

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

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

U.S. Patent No. 8,601,346

**DECLARATION OF DR. PAUL MIN,
UNDER 37 C.F.R. § 1.68 IN SUPPORT OF PETITION FOR
INTER PARTES REVIEW**

TABLE OF CONTENTS

I.	Introduction.....	4
II.	Qualifications and Professional Experience.....	6
III.	Level of Ordinary Skill in the Art	11
IV.	Relevant Legal Standards	12
V.	Background.....	13
	A. Memory Systems.....	13
	B. RAID	14
	C. Parity Data.....	15
VI.	Overview of the '346 Patent.....	15
	A. Summary of the '346 patent.....	15
	B. Prosecution History of the '346 patent.....	19
VII.	Claim Construction.....	20
	A. “without storing each data block in a data buffer.”	20
VIII.	Identification of how the Claims are Unpatentable.....	23
	A. Ground 1: Claims 1-3, 5-11, 13-16, and 18 are obvious in view of Jeddeloh, Kleiman, and Ma.	23
	1. Summary of Jeddeloh.	23
	2. Summary of Kleiman.....	27
	3. Reasons to Combine Jeddeloh and Kleiman.	29
	4. Summary of Ma.	32
	5. Reasons to Combine Ma with Jeddeloh and Kleiman...	34
	6. Claim 1	37

7.	Claim 2.....	73
8.	Claim 3.....	74
9.	Claim 5.....	74
10.	Claim 6.....	78
11.	Claim 7.....	88
12.	Claim 8.....	88
13.	Claim 9.....	89
14.	Claim 10.....	92
15.	Claim 11.....	94
16.	Claim 13.....	97
17.	Claim 14.....	97
18.	Claim 15.....	100
19.	Claim 16.....	100
20.	Claim 18.....	100
B.	Ground 2: Claims 4 and 17 are obvious in view of Jeddeloh, Kleiman, Ma, and Yashiro.	100
1.	Summary of Yashiro.....	100
2.	Reasons to Combine Yashiro with Jeddeloh, Kleiman, and Ma	102
3.	Claim 4.....	105
4.	Claim 17.....	109
IX.	Conclusion	111

I, Paul Min, do hereby declare as follows:

I. INTRODUCTION

1. I am making this declaration at the request of Western Digital Technologies, Inc. in the matter of the *Inter Partes* Review of U.S. Patent No. 8,601,346 (“the ’346 patent”) to Onufryk *et al.*

2. I am being compensated for my work in this matter at my usual hourly rate for this work. I am also being reimbursed for reasonable and customary expenses associated with my work and testimony in this investigation. My compensation is not contingent on the outcome of this matter or the specifics of my testimony.

3. I have been asked to provide my opinions regarding whether claims 1-11 and 13-18 (“the Challenged Claims”) of the ’346 patent are unpatentable as they would have been obvious to a person having ordinary skill in the art (“POSITA”) at the time of the alleged invention, in light of the prior art. It is my opinion that all of the limitations of the Challenged Claims would have been obvious to a POSITA.

4. In the preparation of this declaration, I have studied:

- a. the ’346 patent, Ex.1001;
- b. the prosecution history of the ’346 patent (“’346 File History”), Ex.1002;

- c. U.S. Patent Publication No. 2011/0078496A1 to Jeddeloh (“Jeddeloh”), Ex.1005;
 - d. U.S. Patent No. 5,950,225 to Kleiman (“Kleiman”), Ex.1006;
 - e. U.S. Patent No. 8,341,332 to Ma et al. (“Ma”), Ex.1007;
 - f. U.S. Patent No. 5,787,460 to Yashiro et al. (“Yashiro”), Ex.1008; and
5. In forming the opinions expressed below, I have considered:
- the documents listed above;
 - the relevant legal standards, including the standard for obviousness, and any additional authoritative documents as cited in the body of this declaration; and
 - my own knowledge and experience based upon my work in the field of optical communication as described below, as well as the following materials:
- a. U.S. Patent No. 6,161,165 to Solomon et al. (“Solomon”), Ex.1009;
 - b. The Authoritative Dictionary of IEEE Standards Terms: Seventh Edition; (“IEEE”), Ex.1011;
 - c. U.S. Patent No. 8,074,149 to Stolorowicz (“Stolorowicz”), Ex.1012;
 - d. U.S. Patent No. 8,819,699 to Cota-Robles (“Cota-Robles”), Ex.1013;
- and
- e. U.S. Patent No. 6,092,215 to Hodges et al. (“Hodges”), Ex.1014.

6. Unless otherwise noted, all emphasis in any quoted material has been added.

II. QUALIFICATIONS AND PROFESSIONAL EXPERIENCE

7. My qualifications and professional experience are described in my *Curriculum Vitae*, a copy of which can be found in Exhibit 1004. The following is a brief summary of my relevant qualifications and professional experience.

8. My qualifications and professional experience are described in my *Curriculum Vitae*, a copy of which can be found in Exhibit 1004. The following is a brief summary of my relevant qualifications and professional experience.

9. I received B.S., M.S., and Ph.D. degrees in Electrical Engineering from the University of Michigan in 1982, 1984, and 1987, respectively.

10. I am an internationally recognized expert in the field of high speed communication and computing, and the development of semiconductor devices for the related applications. I joined Washington University in St. Louis in 1990. I currently serve as a Senior Professor at the Preston M. Green Department of Electrical & Systems Engineering of the McKelvey School of Engineering.

11. Before joining Washington University, I spent three years at Bellcore (now Telcordia Technologies, Inc., which is part of Ericsson), where I was a lead systems engineer for the Bell Operating Companies in transitioning away from their voice centric networks at the time of the AT&T divestiture, into multi-services

capable networks suited for the Internet era. I received an Outstanding Achievement Award for his contributions by Bellcore in 1989.

12. After coming to Washington University, I helped the development of a communication, computer, and electronics curricula in the School of Engineering and Applied Science. I regularly teach undergraduate and graduate courses, and supervise students to conduct research in these areas.

13. I designed one of the earliest nationwide, commercial-scale, code division multiple access, or CDMA, networks after winning a cellular communication license from the Korean Government through a highly competitive selection process. I was the leading author of the technical blueprint for this wireless cellular network. Since then, I have been continuously researching in 3G UMTS / 4G LTE / 5G NR IoT wireless networks as well as various generations of WiFi technology.

14. I have collaborated and consulted with numerous companies and organizations around the world, and served as an adviser for several national governments. I also founded and managed two start-up companies during 1997 - 2008. One of the commercial products I developed in my companies received a best product award in 2002 from a major trade journal for the Internet industry. In the companies I founded as well as part of my research activities at Washington University, I have designed and implemented numerous high-speed, high-

performance electronic circuitries.

15. I am a recipient of the Best Paper Award at MOBILITY 2011 in October 2011 in Barcelona, Spain, and a recipient of the Best Paper Award at 18th ISATA Award of Technical Excellence in 1988 in Florence, Italy. I received a Research Initiation Award in 1993 from the Defence Advance Research Project Agency (DARPA). I was a Rockwell Fellow in 1988 and 1989. In addition, I was the recipient of the Best Teaching Assistant Award the Best Graduate Student Award at the University of Michigan in Ann Arbor.

16. I was a member of the Presidential Business Advisors Committee (to President George W. Bush), and was named 2002 Businessman of the Year, for Missouri, by the Wall Street Journal for my entrepreneurial effort. I have served as an Ambassador for the McDonnell International Scholars Academy (2007-2013).

17. I have served as the Chair of the Saint Louis Section of the Institute for Electrical and Electronics Engineering (IEEE) (2014), and served as a member of the Executive Committee (2011-2015) and the Chair of the Communication Society for the Saint Louis Section of the IEEE for a number of years. I have been an organizer for several international symposiums, a guest editor of international journals, and have given a number of invited talks. I hold 11 U.S. patents.

18. My research area includes switching, routing, performance control and security in the communication networks. For example, I and my students

invented and implemented several methods of high performance switching, including multi-channel switches that alleviate the speed and performance constraints of electronic switches. The pattern matching engines that my research group developed are used for processing complex instructions embedded in communication packets at billions of packets per second. The content search engine that my research group developed has been incorporated into numerous cybersecurity systems in active networks and is being developed for machine learning and data mining for various applications including cybersecurity and the Internet of Things (IoT).

19. I have developed and implemented a number of semiconductor integrated circuit chips. These integrated circuit chips include switch fabrics, processors, high speed interfaces, protocol transposers, controllers, content search engine, pattern matching, and memory cells and arrays. Many of the integrated circuit chips I have developed incorporates high speed memory modules interacting with the memory controllers to perform various types of operations including pointer processing, pattern matching, tree-searching, and mathematical manipulation that require extremely fast read/write/bulk-transfer from/to the memory contents. For example, the latest generations of the switching fabric that I developed incorporated what is known in the switching field as the virtual output queue based on shared memory. In the switch fabric based on the virtual output

queue based on shared memory architecture, there is no space-division switching, and rather, the switching is performed from the shared memory that takes in all of the incoming packets, which are manipulated by the pointers that organized the packets in the memory.

20. In addition, I am actively involved in the development of virtualized cluster computing networks. I developed algorithms to control code downloading from the network servers to the end-user units. Based on these algorithms, applications running on the smartphones and portable devices can run much faster without consuming excessive battery powers. By quantifying the information related to the usage patterns, a large amount of user data can be stored, which can help improve the mobile application significantly.

21. I also focus on future generations of wireless technology. I have developed methods that enable coordinated transmission and reception of wireless signals among multiple base stations. Previously, radio signals from multiple base stations are considered as interference. Leveraging on these results, a cluster of base stations can work together as a single resource enhancing the quality of radio signals across extremely noisy wireless channels. This research is critical in accommodating the next generation of the Internet and wireless networks, which integrates numerous Internet of Things (IoT) as part of the intelligent networks.

III. LEVEL OF ORDINARY SKILL IN THE ART

22. I understand there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including (1) the levels of education and experience of persons working in the field at the time of the invention; (2) the sophistication of the technology; (3) the types of problems encountered in the field; and (4) the prior art solutions to those problems.

23. A person of ordinary skill in the art (“POSITA”) in the field of the ’346 patent, as of its earliest possible filing date of August 16, 2010, had a working knowledge of the memory system art that is pertinent to the ’346 patent, including memory systems and memory controllers. That person would have a bachelor’s degree in electrical engineering, computer science, or equivalent training, and approximately two years of experience working in the field of memory systems. Lack of work experience can be remedied by additional education, and vice versa.

24. For purposes of this Declaration, in general, and unless otherwise noted, my statements and opinions, such as those regarding my experience and the understanding of a POSITA generally (and specifically related to the references I consulted herein), reflect the knowledge that existed in the field as of the priority date of the ’346 patent. Unless otherwise stated, when I provide my understanding and analysis below, it is consistent with the level of a POSITA prior to the priority date of the ’346 patent.

IV. RELEVANT LEGAL STANDARDS

25. I am not an attorney. In preparing and expressing my opinions and considering the subject matter of the '346 patent, I am relying on certain basic legal principles that counsel have explained to me. These principles are discussed below.

26. I understand that prior art to the '346 patent includes patents and printed publications in the relevant art that predate the priority date of the alleged invention recited in the '346 patent. For purposes of this Declaration, I am applying August 16, 2010 as the earliest possible priority date of the '346 patent.

27. I have been informed that a claimed invention is unpatentable under 35 U.S.C. § 103 if the differences between the invention and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. I have also been informed by counsel that the obviousness analysis takes into account factual inquiries including the level of ordinary skill in the art, the scope and content of the prior art, and the differences between the prior art and the claimed subject matter.

28. I have been informed by counsel that the Supreme Court has recognized several rationales for combining references or modifying a reference to show obviousness of claimed subject matter. Some of these rationales include the

following: (a) combining prior art elements according to known methods to yield predictable results; (b) simple substitution of one known element for another to obtain predictable results; (c) use of a known technique to improve a similar device (method, or product) in the same way; (d) applying a known technique to a known device (method, or product) ready for improvement to yield predictable results; (e) choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success; and (f) some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

V. BACKGROUND

A. Memory Systems

29. “Memory devices are typically provided as internal, semiconductor, integrated circuits in computers or other electronic devices. There are many different types of memory including volatile and non-volatile memory.” Jeddelloh, [0002]. “Memory devices can be combined together to form a solid state drive (SSD).” Jeddelloh, [0003]. “An SSD can include a number of memory devices, e.g., a number of memory chips.” Jeddelloh, [0005]. Each memory chip may include several “dies and/or logical units.” Jeddelloh, [0005]. Each die may include several memory arrays which are organized into “a number of physical pages, and the

physical pages can be organized into a number of blocks.” Jeddelloh, [0005].

30. When a host writes data to a memory device, it sends a write command or write request to the memory controller. In some cases, the data to be written is included in a write request packet. In other cases, the data is separate from the write request packet. In this case, when the controller processes the write request, it pulls the memory from an address designated in the write request.

B. RAID

31. A redundant array of independent disks (RAID) is a storage scheme that is used to “divide and/or replicate data among multiple memory devices.” Jeddelloh, [0006]. When using a RAID scheme, “[t]he multiple memory devices in a RAID array may appear to a user and the operating system of a computer as a single memory device.” Jeddelloh, [0006]. When writing data with a RAID scheme, “write data can be striped across multiple solid state memory devices in a RAID operation.” Jeddelloh, [0028].

32. Striping data refers to the process of distributing data across multiple, different memory devices. For example, a file may be stored in memory such that different pieces of that file are stored on different memory devices. A data stripe in a RAID scheme includes write data (i.e., the data that makes up an actual file) and parity data. In one example described by Jeddelloh, “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.” Jeddelloh, [0044].

C. Parity Data

33. Parity data is used to recover lost data. It is known that a “memory channel, a memory device, a block of memory, a page of memory, or another portion of a memory device could experience an uncorrectable error.” Jeddelloh, [0047]. However, “[s]toring parity data can allow data associated with an effectively removed portion of a memory stripe to be recreated.” If a particular memory device within a stripe fails, the remaining data plus the parity data can be used to recreate the piece of the stripe that was on the failed memory device.

34. Parity data is commonly generated using the XOR operation. “The parity data is an XOR of the data in each of the other disks. The disks are typically striped so that sectors in each of the disks belong to a stripe.” Solomon, 5:21-23. Due to the high speeds at which memory operates, this parity data is often generated “on-the-fly.” In other words, as each of the pieces of data within a stripe passes through the memory controller, it continually updates the parity block with another XOR operation until all of the write data has passed.

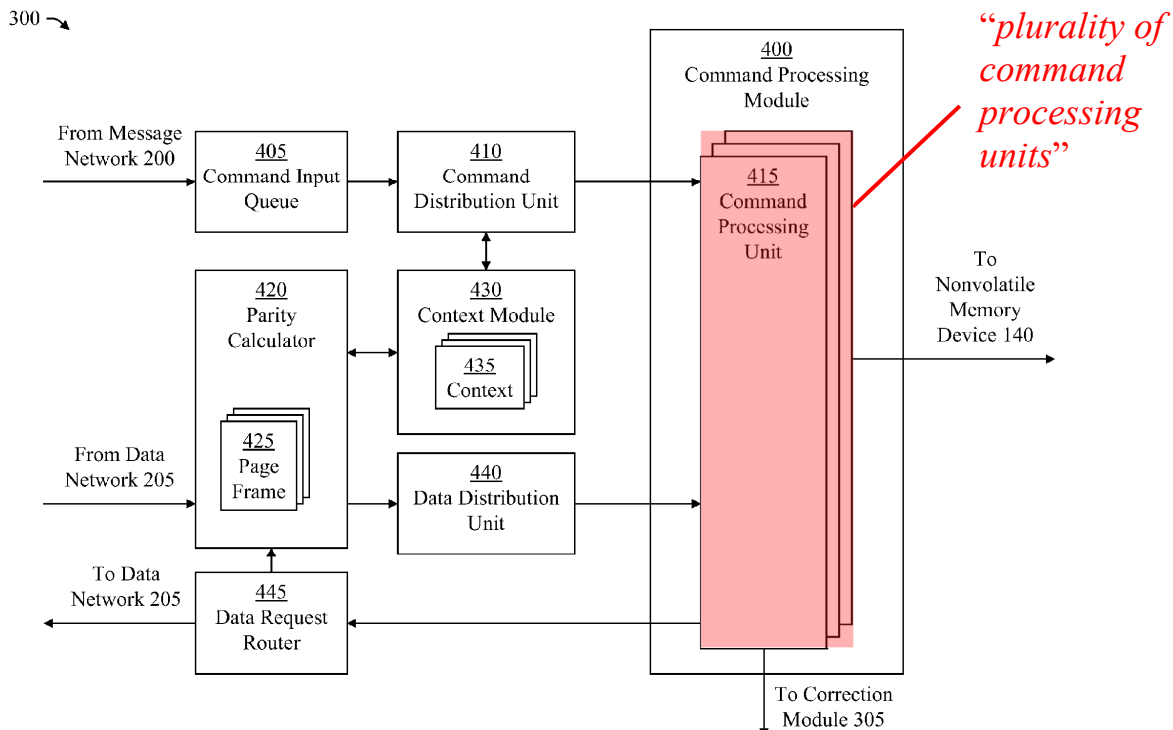
VI. OVERVIEW OF THE ’346 PATENT

A. Summary of the ’346 patent

35. 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.” ’346 patent, 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.” ’346 patent, abstract. The non-volatile memory controller also includes a “parity calculator [that] generates a parity block.” ’346 patent, abstract.

36. 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.” ’346 patent, 1:53-55. The plurality of command processing units are shown in Fig. 4 below.

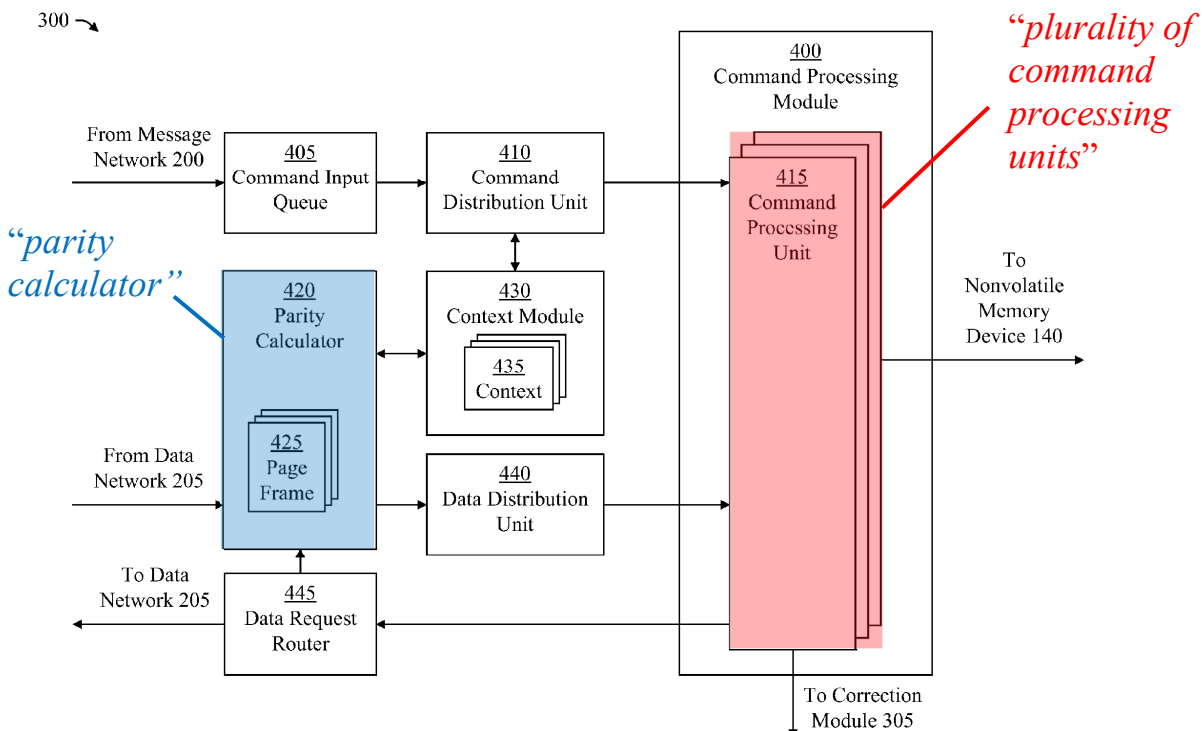


Ex.1001, Fig. 4 (annotated).

37. 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.” '346 patent, 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.” '346 patent, 2:16-17. These “data buffers consume a considerable amount of area and power in integrated circuit implementations of the disk drive controller.” '346 patent, 1:44-46.

38. 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.” ’346 patent, 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.” ’346 patent, 1:63-67. The parity calculator is shown in Fig. 4 below.



’346 patent, Fig. 4 (annotated).

39. 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.

40. 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.

B. Prosecution History of the ’346 patent.

41. 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”).

42. 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.

43. 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.

VII. CLAIM CONSTRUCTION

44. It is my understanding that in order to properly evaluate the ’346 patent, the terms of the claims must first be interpreted. It is my understanding that for the purposes of this *inter partes* review, the claims are to be construed under the so-called *Phillips* standard, under which claim terms are given their ordinary and customary meaning as would be understood by one of ordinary skill in the art in light of the specification and prosecution history, unless the inventor has set forth a special meaning for a term. The term below would have been understood by a POSITA as explained below.

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

45. Based on my review of the structure of the claims as well as the specification and the prosecution history, it is my opinion that 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.

46. 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*.

47. 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.

48. 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.

49. 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.

50. 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*.

VIII. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE

51. I have been asked to provide my opinion as to whether the Challenged Claims of the '346 patent would have been obvious in view of the prior art. The discussion below provides a detailed analysis of how the prior art references identified below teach the limitations of the Challenged Claims of the '346 patent.

52. As part of my analysis, I have considered the scope and content of the prior art and any differences between the alleged invention and the prior art. I describe in detail below the scope and content of the prior art, as well as any differences between the alleged invention and the prior art, on an element-by-element basis for each Challenged Claims of the '346 patent.

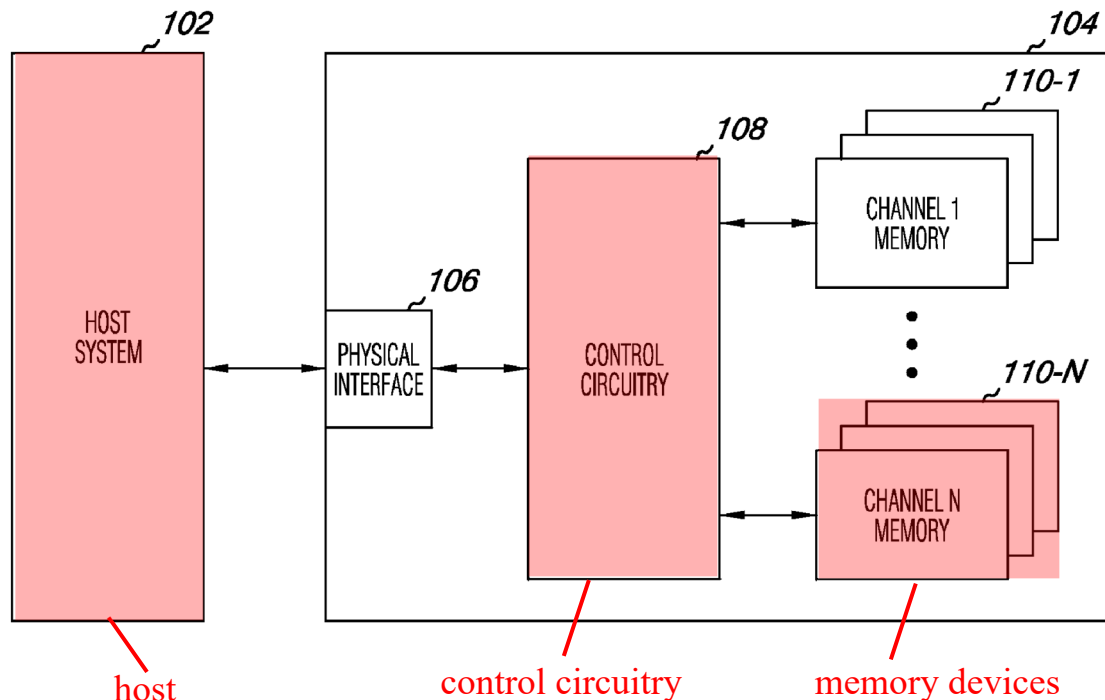
53. As described in detail below, the alleged invention of the Challenged Claims would have been obvious in view of the teachings of the identified prior art references as well as the knowledge of a POSITA.

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

1. Summary of Jeddelloh.

54. Jeddelloh, like the '346 patent, "relates generally to semiconductor memory devices." Ex.1005, 1:5-6. Consistent with its title, Jeddelloh describes a

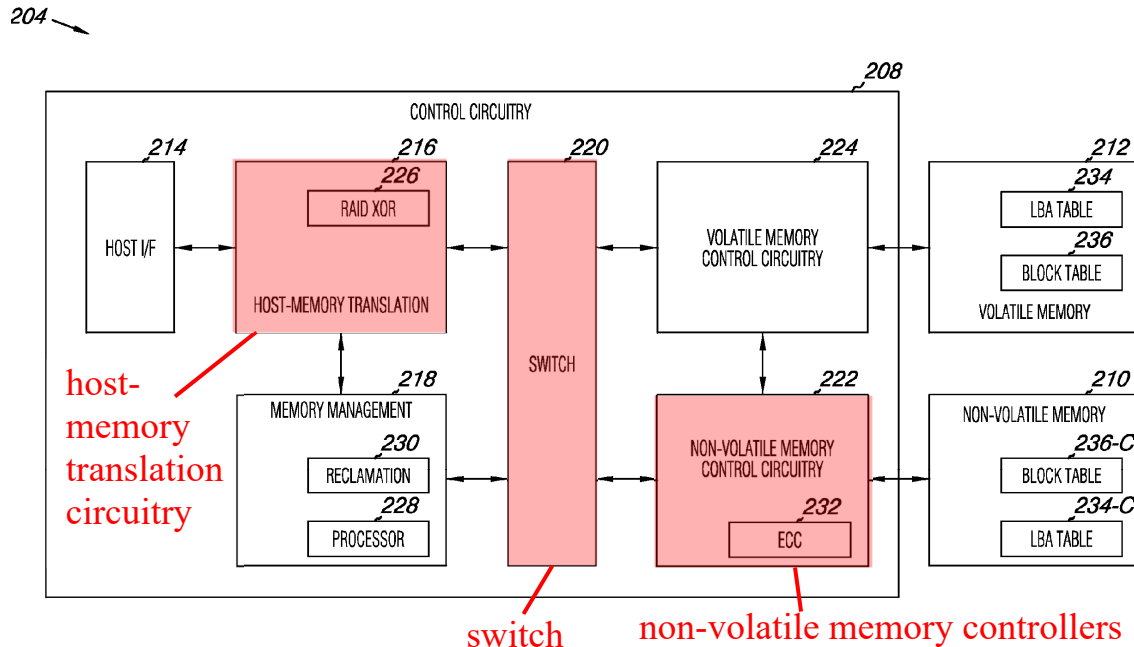
“stripe-based memory operation.” Ex.1005, 1:7. To perform stripe-based memory operations, Jeddeloh’s system includes memory control circuitry 108 for managing memory operations between a host 102 and the multiple memory devices 110 of the stripe operation: “In the embodiment illustrated in FIG. 1, the memory system 104, e.g., a solid state drive (SSD), can include a physical host interface 106, memory system control circuitry 108, e.g., an SSD controller, and one or more solid state memory devices 110-1, . . . , 110-N.” Jeddeloh, [0015].



Jeddeloh, Fig. 1 (annotated).

55. 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.” Jeddelloh, [0024].



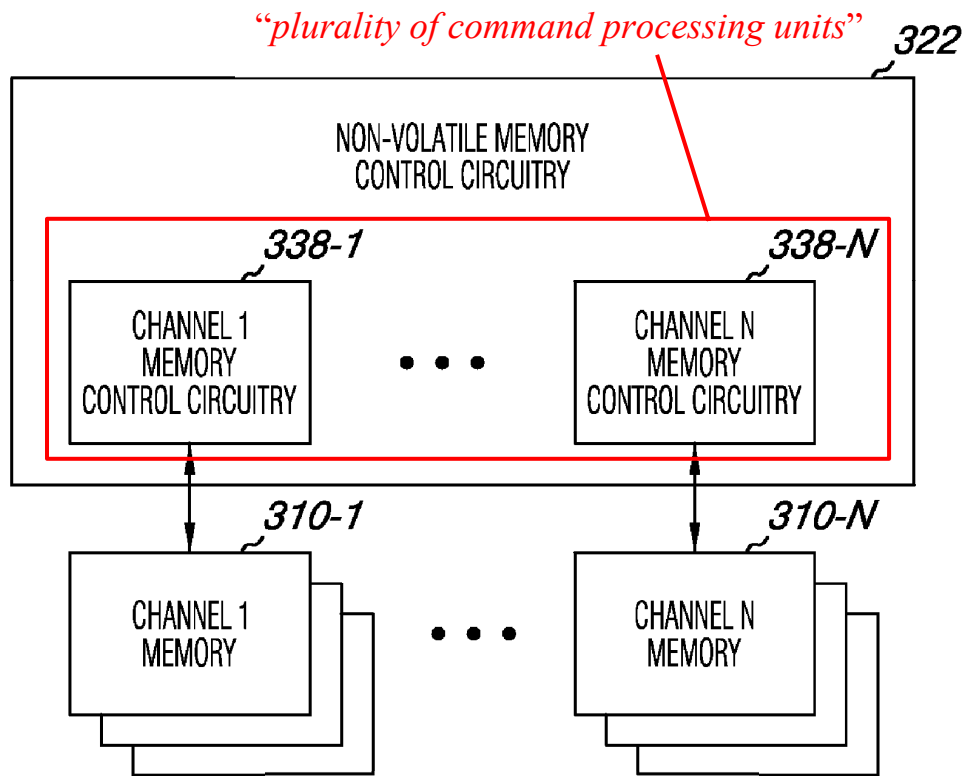
Jeddelloh, Fig. 1 (annotated).

56. 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 22.” Jeddelloh, [0033]. Jeddelloh further explains that “the switch 220 can be a direct memory access (DMA) module.” Jeddelloh, [0033].

57. 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.” Jeddeloh, [0027]. “The memory system, e.g., RAID XOR circuitry 226 associated with host-memory translation circuitry 216 in FIG. 2, can calculate parity data for the write data.” Jeddeloh, [0043].

58. Also like the '346 patent, Jeddeloh 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 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.



Jeddeloh, Fig. 3 (annotated).

59. Thus, Jeddeloh 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.

2. Summary of Kleiman.

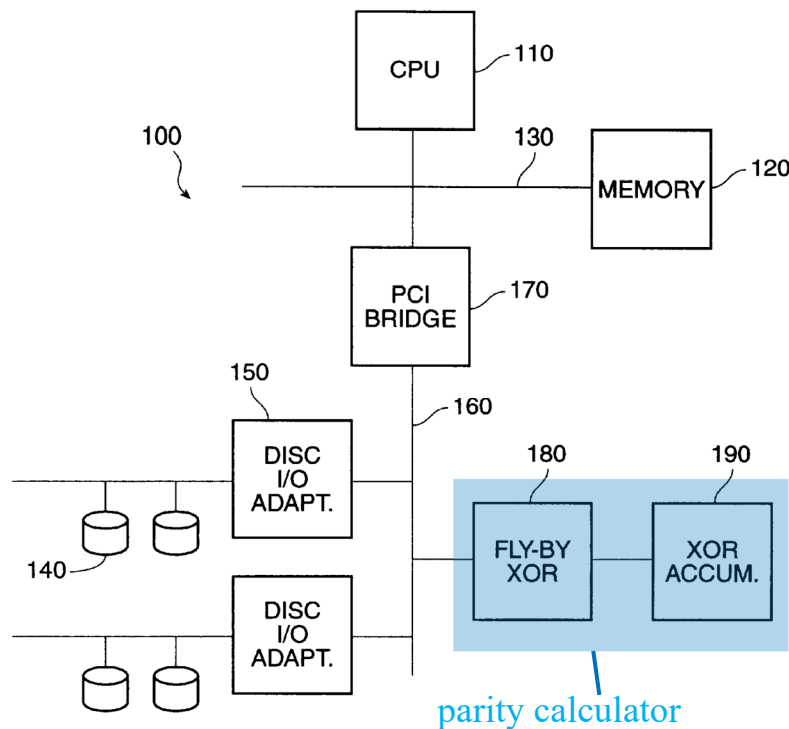
60. Kleiman provides evidence that it was known to perform parity calculations “on-the-fly.” Kleiman also provides evidence that it was known to perform several stripe operations in parallel, and to use identifiers to distinguish the different operations.

61. Like Jeddeloh and the ’346 patent, Kleiman relates to a “method and system for performing XOR operations without consuming substantial computing

resources.” Kleiman, 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* Kleiman abstract.

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

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



Kleiman, Fig. 1 (annotated).

63. 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.”

Kleiman 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.” Kleiman, 3:65-67. “[T]he 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.” Kleiman, 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.

64. Kleiman also describes performing several stripe operations in parallel. Kleiman’s “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.” Kleiman, 5:32-36. “Therefore, up to 32 different RAID stripes can have XOR accumulation performed simultaneously by the fly-by XOR element 180.” Kleiman, 5:36-38.

3. Reasons to Combine Jeddelloh and Kleiman.

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

circuitry.

66. 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.”

Kleiman 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.”

Kleiman 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.” Kleiman, 3:65-67.

67. A POSITA would have been motivated to utilize Kleiman’s on-the-fly technique because the on-the-fly technique “performs XOR operations on data transferred to and from the disk drives without requiring separate transfers.”

Kleiman, 1:67-2:2. Additionally, 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.” Kleiman 3:17-20.

68. 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.” Jeddeloh, [0028]. Then, “The memory system, e.g., RAID XOR circuitry 226 ... can calculate parity data for the write data.” Jeddeloh, [0044].

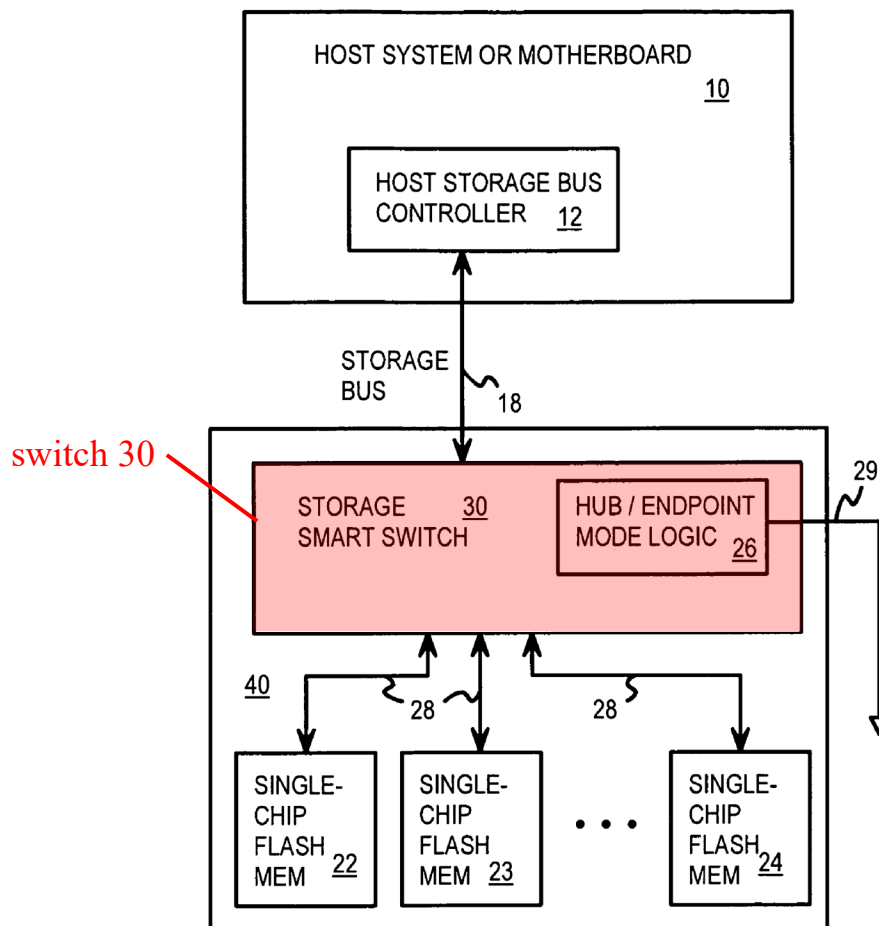
69. Kleiman likewise performs parity calculation for a RAID scheme. Kleiman explains that “When a RAID stripe is read or written, either entirely or partially, the fly-by XOR element 180 thus dynamically computes a set of parity information for the RAID stripe.” Kleiman, 3:14-17. Accordingly, a POSITA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

70. 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.*, Solomon, abstract (“A high performance data path for performing XOR on the fly.”); Stolowitz, 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.’”).

71. 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).

4. Summary of Ma.

72. Ma provides additional details as to the processing of write commands. Ma, like Jeddeloh and the '346 patent, “relates to flash-memory solid-state-drive (SSD) devices.” Ma, 1:52-53. In particular, Ma describes “a smart storage switch connecting to multiple single-chip flash-memory-device endpoints.” Ma, 1:53-55. As shown in Fig. 1 below, Ma describes a smart storage switch 30 that connects a host to multiple flash memory chips 22, 23, 24.



Ma, Fig. 1 (annotated).

73. Fig. 3, which illustrates the smart switch 30 and flash memory devices

FIG. 3

HOST MOTHERBOARD 10

STORAGE BUS I/F 18

STORAGE SMART SWITCH 30

UPSTREAM INTERFACE 34

LED CTR 33

SDRAM BUFFER 60

SDRAM_CLK CLK SRC 62

VIRTUAL BUFFER BRIDGE 32

SMART STORAGE TRANSACTION MANAGER 36

VIRTUAL STORAGE PROCES'R 38

VIRTUAL STORAGE BRIDGE 42

VIRTUAL STORAGE BRIDGE 43

CLK_SRC

LBA

28

38

CLK SRC 46

CK_SR

PBA

NV MEM 68

NV MEM 68

SINGLE-CHIP FLASH MEM 73

NVM CTR 76

memory controllers 76

switch 30

Ma, Fig. 3 (annotated).

33

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.” Ma, 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.” Ma, 11:51-54.

75. Accordingly, Ma provides additional details related to DMA operations of a switch connected to multiple memory controllers.

5. Reasons to Combine Ma with Jeddeloh and Kleiman.

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

77. 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.” Jeddeloh, [0033]. Jeddeloh further explains that “the switch 220 can be a direct memory access (DMA) module.” Jeddeloh, [0033].

78. 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.” Jeddelloh, [0033]. Accordingly, Jeddelloh’s channel memory controller receive write data through the switch 220. Jeddelloh’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.” Jeddelloh, [0040]. Jeddelloh 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.” Jeddelloh, [0040].

79. Given that Jeddelloh’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. 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 Jeddelloh and the ’346 patent, Ma “relates to flash-memory solid-state-drive (SSD) devices.” Ma, 1:52-53.

80. 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.” Ma, 11:42-44. 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.” Ma, 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.” Ma, 11:51-54.

81. 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. 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.

82. 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.*, Jeddelloh, [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.”); Ma, 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 Jeddelloh’s system.

83. 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).

6. Claim 1

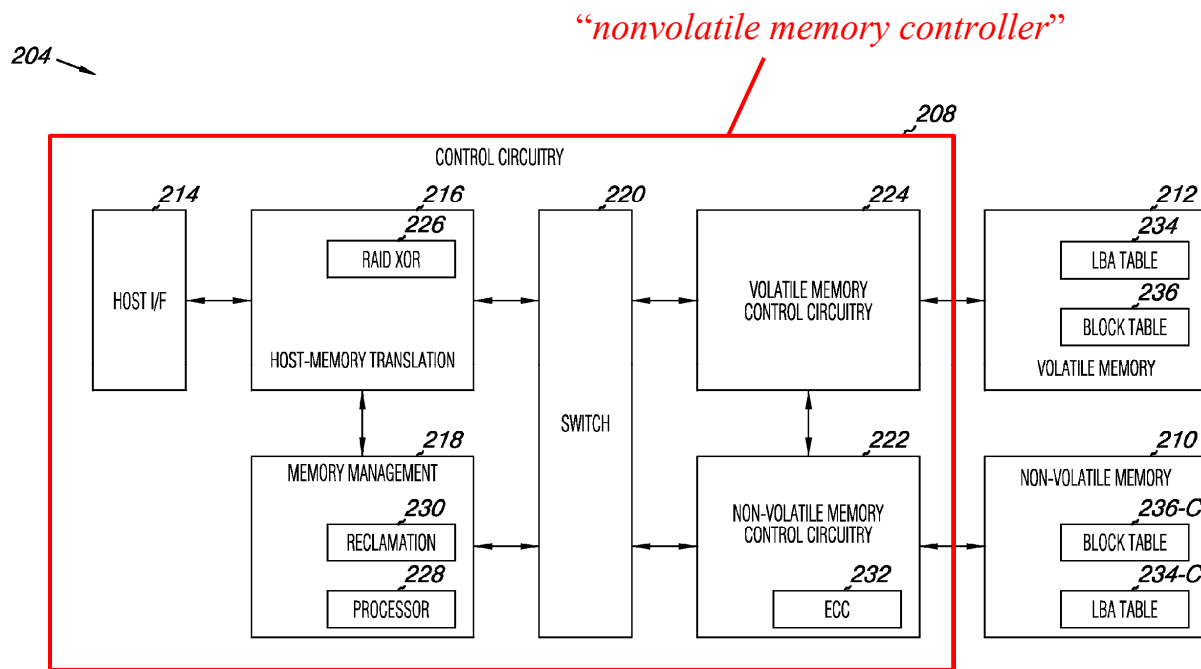
[1.0] *A nonvolatile memory controller for performing a data stripe operation on a plurality of data blocks, the nonvolatile memory controller comprising:*

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

85. **First**, Jeddeloh describes memory system control circuitry 208 ("*nonvolatile memory controller*") for controlling non-volatile memory.

FIG. 2 is a functional block diagram of a memory system 204 in accordance with one or more embodiments of the present disclosure. 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** and/or volatile memory 212. Memory system 204 and memory system control circuitry 208 can be analogous to memory system 104 and memory system control circuitry 108 respectively, illustrated in FIG. 1.

Jeddeloh, [0023].



Jeddeloh, Fig. 2 (annotated).

86. Second, Jeddeloh’s memory system control circuitry performs a stripe-based memory operation (“*data stripe operation*”). See Jeddeloh, 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**. A portion of the first stripe is updated by writing updated data in a portion of a second stripe across the storage volume of the plurality of memory devices. The portion of the first stripe is invalidated. The invalid portion of the first stripe and a remainder of the first stripe are maintained until the first stripe is reclaimed.

Jeddeloh, [0012].

87. 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. 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. A page write operation can allow access to a page of memory. 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. 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. Data from a page write operation can be transferred from a write buffer to the page in the solid state memory device 310-1, . . . , 310-N by the channel memory controller 338-1, . . . , 338-N. Write data for a stripe write operation can be received from a host system, e.g., host system 102 in FIG. 1.

Jeddeloh, [0040].

88. 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. 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** 580-1, 580-2, . . . , 580-S. The indicator "S" is used to indicate that a page can include a number of sectors. Each sector 580-1, 580-2, . . . , 580-S can store system and/or user data and can include overhead information, such as error correction code (ECC) information, and **logical block address** (LBA) information. **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. As an example, a sector of data can be a number of bytes of data, e.g., 256 bytes, 512 bytes, or 1,024 bytes. For example, an SSD can have 4, 8, or 16 sectors in a page, where a sector can be 512 bytes, and an SSD can have 128, 256, or 512 pages per physical block, therefore physical block sizes are 131072 bytes, 262144 bytes, and 524288 bytes. Embodiments are not limited to these examples.

Jeddeloh, [0062].

89. 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.*"

90. As another example, Jeddeloh explains that its memory pages can be

organized into a larger structure referred to as a block.

An SSD can include a number of memory devices, e.g., a number of memory chips (as used herein, “a number of” something can refer to one or more of such things, e.g., a number of memory devices can refer to one or more memory devices). As one of ordinary skill in the art will appreciate, a memory chip can include a number of dies and/or logical units (LUNs). Each die can include a number of memory arrays and peripheral circuitry thereon. **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.**

Jeddeloh, [0005].

91. 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.

92. 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 obvious a “*nonvolatile memory controller for performing a data stripe operation on a plurality of data blocks.*”

[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,*

93. 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"*).

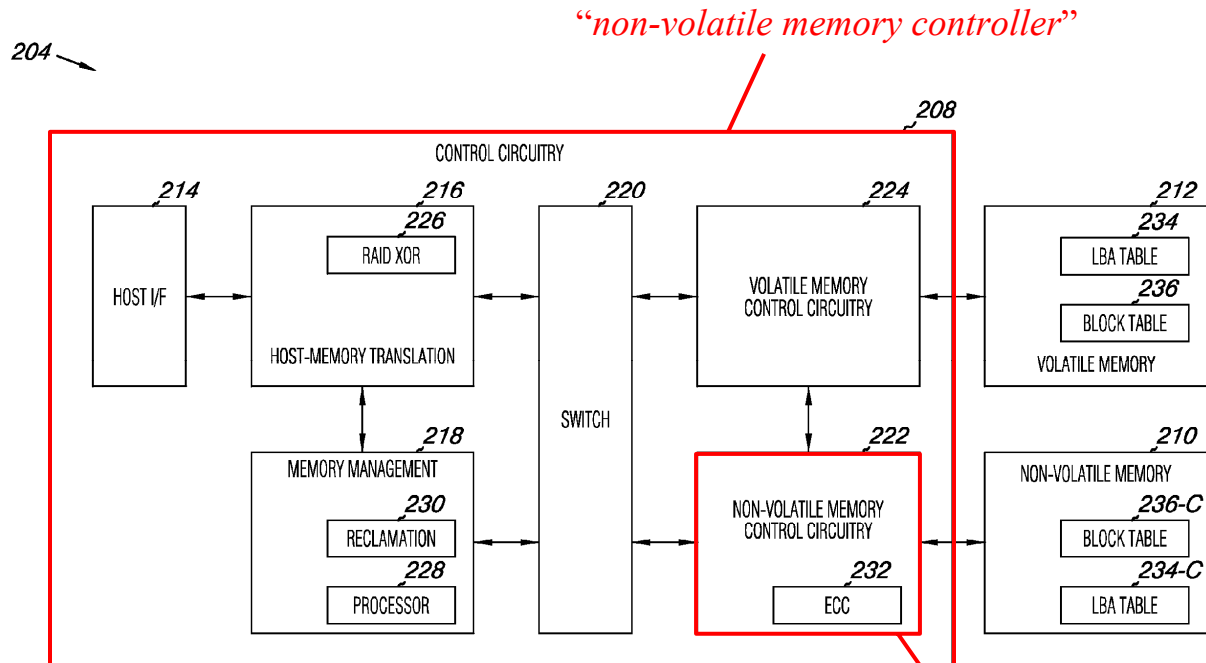
The memory system control circuitry 208 can include host interface circuitry 214, host-memory translation circuitry 216, memory management circuitry 218, a switch 220, **non-volatile memory control circuitry 222**, and/or volatile memory control circuitry 224. As described herein, the memory system control circuitry 208 can be provided in the form of an ASIC, however, embodiments are not so limited.

Jeddeloh, [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. The non-volatile memory control circuitry 222 can be configured to write updated data in a second stripe across the number of memory channels without reading or copying the data in the first stripe. In one or more embodiments, the non-volatile memory control circuitry 222 can be configured to read portions of a stripe on a page-basis without reading a remainder of the stripe. 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. 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).

Additional detail on stripe-based memory operation is provided in conjunction with the discussion of FIG. 4 below and throughout the present disclosure.

Jeddeloh, [0036].



Jeddeloh, Fig. 2 (annotated).

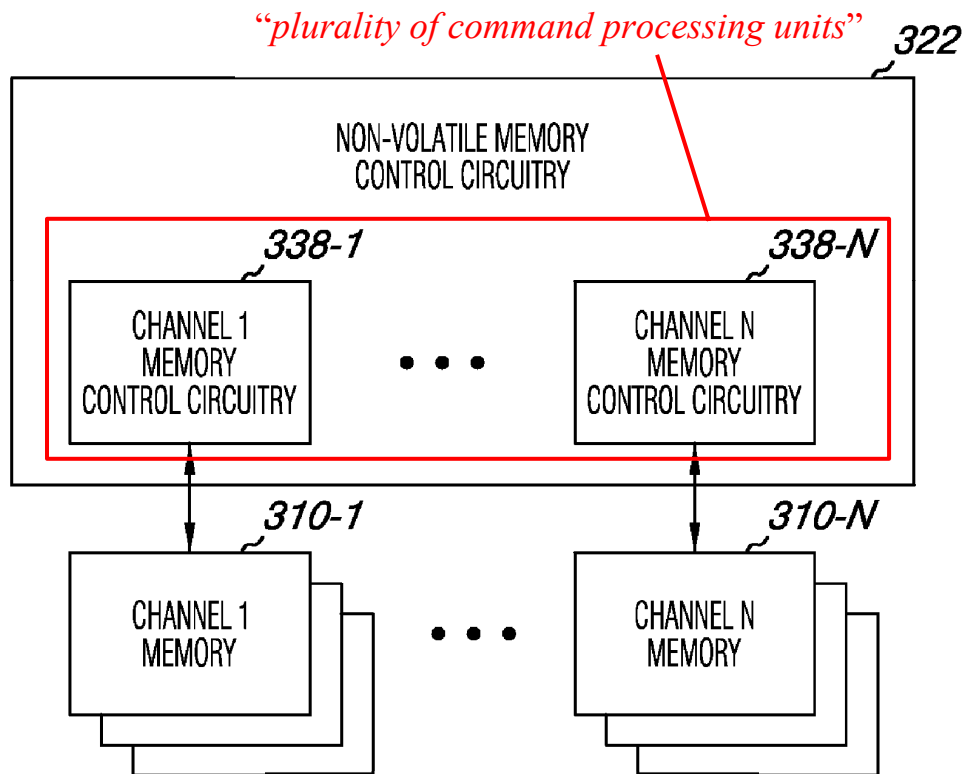
non-volatile memory
control circuitry

94. 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”*).

FIG. 3 is a functional block diagram of non-volatile memory control circuitry 322 in accordance with one or more embodiments of the present disclosure. The non-volatile memory control circuitry 322 can be analogous to the non-volatile memory control circuitry 222

illustrated in FIG. 2. The non-volatile memory control circuitry 322 can be coupled to one or more solid state memory devices 310-1, . . . , 310-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 by a separate channel. Each channel can be associated with discrete channel memory control circuitry 338-1, . . . , 338-N.** However, embodiments are not so limited, as fewer, e.g., one, non-volatile memory control circuit 322 can provide control to a plurality of channels in one or more embodiments. Non-volatile memory control circuitry can perform **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.**

Jeddeloh, [0038].



Jeddeloh, Fig. 3 (annotated).

95. Jeddeloh’s channel memory control circuits 338 are command processing units because they process various operations “including page read, stripe write, and stripe erase.” Jeddeloh, [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. 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. A page write operation can allow access to a page of memory. **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. 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. Data from a page write operation can be transferred from a write buffer to the page in the solid state memory device 310-1, . . . , 310-N by the channel memory controller 338-1, . . . , 338-N. Write data for a stripe write operation can be received from a host system, e.g., host system 102 in FIG. 1.

Jeddeloh, [0040].

96. 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.** A page write operation can allow

access to a page of memory. 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. 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. Data from a page write operation can be transferred from a write buffer to the page in the solid state memory device 310-1, . . . , 310-N by the channel memory controller 338-1, . . . , 338-N. Write data for a stripe write operation can be received from a host system, e.g., host system 102 in FIG. 1.

Write data can be striped across a storage volume of a plurality of solid state memory devices 310-1, . . . , 310-N. For example, a first page of write data received from a host can be written to a first memory device, a second page to a second memory device, etc. In another example embodiment, a first fragment, e.g., a sector, of a first page of write data received from a host can be written to a first memory device, a second fragment of the first page to a second device, etc. Although data may be written in a stripe-based operation, data may still be read in a page-based operation.

Jeddeloh, [0040]-[0041].

97. 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 obvious "*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” as recited in the claim.

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

98. 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, write data, and a number of LBAs associated with the write data. Write data associated with the number of LBAs can be written in a first stripe 444-1. Subsequently, **the 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.** In some instances, the subsequent write command may be accompanied by unrelated data, e.g., data associated with LBAs for which associated data was not written in the first stripe 444-1, in addition to the updated data. In some instances, the subsequent write command including updated data can be combined with another subsequent write command that includes unrelated data. In one or more embodiments, a portion of the first stripe 444-1 can be updated by writing updated data in a portion of second stripe 444-2 that is written across the storage volume 440. The memory system can write the updated data, and concurrently write other unrelated data, in a second stripe 444-2. Writing updated data and unrelated data in the

second stripe 444-2 increases a utilization of the storage volume of the second stripe 444-2 as compared to writing updated data alone in the second stripe 444-2. “Unrelated data” means data not associated with LBAs for which associated data was written in the first stripe. In association with writing the second stripe 444-2, the memory system can update an LBA table, e.g., LBA table 234 in FIG. 2, with the location of the updated data for the particular LBA. The memory system can update a block table, e.g., block table 236 in FIG. 2, to indicate that the portion of the first stripe 444-1 associated with the data that was updated is now invalid.

Jeddeloh, [0050]-[0051].

99. 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. Jeddeloh, [0050].

100. 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. A memory channel, a memory device, a block of memory, a page of memory, or another portion of a memory device could experience an uncorrectable error or other error. For example, write data can be written in the first stripe 444-1 across channels 442-1, . . . , 442-(N-1). Parity data can be written in the first stripe 444-1 across channel 442-N. If the first channel 442-1

experiences an uncorrectable error or other error, write data associated with channels up to 442-(N-1) and parity data associated with channel 442-N can be used to recreate the data written across the first channel 442-1. Similar to the case of creating the parity data, the memory system, e.g., RAID XOR circuitry 226 in FIG. 2, can use parity data along with write data to calculate replacement data for data associated with a portion of the memory system that experiences an uncorrectable error or other error.

The ECC circuitry 232 associated with the non-volatile memory control circuitry 222 in FIG. 2, can calculate error correction data 448 for fragments of the write data 446 and/or the parity data 450 to be stored in the memory devices.

FIG. 4 includes an illustration of write data 446 and error correction data 448 being stored in association with channels 442-1, . . . , 442-(N-1) and parity data 450 and error correction data 448 being stored in association with channel 442-N. When a stripe is written, both write data and error correction data can be written together. If an error is detected in data, whether write data or parity data, error correction data can be used in attempt to correct the error in a first level of error correction before and/or without implementing a second level of error correction, e.g., the correction associated with parity data described above. A memory system may store sufficient error correction data to enable a threshold number of erroneous bits to be corrected. If an error is detected, e.g., a UECC error during a read operation, that includes more bit errors than are correctable with the error correction data, then the second level of error correction, e.g., using the parity data described above, may be

implemented, e.g., the data associated with the error(s) may be recreated using parity data and a remainder of the write data.

Data can be written in a first stripe 444-1. Writing data in a stripe can include splitting write data into a number of fragments, e.g., pages, sectors, etc., and writing the fragments to memory devices associated with each channel 442-1, . . . , 442-(N-1), 442-N. One or more embodiments can include writing at least one fragment to each of the plurality of memory devices, e.g., to each channel 442-1, . . . , 442-(N-1), 442-N. **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, e.g., channels 442-1, . . . , 442-(N-1), and parity data can be written to one or more memory devices associated with a second subset of the channels, e.g., channel 442-N.**

Jeddeloh, [0047]-[0049]; *see also* Jeddeloh, [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”).

101. Jeddeloh explains that one memory channel and its associated memory device may be dedicated to parity data.

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. **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)**.

Jeddeloh, [0036].

102. Accordingly, a write command processed by the channel memory controller 338 for the dedicated parity device is a “*parity write command*” as claimed.

103. 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.*, Hodges, 3:3-5 (“The current proposed ANSI standard for “XOR commands” on SCSI disk drives includes read (READ), data update (XDWRITE), parity update (XPWRITE)”). Accordingly, it would have been obvious to use a parity update command, such as XPWRITE, when writing parity data to a dedicated parity channel.

104. Thus, because Jeddeloh describes that the commands processed by the memory channel circuits include data update commands and parity write commands, Jeddeloh renders obvious “*the plurality of commands including a plurality of data update commands and a parity write command*” as claimed.

[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,

105. 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*”).

106. 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. 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. Data from a page write operation can be transferred from a write buffer to the page in the solid state memory device 310-1, . . . , 310-N by the channel memory controller 338-1, . . . , 338-N. **Write data for a stripe write operation can be received from a host system, e.g., host system 102 in FIG. 1.**

Jeddeloh, [0040].

107. As explained above at VIII.A.5, it would have been obvious for

Jeddeloh's memory control circuitry 208 to utilize DMA operations. *See e.g.*, Jeddeloh, [0033]. 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. This is evidenced by Ma:

Single-chip flash-memory device 73 at endpoint-1 responds by setting up for the direct-memory access (DMA) transfer, and sends a DMA activate to smart storage switch 30, **which then sends a DMA activate back to host 10 to indicate that it is ready for the DMA transfer to begin.**

Ma, 11:40-44.

108. Thus, 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* VIII.A.5.

109. 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 obvious *"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” as claimed.

[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*

110. First, Jeddeloh explains that the channel memory controllers 338 write the data blocks to non-volatile memory.

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. 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. Data from a page write operation can be transferred from a write buffer to the page in the solid state memory device 310-1, . . . , 310-N by the channel memory controller 338-1, . . . , 338-N. Write data for a stripe write operation can be received from a host system, e.g., host system 102 in FIG. 1. Write data can be striped across a storage volume of a plurality of solid state memory devices 310-1, . . . , 310-N. For example, a first page of write data received from a host can be written to a first memory device, a second page to a second memory device, etc. In another example embodiment, a first fragment, e.g., a sector, of a first page of write data received from a host can be written to a first memory device, a second fragment of the first page to a second device, etc.

Although data may be written in a stripe-based operation, data may still be read in a page-based operation.

Jeddeloh, [0040]-[0041].

111. 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 in response to the request. As explained by Ma:

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, with the data to write into flash memory blocks in first single-chip flash-memory device 73.** Smart storage switch 30 acts as the endpoint to host 10, and as the host to endpoint-1.

Ma, 11:45-50.

112. Accordingly, Jeddeloh's channel memory controllers—implementing Ma's DMA technique—would similarly receive the data-out packets that includes the data to write to memory (*“receive the data block in response to the request”*).

113. Second, Jeddeloh's channel memory controllers write the data to their respective memory devices *“in response”* to receiving that data from the DMA request. Jeddeloh explains that data that is written is *“received from the host.”* Jeddeloh, [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:

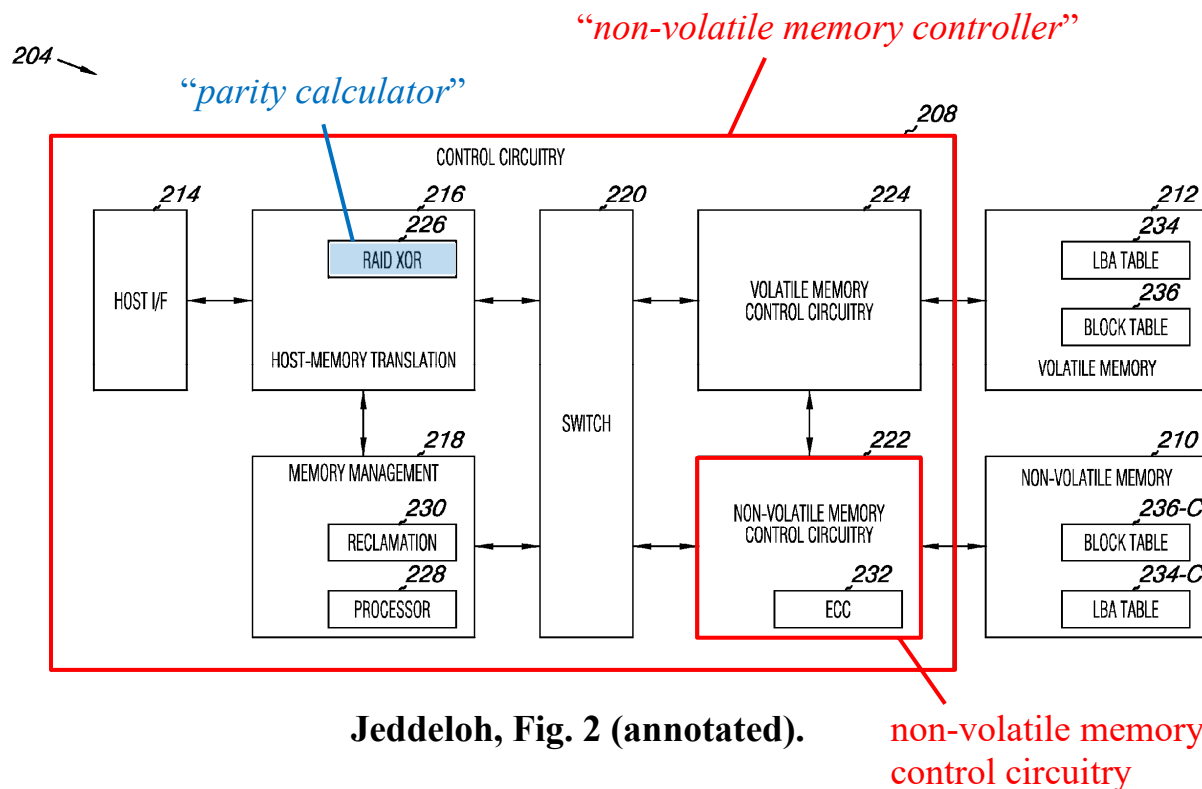
NVM controller 76 in first single-chip flash-memory device 73 receives these data packets and then writes this data into its flash memory blocks 68 and responds with a status packet back to smart storage switch 30. Smart storage switch 30 generates its own status packet to send to host 10.

Ma, 11:51-55.

114. Thus, because Jeddeloh'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, Jeddeloh and Ma render obvious "*[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*" as claimed.

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

115. First, Jeddeloh's memory control circuit includes RAID XOR circuitry 226 ("*a parity calculator*").



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. The host-memory translation circuitry 216 can be configured to translate host addresses to memory addresses, e.g., addresses associated with a received command such as a read and/or write command. 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. Each host operation can be translated into single or multi-sector non-volatile memory 210 operation. For example, host read and write commands can be translated into non-volatile memory 210 read and write commands. **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.

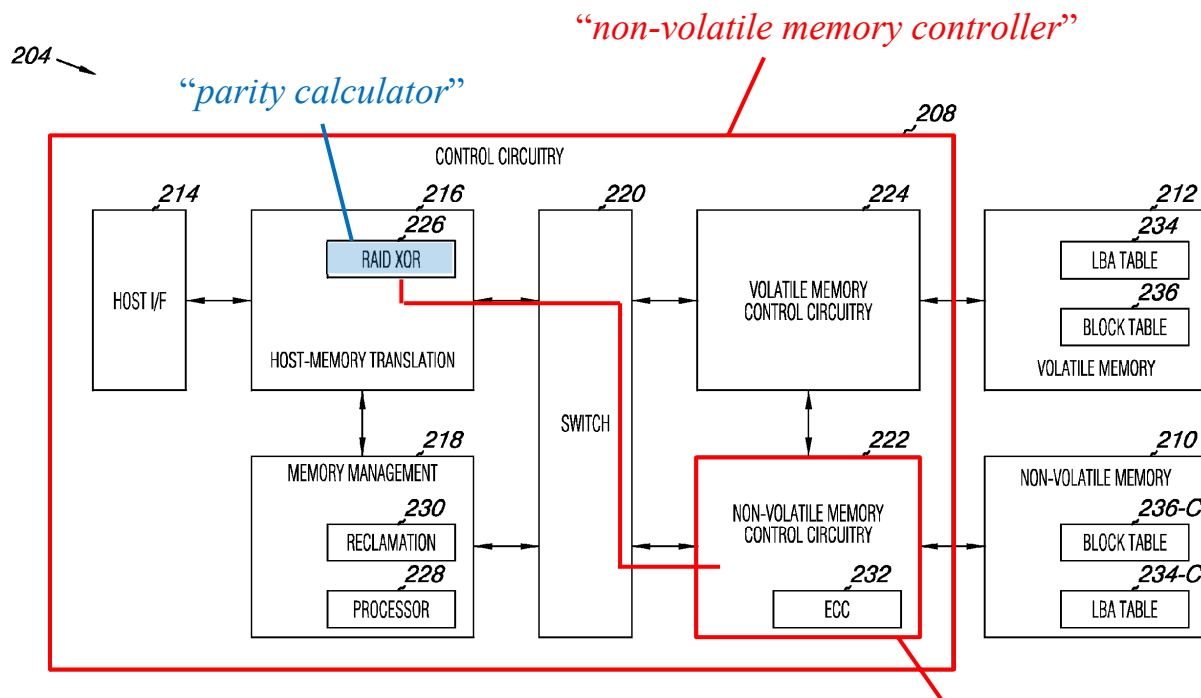
Jeddeloh, [0027]

A memory system can receive write data from a host system. **The memory system, e.g., RAID XOR circuitry 226 associated with host-memory translation circuitry 216 in FIG. 2, can calculate parity data for the write data.** In one or more embodiments, the write data and parity data can be striped across the storage volume 440 such that write data is written across a subset of the number of available channels, e.g., channels 442-1, . . . , 442-(N-1) and parity data can be written across a second subset of the number of available channels, e.g., channel 442-N. For example, 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, however embodiments are not so limited. However, embodiments are not limited to using N-1 of N channels for write data and a single channel for parity data. According to one or more embodiments of the present disclosure, RAID structure and/or stripe size can be programmable options.

Jeddeloh, [0044].

116. Second, as shown in Fig. 2 below, the RAID XOR circuitry (“*parity calculator*”) is coupled to the channel memory control circuits (“*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].



Jeddeloh, Fig. 2 (annotated).

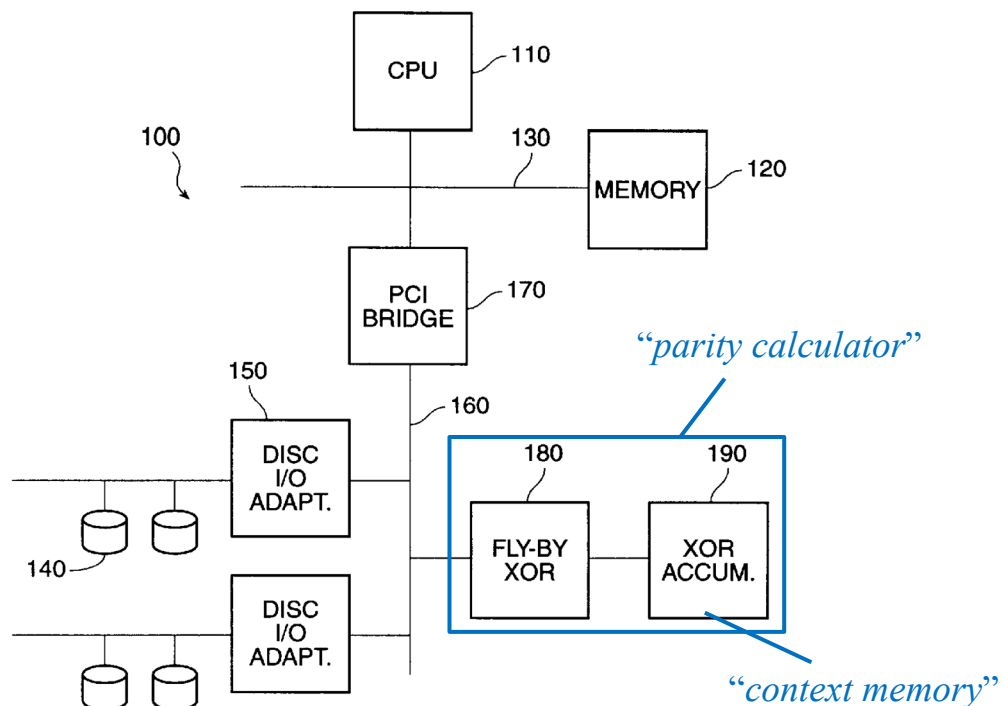
non-volatile memory
control circuitry

117. 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, Jeddeloh renders obvious “a parity calculator coupled to the plurality of command processing units” as claimed.

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

118. For the reasons explained above at VIII.A.2, it would have been obvious for Jeddeloh's RAID XOR circuitry 226 to be implemented according to Kleiman's on-the-fly technique.

119. First, Jeddeloh's parity calculator—implemented according to Kleiman's techniques—includes a “*context memory*” as claimed. For the reasons explained above at VIII.A.3, a POSITA would have found it obvious to implement Jeddeloh's RAID XOR circuitry using on-the-fly techniques as taught by Kleiman. 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.



Kleiman, Fig. 1 (annotated).

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.

Kleiman, 3:1-5.

When the processor 110 directs the transfer of data between the memory 120 and one or more of the disk drives 140, the data is transferred using the system bus 130, the PCI bridge 170, and the mezzanine bus 160. 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. 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, without requiring additional resources from any part of the system 100. No additional data transfers are required, and the processor 110 and the memory 120 are not required to perform these XOR operations.

Kleiman, 3:6-22.

120. 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.

[T]he context memory 600 stores the data block contained in the completion packet into the page frame 425 associated with the context 435. The data block stored in the page frame 425 is an initial parity

block for the data stripe operation associated with the data block. Further, the parity calculator controller 605 modifies (e.g., updates) the context state 505 of the context 435 to the initialized state.

Ex.1001, 14:24-30.

121. 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 transmits a memory address of the memory buffer to the fly-by XOR element 180 and directs the fly-by XOR element 180 to recognize data transferred to or from that memory buffer. **The processor 110 also allocates one of the XOR accumulator buffers in the XOR buffer 190 to accumulate the results of the XOR operation.**

Kleiman, 3:49-56.

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. **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.** When all data for the recognized memory buffer has been transferred, the associated XOR accumulator buffer holds the computed parity for the RAID stripe.

Kleiman, 3:57-67.

122. Accordingly, because the XOR buffer stores the updated parity block

throughout the stripe operation, it is a “*context memory*” as claimed.

123. Second, Kleiman’s XOR buffer “*context memory*” includes at least one XOR accumulator buffer (“*page frame*”). Kleiman explains that “The 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.” Kleiman, 5:33-36.

124. 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* Jeddeloh, [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.*, IEEE, 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.

125. 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 obvious “*the parity calculator further comprising a context memory including a page frame*” as claimed.

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

126. First, Jeddeloh describes that the “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.” Jeddeloh, [0027].

127. Second, when using Kleiman’s on-the-fly technique, the data being received from the host interface circuitry is received in sequence.

The associated 5-bit value is used to select one of the 32 XOR accumulator buffers in the RAM 340, while bits 2 through 11 of the bus address path 312 are used to select an individual word in the XOR accumulator buffer in the RAM 340, so that the data values on the mezzanine bus 160 are read into the FIFO queue 350 word by word. The timing and control element 320 sequences each word in the FIFO queue 350, in conjunction with a corresponding word read from the RAM 340 into the FIFO register 353, 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. The timing and control element 320 then causes

the results of the XOR operation to be written back into the associated XOR accumulator buffer in the RAM 340.

Kleiman, 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**, so as to accumulate parity information in one of the 32 XOR accumulator buffers in the RAM 340

Kleiman, 5:39-44.

128. 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.

129. Thus, because Kleiman's fly-by XOR element receives data in sequence, Jeddeloh and Kleiman render obvious *"the parity calculator configured to receive the plurality of data blocks as a sequence of data blocks"* as claimed.

[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,

130. 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**." Kleiman, 3:62-64.

131. Kleiman also explains that before the parity block operation, the XOR accumulator is “reset to a uniform zero value.” Kleiman, 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.*”

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

133. **Second**, for each subsequent piece of data passing through, Kleiman’s fly-by XOR element replaces (“*update[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*”).

When the processor 110 directs the transfer of data between the memory 120 and one or more of the disk drives 140, the data is transferred using the system bus 130, the PCI bridge 170, and the mezzanine bus 160. **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.** 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, without requiring additional resources from any part of the system 100. No additional data transfers are required, and the processor 110 and the memory 120 are not required to perform these XOR operations.

Kleiman, 3:7-21.

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. **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.** When all data for the recognized memory buffer has been transferred, the associated XOR accumulator buffer holds the computed parity for the RAID stripe.

Kleiman, 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.

The data path control engine 44 includes a DMA/XOR engine 46 and an XOR buffer 48. 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. 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.** In performing a write operation to interrelated RAID

disks, when all data in a corresponding group has been XORed, the parity data is found in the XOR buffer.

Solomon, 4:22-41.

134. Using Kleiman's technique in Jeddeloh, the RAID XOR circuitry 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.

135. 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 renders obvious "[*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" as claimed.

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

136. Jeddeloh's RAID XOR circuitry, implemented with Kleiman's on-the-fly technique, does not have to store each data block in a buffer 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.*"

137. Kleiman's technique is consistent with the on-the-fly technique described in the '346 patent. As explained above at VII, 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.***"

The parity calculator receives the data blocks as a sequence of data blocks through the same data path from which the command processing units receive the data blocks. 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.

'346 Patent, 1:61-2:3.

138. 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 obvious "*without storing each data block in a data buffer*" as claimed.

[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.

139. 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].

According to one or more embodiments of the present disclosure, write data can be striped across multiple solid state memory devices in a RAID operation. As one of ordinary skill in the art will appreciate, striping includes splitting data so that it is stored on more than one device. The portions of the more than one device that store the split data are collectively referred to as a stripe. In contrast, mirroring can include storing duplicate copies of data on more than one device. Striping is described in more detail in connection with FIG. 4. **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.** For each bit written to the seven of eight memory devices, a parity bit can be calculated by the RAID XOR circuitry 226 and written to the eighth memory device, e.g., bit 0 from each of the seven memory devices can be used to generate bit 0 in the parity (eighth) memory device.

Jeddeloh, [0028].

140. The parity data is written to a specific memory channel and thus processed by one of the channel memory controllers (“*command processing units*”).

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. **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).** Additional detail on stripe-based memory operation is provided in conjunction with the discussion of FIG. 4 below and throughout the present disclosure.

Jeddeloh, [0036].

The ECC circuitry 232 associated with the non-volatile memory control circuitry 222 in FIG. 2, can calculate error correction data 448 for fragments of the write data 446 and/or the parity data 450 to be stored in the memory devices. FIG. 4 includes an illustration of write data 446 and error correction data 448 being stored in association with channels 442-1, . . . , 442-(N-1) and **parity data 450 and error correction data 448 being stored in association with channel 442-N.** When a stripe is written, both write data and error correction data can be written together. If an error is detected in data, whether write data or parity data, error correction data can be used in attempt to correct the error in a first level of error correction before and/or without implementing a second level of error correction, e.g., the correction associated with parity data described above. A memory system may store sufficient error correction data to enable a threshold number of erroneous bits to be corrected. If an error is detected, e.g., a UECC error during a read operation, that includes more bit errors than are correctable with the error correction data, then the second level of error correction, e.g.,

using the parity data described above, may be implemented, e.g., the data associated with the error(s) may be recreated using parity data and a remainder of the write data.

Jeddeloh, [0048].

141. Thus, because Jeddeloh describes using channel memory controllers to write parity data to memory, Jeddeloh renders obvious “*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*” as claimed.

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.*

142. 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.*, IEEE, 402.

143. Thus, because Kleiman’s on-the-fly technique uses a XOR operation, which is a Boolean logical operation, Jeddeloh and Kleiman render obvious

“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.”

8. Claim 3

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

144. 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 obvious “*wherein the logical operation includes an exclusive-or operation*” as claimed.

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,*

145. 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 can be coupled to the host interface circuitry 214, to the memory management circuitry 218, and/or to the switch 220. The host-memory translation circuitry 216 can

be configured to translate host addresses to memory addresses, e.g., addresses associated with a received command such as a read and/or write command. **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.** Each host operation can be translated into single or multi-sector non-volatile memory 210 operation. For example, host read and write commands can be translated into non-volatile memory 210 read and write commands. 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.

Jeddeloh, [0027].

146. 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.*”

147. 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.

The non-volatile memory control circuitry 222 can be configured to write updated data in a second stripe across the number of memory channels without reading or copying the data in the first stripe. In one or more embodiments, the non-volatile memory control circuitry 222 can be configured to read portions of a stripe on a page-basis without reading a remainder of the stripe. 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. 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). Additional detail on stripe-based memory operation is provided in conjunction with the discussion of FIG. 4 below and throughout the present disclosure.

Jeddeloh, [0036].

148. 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.*”

149. Accordingly, the host-memory translation circuitry 216 renders obvious a “*command distribution unit configured to distribute the plurality of data update commands to command processing units of the plurality of command processing units*” as claimed.

[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.*

150. 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.” Kleiman, 3:65-67.

At a step 224, 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. In either case, readout from the fly-by XOR element 180 is performed by processor reads or DMA transfer to one of the disk drives 140. In a preferred embodiment,

readout of the accumulated parity information causes the associated XOR accumulator buffer to be reset to zero.

Kleiman, 4:1-11.

151. 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.

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

153. 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 obvious a 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”* as claimed.

10. Claim 6

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

154. *See Claim 1.*

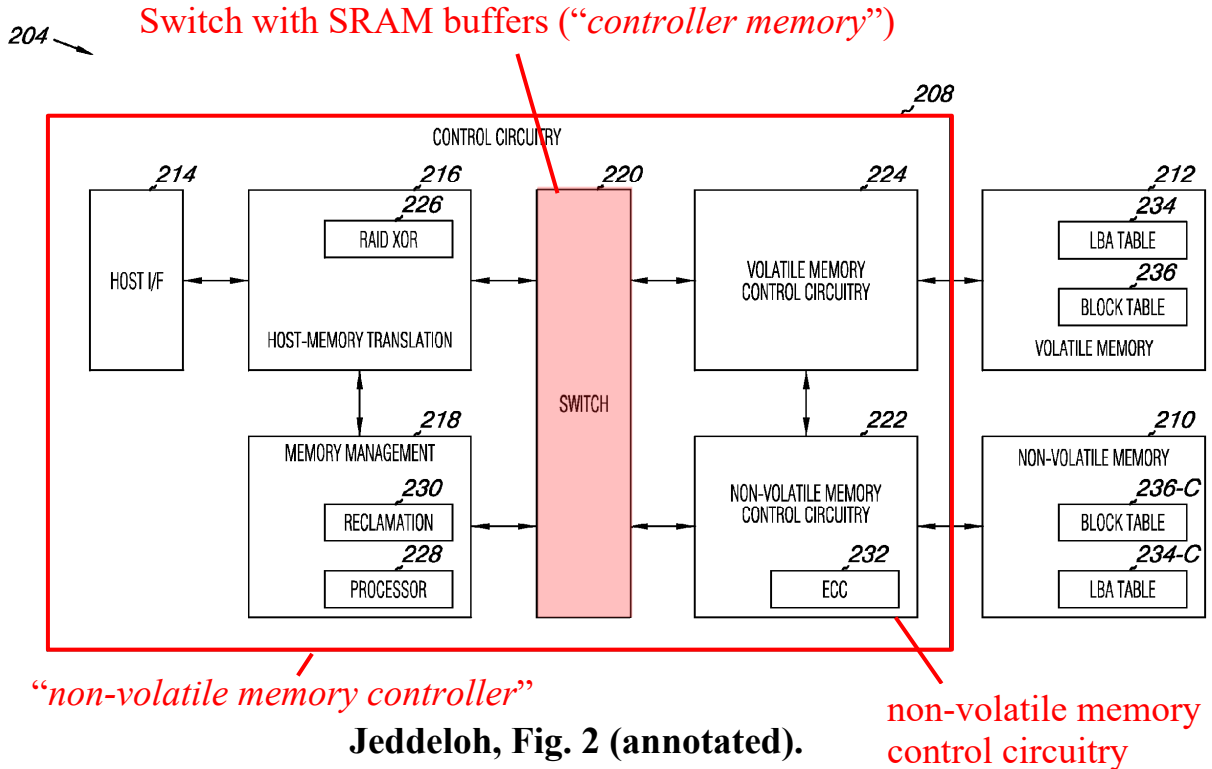
[6.1] *a controller memory configured to store the plurality of data blocks; and*

155. First, Jeddeloh's memory controller includes a switch 220 with

SRAM buffers (“*controller memory*”).

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. **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.** The switch 220 can provide an interface between various components of the memory system control circuitry 208. The switch 220 can account for variations in defined signaling protocols that may be associated with different components of the memory system control circuitry 208 in order to provide consistent access and implementation between components. In one or more embodiments, the switch 220 can be a direct memory access (DMA) module.

Jeddeloh, [0033].



156. **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**. Once the data is stored in the SDRAM buffer, smart storage switch 30 generates a status packet to host 10 indicating that the first DMA transfer completed.

Ma, 12:44-49.

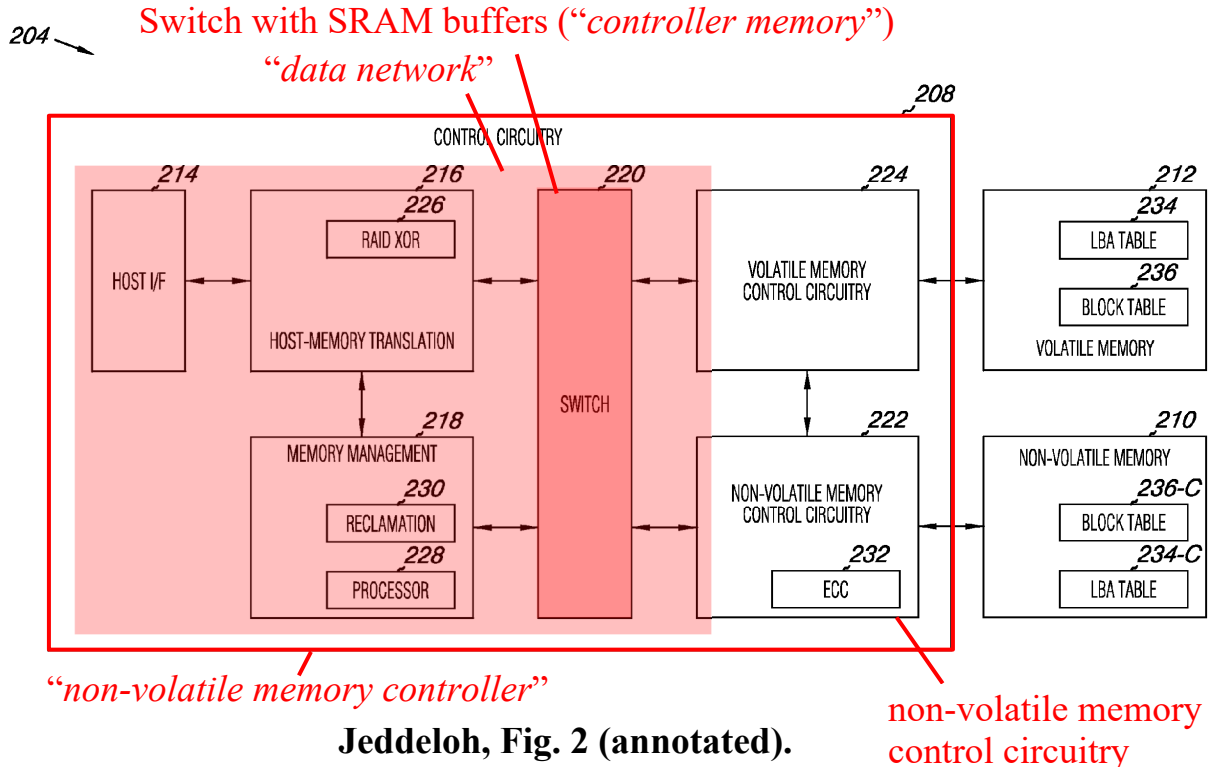
157. Accordingly, it would have been obvious for Jeddeloh's switch 220 to similarly buffer incoming write data ("*data blocks*") in its SRAM buffers.

158. 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 obvious "*a controller memory configured to store the plurality of data blocks*" as claimed.

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

159. 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."

Jeddeloh, [0033].



160. Second, the network is connected to the switch 220 that includes 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." Jeddeloh, [0033]. Accordingly, the *"data network [is] coupled to the controller memory."*

161. 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." Jeddeloh, [0034]. Accordingly, the *"data network [is]*

coupled to ... the plurality of command processing units” as claimed.

162. 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.” Jeddelloh, [0033]. According, the “*data network [is] coupled to the ... parity calculator*” as claimed.

163. Thus, because Jeddelloh’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, Jeddelloh renders obvious “*a data network coupled to the controller memory, the plurality of command processing units, and the parity calculator*” as claimed.

[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,

164. 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.

165. 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.” Ma, 12:43-46.

166. 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 obvious “*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*” as claimed.

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

167. 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.” Jeddeloh, [0033].

168. 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, with the data to write into flash memory blocks** in first single-chip flash-memory device 73. Smart

storage switch30 acts as the endpoint to host 10, and as the host to endpoint-1.

Ma, 11:45-50.

169. When Ma's technique is implemented in Jeddeloh, the endpoint-1 becomes the channel memory circuits 338. 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*").

170. 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 obvious "*the controller memory is further configured to generate a plurality of completion packets including the plurality of data blocks*" as claimed.

[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.

171. 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*").

172. 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. The host-memory translation circuitry 216 can be configured to translate host addresses to memory addresses, e.g., addresses associated with a received command such as a read and/or write command. 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. Each host operation can be translated into single or multi-sector non-volatile memory 210 operation. For example, host read and write commands can be translated into non-volatile memory 210 read and write commands. **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 databased on data received from the host interface circuitry 214.

According to one or more embodiments of the present disclosure, write data can be striped across multiple solid state memory devices in a RAID operation. As one of ordinary skill in the art will appreciate, striping includes splitting data so that it is stored on more than one device. The portions of the more than one device that store the split data are collectively referred to as a stripe. In contrast, mirroring can include storing duplicate copies of data on more than one device. Striping is described in more detail in connection with FIG. 4. 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. **For each bit written to the seven of eight memory devices, a parity bit can be calculated by the RAID XOR circuitry 226 and written to the eighth memory device**, e.g., bit 0 from each of the seven memory devices can be used to generate bit 0 in the parity (eighth) memory device.

Jeddeloh, [0027]-[0028].

173. 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.*”

174. 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.*”

175. Thus, because Jeddeloh’s RAID XOR circuitry and channel memory controllers receive write data over interconnections to the host interface, Jeddeloh renders obvious “*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” as claimed.

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,*

176. See [5.1].

[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.*

177. See [5.2].

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.*

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

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. A page write operation can allow access to a page of memory. **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. 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. Data from a page write operation can be transferred from a write buffer to the page in the solid state memory device 310-1, . . . , 310-N by the channel memory controller 338-1, . . . , 338-N. Write data for a stripe write operation can be received from a host system, e.g., host system 102 in FIG. 1.

Jeddeloh, [0040].

179. Thus, because Jeddeloh's channel memory controllers queue received commands, Jeddeloh renders obvious "*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*" as claimed.

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.*

180. 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*").

181. 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 as in the step 221 and the step 222, 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 (or a 20-bit memory address for a null memory buffer), and associates that 20-bit memory address with a 5-bit identifier for one of the 32 XOR accumulator buffers. Bits 2 through 9 of the bus address path 312 are coupled to the address input port 331 and provide the address of a new entry for the CAM 330. The CAM multiplexer 336, under control of the timing and control element 320, selects the bus data path 313 for coupling to the tag port 332. The CAM 330 inserts bits 12 through 31 of the bus data value from the bus data path 313 as a new 20-bit tag at the address specified at the address input port 331, and bits 0 through 4 of the bus data value from the bus data path 313 as a new 5-bit value. The processor 110 can also read a selected entry in the CAM 330 for diagnostic purposes.

Kleiman, 6:36-55.

182. 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.

183. 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.” Kleiman, 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.” Kleiman, 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. 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.

Kleiman, 6:56-64

184. Then, “[t]he associated 5-bit value is used to select one of the 32 XOR accumulator buffers in the RAM 340.” Kleiman, 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*”).

185. As explained above at VIII.A.3, it would have been obvious for

Jeddeloh's RAID XOR circuit 226 of Jeddeloh's 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*").

186. Thus, Jeddeloh's 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 obvious "*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*" as claimed.

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.*

187. Jeddeloh explains that data stripe operations are performed "simultaneously" ("*in parallel*").

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.** A page write operation can allow access to a page of memory. 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. 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. Data from a page write operation can be transferred from a write buffer to the page in the solid state memory device 310-1, . . . , 310-N by the channel memory controller 338-1, . . . , 338-N. Write data for a stripe write operation can be received from a host system, e.g., host system 102 in FIG. 1.

Jeddeloh, [0040].

188. Additionally, Kleiman describes:

The 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. **Therefore, up to 32 different RAID stripes can have XOR accumulation performed simultaneously by the fly-by XOR element 180.**

Kleiman, 5:32-37.

189. 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 obvious "*wherein the plurality of command processing units is further configured to perform a plurality of data stripe operations in parallel.*"

15. Claim 11

[11.0] *A nonvolatile memory controller comprising:*

190. For the reasons explained at [1.0], the prior art teaches a "non-volatile memory controller" as claimed.

[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;*

191. 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.

192. Second, as explained above at [1.2], the plurality of commands include "*a plurality of data update commands and a parity write command.*"

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

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

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

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

196. Thus, because Jeddeloh includes a plurality of channel memory controllers coupled to the host translation circuitry, Jeddeloh renders obvious “*a plurality of command processing units coupled to the command distribution unit*” as claimed.

[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,*

197. See [1.0].

[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*

198. For the reasons explained above at [1.3a] and [1.3b], Jeddeloh and Ma render obvious “*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” as claimed.

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

199. See [1.4].

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

200. See [1.5].

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

201. See [1.6].

[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,

202. 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.*”

203. 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.*”

[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,*

204. See [5.1].

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

205. See [1.9].

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.*

206. See [9.1].

17. Claim 14

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

207. 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.” Jeddeloh, [0012]. The data stripe operation uses “RAID XOR circuitry 226 [to] calculate parity databased on data received from the host interface circuitry 214.” Jeddeloh, [0027].

[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;

208. For the reasons explained above at [1.1], Jeddeloh renders obvious “*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.*”

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

209. For the reasons explained above at [1.3a], Jeddeloh and Ma render obvious “*generating a plurality of data requests by the plurality of command processing units based on the plurality of data update commands*” as claimed.

[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;*

210. 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.

211. 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.

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

212. For the reasons explained above at [1.3b], Jeddeloh and Ma render obvious “writing the plurality of data blocks to a plurality of nonvolatile memory devices by the plurality of command processing units.”

[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;*

213. For the reasons explained above at [1.4]-[1.8], Jeddeloh renders obvious “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” as claimed.

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

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

[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.*

215. 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.

18. Claim 15

[15.1] *The method of claim 14, wherein updating the data block stored in the page frame with each data block following the first data block in the sequence of data blocks comprises performing an 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.*

216. See [2.1].

19. Claim 16

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

217. See [3.1].

20. Claim 18

[18.1] *The method of claim 14, wherein the context memory includes a plurality of page frames and each command of the plurality of commands includes a flow identifier, the method further comprising allocating a page frame in the context memory to the data stripe operation by associating the flow identifier with the page frame.*

218. See [9.1].

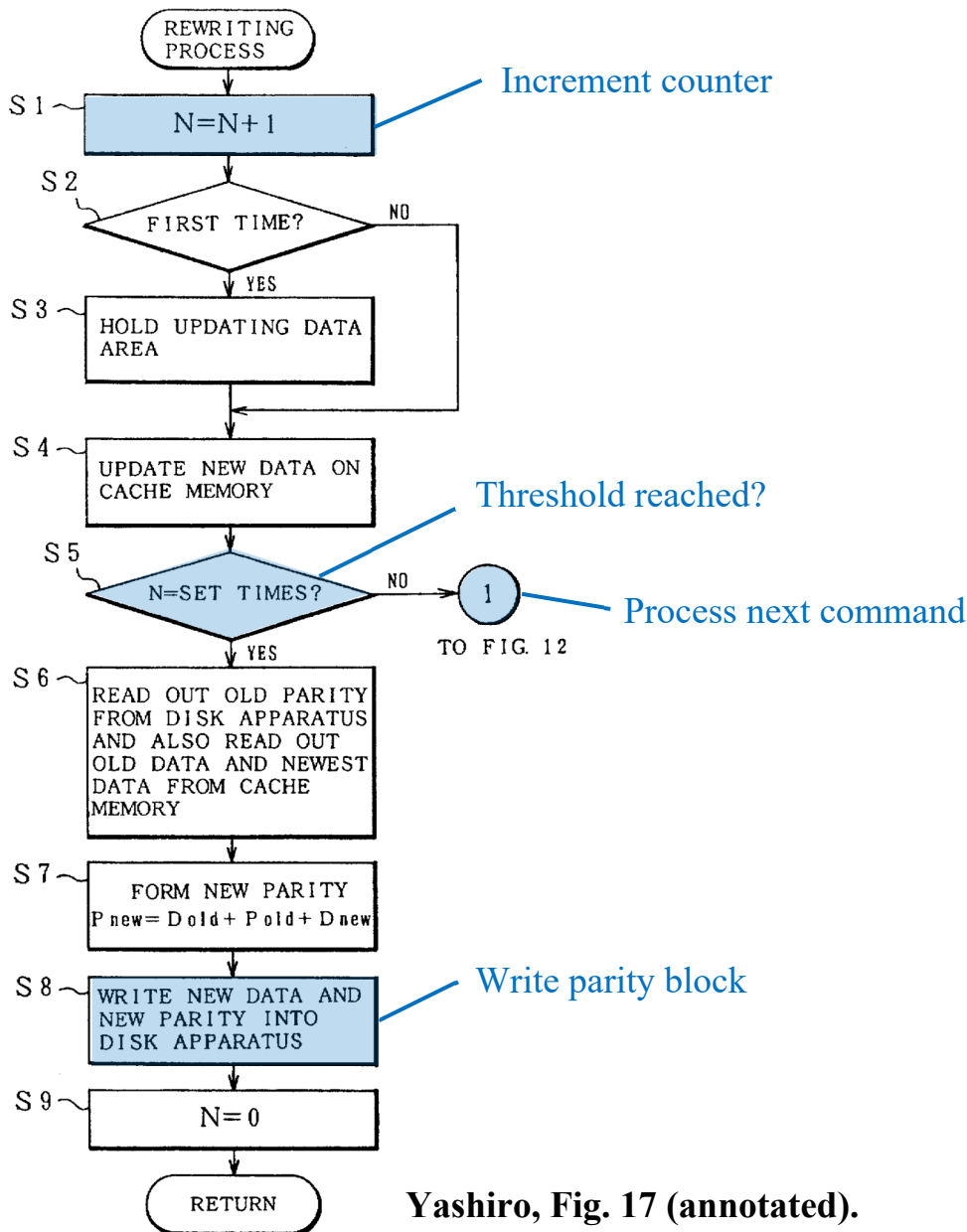
B. Ground 2: Claims 4 and 17 are obvious in view of Jeddelloh, Kleiman, Ma, and Yashiro.

1. Summary of Yashiro

219. Like Kleiman, Yashiro describes calculating parity data using the XOR (exclusive OR) operation. See Yashiro, abstract; 3:61-65. 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.” Yashiro, 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.” Yashiro, 14:30-31.

220. 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. This process is illustrated below in Fig. 17. Yashiro, 14:28-55.



221. Accordingly, Yashiro provides evidence that it was known to use a counter to determine whether a parity block is complete.

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

222. A POSITA would have found it obvious to use a counter for

determining that the parity block of the Jeddeloh/Kleiman combination is complete.

223. As explained above at VIII.A.2, Kleiman describes a known method in which a parity block is calculated on-the-fly. Kleiman, 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.” 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.” Kleiman, 3:65-4:6.

224. 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. 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.” Yashiro, 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.” Yashiro, 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* Yashiro, 14:28-55.

225. A POSITA would have been motivated to use Yashiro's counter technique because it provides a tried-and-true technique for determining when the parity block is complete. 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.

226. 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. Kleiman uses "XOR accumulator buffers in the XOR buffer 190 to accumulate the results of the XOR operation." Kleiman, 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." Yashiro, 4:22-25.

227. 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).

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.*

228. 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. 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." Kleiman, 3:3-5.

229. Second, it would have been obvious for Kleiman's on-the-fly XOR technique to maintain a stripe count as part of the context of the operation. Kleiman explains that the system updates the XOR buffer with each passing data sector. "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." Kleiman, 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.” Kleiman, 3:65-4:3.

230. One known way to determine whether each of the sectors has passed through would be to maintain a counter. *See generally*, Yashiro, 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.

Yashiro, 14:28-55.

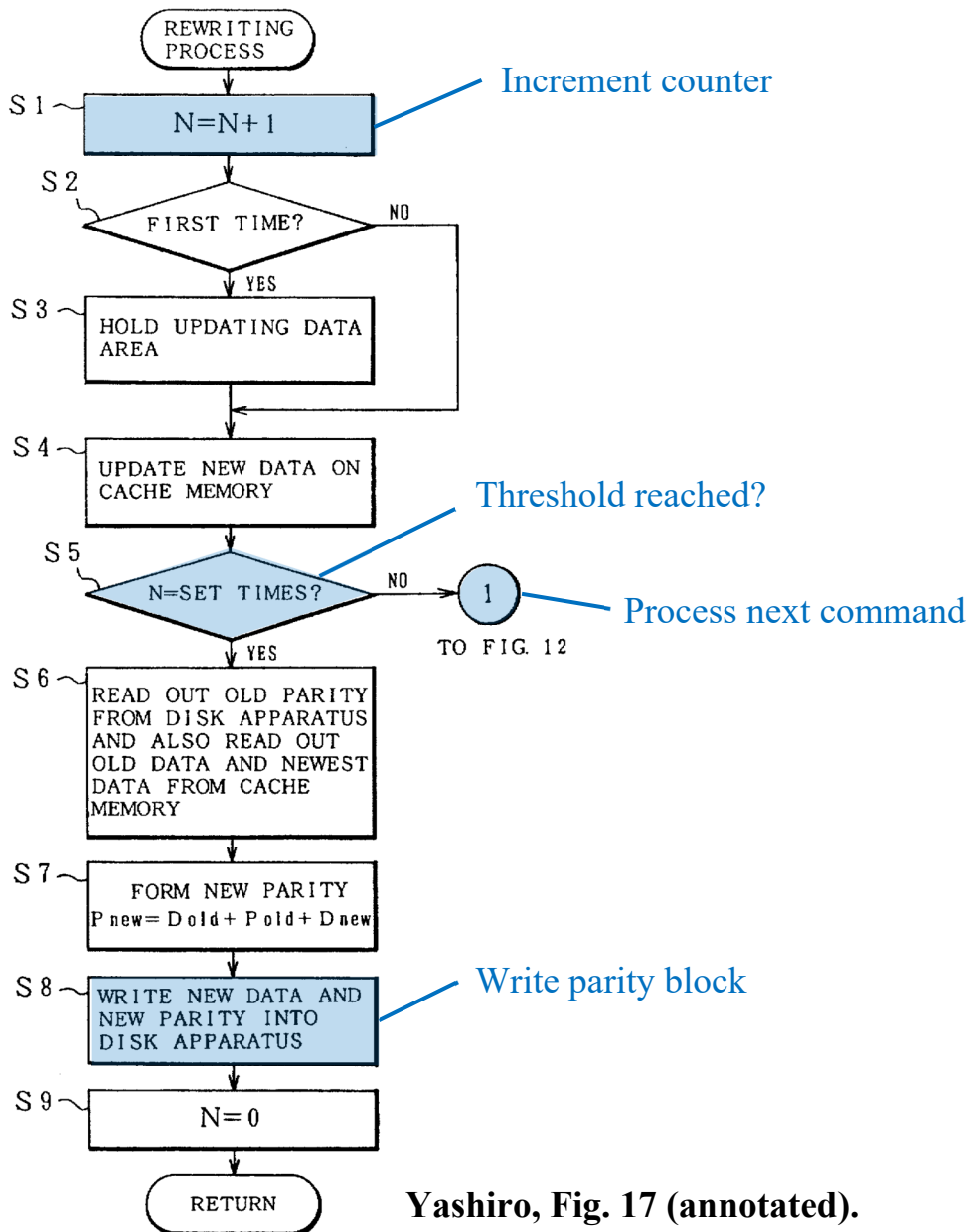


FIG. 17 is a flowchart showing the details of the rewriting process in step S8 in FIG. 15. In FIG. 17, first, in step S1, the count value of the counter N to set the number of updating times is increased by 1. The counter N has been cleared to $N=0$ in the initial state. In step S2, a check is made to see if the number of the updating time is the first time or not. If NO, the area of the updating data is newly assured in the cache

memory 26 in step S3. The new data is updated for the first time in the area newly assured in the cache memory 26 in step S4. In step S5, a check is made to see if the counter N indicative of the number of updating times has reached a set number of times, for example, three times or not. If it is less than 3, the processes in steps S6 to S9 are skipped and the processing routine is returned to (1) in the main routine in FIG. 12. When the counter N indicative of the number of updating times has reached the set number of times in step S5, step S6 follows and the old parity before updating of the same segment as that of the updating data is read out from the disk apparatus for parities, and the old data before updating and the newest updated data are read out from the cache memory 26. In step S7, the new parity is formed in accordance with the equation (7). Subsequently, in step 8, the new data and the new parity are respectively written into the corresponding disk apparatuses. After that, the counter N is reset in step S9. The processing routine is returned to step S7 in FIG. 15. The newest updating data after it was rewritten is stored into the cache memory 26 and the apparatus waits for the next access.

Yashiro, 14:28-56.

231. For the reasons explained above at VIII.B.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.

232. 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 obvious "*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*" as claimed.

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.*

233. 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* Cota-Robles 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”).

234. 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.

235. 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 obvious “*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*” as claimed.

IX. CONCLUSION

236. This declaration and my opinions herein are made to the best of my knowledge and understanding, and based on the material available to me, at the time of signing this declaration. I declare that all statements made herein on my own knowledge are true and that all statements made on information and belief are believed to be true, and further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 or Title 18 of the United States Code.

Date: January 21, 2025

Paul Min