionics Test Favors Institution.

Denial under 35 U.S.C. § 325(d) is not warranted because the challenges presented in this petition are neither cumulative nor redundant to the prosecution of the ’346 patent. Neither Jeddeloh, Kleiman, Ma, or Yashiro were before the patent office during examination of the ’346 patent. Accordingly, discretionary denial under 325(d) is not warranted.

XII. CONCLUSION

Petitioner has established a reasonable likelihood that the Challenged Claims are unpatentable.

Respectfully submitted,

Dated: January 23, 2025
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, Texas 75201
Customer No. 27683

/David M. O'Dell/
David M. O'Dell
Lead Counsel for Petitioner
Registration No. 42,044

XIII. MANDATORY NOTICES

A. Real Party-in-Interest

The real parties-in-interest (“RPI”) are Sandisk Technologies, Inc. and Western Digital Technologies, Inc. (“Petitioner”).

B. Related Matters

Pursuant to 37 C.F.R. § 42.8(b)(2), to the best knowledge of the Petitioner, the ’346 patent is or was involved in the following cases:

Case Heading	Number	Court	Filed	Status
<i>Polaris PowerLED Technologies, LLC v. Western Digital Corporation et al.</i>	2-24-cv-02864	CDCA	April 9, 2024	Pending
<i>Polaris PowerLED Technologies, LLC v. Western Digital Corporation et al.</i>	1-24-cv-00101	WDTX	January 30, 2024	Pending

C. Lead and Back-up Counsel and Service Information

Lead Counsel

David O’Dell
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, TX 75201

Phone: (972) 739-8635
Fax: (214) 200-0853
david.odell.ipr@haynesboone.com
USPTO Reg. No. 42,044

Back-up Counsel

Calmann J. Clements
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300

Phone: (972) 739-8638
Fax: (214) 200-0853
calmann.clements.ipr@haynesboone.com

Dallas, TX 75201

USPTO Reg. No. 66,910

Hong Shi
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, TX 75201

Phone: (512) 867-8440
Fax: (214) 200-0853
hong.shi.ipr@haynesboone.com
USPTO Reg. No. 69,009

Please address all correspondence in this proceeding to lead and back-up counsel. Petitioner consents to service in this proceeding by email at the addresses above

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 13,967. 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.

Dated: January 23, 2025

/David M. O'Dell/

David M. O'Dell
Lead Counsel for Petitioner
Registration No. 42,044

CERTIFICATE OF SERVICE

The undersigned certifies that, in accordance with 37 C.F.R. § 42.6(e) and 37 C.F.R. § 42.105, service was made on Patent Owner as detailed below.

Date of service January 23, 2025

Manner of service FEDERAL EXPRESS

Documents served Petition for *Inter Partes* Review Under 35 U.S.C. § 312 and 37 C.F.R. § 42.104 of U.S. 8,601,346; Petitioner's Power of Attorney; Petitioner's Exhibit List; All Exhibits.

Persons served Glass & Associates
236 N. Santa Cruz Avenue, Suite 243
Los Gatos, CA 95030

/David M. O'Dell/
David M. O'Dell
Lead Counsel for Petitioner
Registration No. 42,044