

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

ADVANCED MICRO DEVICES, INC. and PENSANDO SYSTEMS, INC.,
Petitioners,

v.

CONCURRENT VENTURES, LLC and XTREAMEDGE, INC.,
Patent Owners.

IPR2025-00478

U.S. Patent No. 8,924,596

PETITION FOR *INTER PARTES* REVIEW

OF U.S. PATENT NO. 8,924,596

TABLE OF CONTENTS

	<u>Page</u>
I. BACKGROUND	3
II. THE '596 PATENT	5
A. Prosecution History	6
B. PO's Allegations.....	7
C. Priority Date	8
III. FIELD AND LEVEL OF SKILL	8
IV. CLAIM CONSTRUCTION	8
A. “[input/output] queue implemented in hardware”	8
V. INVALIDITY GROUNDS.....	9
A. Ground 1	10
1. U.S. Patent No. 8,559,439 (“Dongare”)	10
2. U.S. Patent No. 8,051,227 (“Gewirtz”)	11
3. Claim 1	12
4. Claims 2, 8, 14	35
5. Claims 5, 11, 17	36
6. Claims 6, 12, 18	37
7. Claim 7	37
8. Claim 13	39
B. Ground 2.....	40
1. U.S. Patent No. 6,810,462 (“Matsunami”)	40
2. Claims 3-4, 9-10, 15-16	41
C. Ground 3.....	44
1. U.S. Patent No. 7,398,528 (“Khawand”).....	45
2. U.S. Patent No. 5,805,930 (“Rosenthal”)	46
3. Claim 1	47
4. Claims 2, 8, 14	65
5. Claims 6, 12, 18	65

6.	Claim 7	66
7.	Claim 13	70
D.	Ground 4	72
1.	U.S. Patent No. 8,028,284 (“Ohkawa”)	73
2.	Claim 1	74
3.	Claims 2, 8, 14	79
4.	Claims 6, 12, 18	79
5.	Claim 7	80
6.	Claim 13	80
E.	Ground 5	82
1.	Claims 3-4, 9-10, 15-16	83
VI.	STANDING	87
VII.	DISCRETIONARY ANALYSIS	87
VIII.	MANDATORY NOTICES AND FEES	90

EXHIBIT LIST

Exhibit No.	Description
1001	U.S. Patent No. 8,924,596 to Beeson, et al. (“the ’596 patent”)
1002	Certified Prosecution File History for the ’596 patent (“Prosecution History”)
1003	Declaration of Dr. Samuel Russ (“Russ Declaration”)
1004	European Prosecution History for 14867963 (“European Prosecution History”)
1005	U.S. Patent No. 8,559,439 to Dongare, et al. (“Dongare”)
1006	U.S. Patent No. 8,051,227 to Gewirtz, et al. (“Gewirtz”)
1007	U.S. Patent No. 6,810,462 to Matsunami, et al. (“Matsunami”)
1008	U.S. Patent No. 7,398,528 to Khawand, et al. (“Khawand”)
1009	U.S. Patent No. 5,805,930 to Rosenthal, et al. (“Rosenthal”)
1010	U.S. Patent No. 8,028,284 to Ohkawa (“Ohkawa”)
1011	U.S. Patent No. 5,204,954 to Hammer, et al. (“Hammer”)
1012	N. Wirth, “On Multiprogramming, Machine Coding, and Computer Organization,” Communications of the ACM, Vol. 12, No. 9, Sept. 1969, pp. 489-498
1013	U.S. Patent No. 5,276,886 to Dror (“Dror”)
1014	J. Agron et al., “Run-Time Services for Hybrid CPU/FPGA Systems on Chip,” 2006 27th IEEE International Real-Time Systems Symposium (RTSS’06), Rio de Janeiro, Brazil, 2006, pp. 3-12
1015	John Goodacre, Andrew N. Sloss, “Parallelism and the ARM Instruction Set Architecture,” <i>Computer</i> , July 2005, pp. 42-50
1016	Shih-Hao Ou, Tay-Jyi Lin, Xiang Sheng Deng, Zhi Hong Zhuo and Chih Wei Liu, “Multithreaded coprocessor interface for multi-core multimedia SoC,” 2008 Asia and South Pacific Design Automation Conference, Seoul, Korea (South), 2008, pp. 115-116
1017	Excerpts from Plaintiffs’ Supplemental Claim Chart for the ’596 Patent
1018	<i>Concurrent Ventures LLC et al. v. Advanced Micro Devices, Inc. et al.</i> , Case No: 1:24-cv-00335-ADA, Docket No. 1 – Complaint for Patent Infringement
1019	Excerpts from “The Art of Electronics” by Paul Horowitz and Winfield Hill, cover, inside cover, copyright page, table of contents, 512

Exhibit No.	Description
1020	Excerpts from “Digital Design Principles & Practices” by John F. Wakerly, cover, inside cover, copyright page, table of contents, 590
1021	Excerpts from “Electrical Engineering An Introduction” by Schwarz & Oldham, cover, inside cover, title page, copyright page, table of contents, 435
1022	Excerpts from “Principles of Digital Design” by Daniel D. Gajski, cover, copyright page, table of contents, 268
1023	Excerpts from “Comprehensive Dictionary of Electrical Engineering”
1024	Excerpts from “Electrical Engineering Dictionary”
1025	Atmel 8-bit Microcontroller Datasheet
1026	U.S. Patent No. 5,907,485 to Van Loo, et al. (“Van Loo”)
1027	U.S. Patent No. 5,717,954 to Grieff, et al. (“Grieff”)
1028	U.S. Patent No. 8,463,958 to Mannava, et al. (“Mannava”)
1029	Plaintiffs’ Proposed Preliminary Claim Constructions served on January 23, 2025, in <i>Concurrent Ventures LLC et al. v. Advanced Micro Devices, Inc. et al.</i> , Case No: 1:24-cv-00335-ADA
1030	AMD’s Proposed Preliminary Claim Constructions served on January 23, 2025, in <i>Concurrent Ventures LLC et al. v. Advanced Micro Devices, Inc. et al.</i> , Case No: 1:24-cv-00335-ADA
1031	“Applied Micro ARM servers all over Open Compute Project” by Timothy Morgan, available at https://www.theregister.com/2013/01/17/applied_micro_x_gen_e_open_compute , published on January 17, 2013.

The '596 patent seeks to avoid software semaphores/mutexes when accessing queues. But the alleged invention—a “reservation register” that stores the amount of space in the queue—was trivial and already known, e.g., as taught by Gewirtz and Rosenthal. This IPR simply combines Gewirtz (Ground 1) and Rosenthal (Ground 3) with conventional computer architectures that describe the various processing elements and queues recited in the Challenged Claims.

I. BACKGROUND

Queues and registers are basic structures that have been used in computers and logic design for decades. Like a line of people, a queue inside a computer is also a line, where the first item of data to enter is also the first to leave. This first-in first-out policy of queues is often abbreviated “FIFO.” Ex-1003, ¶¶1004-1007.

In computers, a queue is a data structure with a fixed amount of space to hold data. Once a queue is full, no new data can be placed into it. That is why queue structures included mechanisms to indicate how much space was left. By being able to obtain this information, whoever was writing to the queue was able to avoid overflowing it with too much data. This process of managing the flow of data into and out of queues was known as flow control. Ex-1003, ¶1008.

A common mechanism for reporting queue space was with a register. For example, Gewirtz described a “reservation register” that tracked the “available entries for reservation” in a queue. Ex-1006, 4:51-5:6, 4:5-7. Rosenthal also

described a flow control register that “stores an indication of the number of available spaces for commands” in its command queue. Ex-1009, 16:33-37. The prior art is replete with examples of registers that tracked the amount of space free in a queue. *See* Ex-1011, 9:47-58 (describing a counter for a queue and setting a queue full flag when the counter reaches the queue’s size limit); Ex-1026, 9:12-18 (describing a counter that is checked against the queue size to determine if a request can be issued to that queue); Ex-1027, 8:26-32 (describing using a counter to determine the available space in a FIFO); Ex-1028, 8:32-43 (describing a counter that counts the number of available storage locations in a buffer).

Queues were often used as buffers between processing elements. By 2013, many systems included multiple processing engines that consumed and passed data at different rates, e.g., CPUs, dedicated hardware engines, DSPs, embedded processors, and other types of processors. Queues were often used to buffer communication and create a steadier flow of data. An exemplary computer architecture is shown below, with three processing elements passing commands to each other using input and output queues.

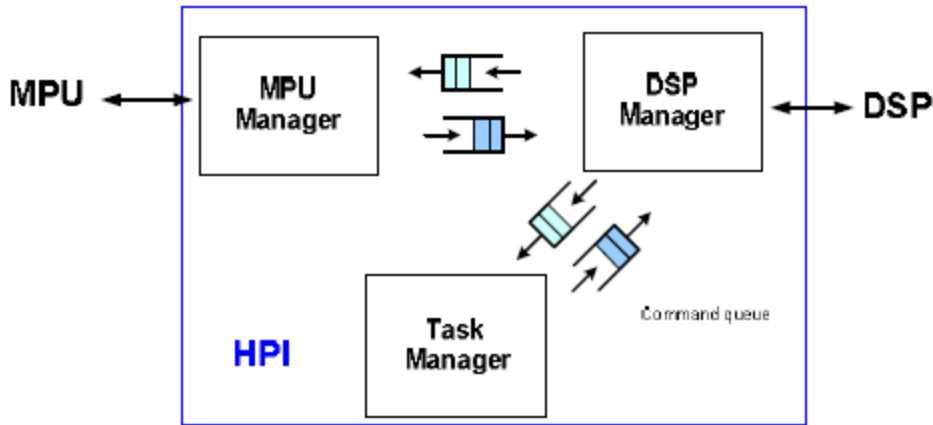


Fig. 3 HPI block diagram

Ex-1016, Fig. 3; Ex-1003, ¶¶10-13.

A common way for processors to access control registers, including the queue status registers discussed above, was by mapping the registers to addresses in memory space. This process, known as memory-mapping, simply assigned a specific address to that register so that reads and writes to that memory address were directed to the register. Ex-1003, ¶¶14-16.

To summarize, by 2013, queues were widely used in computer architectures with more than one processor, and they were often used with reservation registers that reported the amount of remaining space in the queue. Ex-1003, ¶33.

II. THE '596 PATENT

According to the '596 patent, the prior art used software techniques, such as semaphores and mutexes, to coordinate access to queues, but software techniques were inefficient and not optimal. Ex-1001, 1:22-27. To address these purported shortcomings, the '596 patent purports to invent a “reservation register” that

“store[s] a value indicative of available space” in the input queue. However, this was not a novel concept in 2013, and the prior art already included numerous examples of reservation registers indicating the amount of free space in queues, applied to multiprocessor architectures. Ex-1003, ¶¶34-37.

A. Prosecution History

On August 19, 2014, the ’596 patent received a first action allowance after Examiner’s Amendments were entered with no substantive discussion of any prior art references. Ex-1002, 000060-75; Ex-1003, ¶¶38-41.

During prosecution of a European counterpart, EP14867963 (the “EU Counterpart”), a Supplementary Search Report identified Gewirtz and another reference Kim *after* the ’596 had already issued in the U.S. Ex-1004, 000080-93, 000111-113. Gewirtz and Kim were used as a basis for a rejection in the EU (Ex-1004, 000114-123), and the EU Counterpart was eventually abandoned after only one substantive response from the applicant. Ex-1004, 000128-144, 000159; Ex-1003, ¶¶42-46.

Regarding the queues recited in the patent, the EU Search Opinion specifically noted that “[i]t is common general knowledge of the skilled person [] that an output queue [] is an implementation choice for buffering request results from a storage device.” Ex-1004, 000119; Ex-1003, ¶44.

B. PO's Allegations

In district court, PO alleged that Pensando's Distributed Services Card ("DSC") NIC infringed the '596 patent. Ex-1017, 000001. According to PO, the alleged reservation register was simply data in DRAM that reported "current state in a DRAM-based qstate record, which includes a count of enqueued objects, pointers to arrays or linked lists of enqueued objects, connection state and peer information, and a process ID used for memory protection." Ex-1018, 000016. To counteract PO's improper infringement theory, Petitioners proposed a construction that "register" means "set of flip-flops that store a set of related bits." Ex-1030, 000009. PO proposed "a memory location within a microprocessor used to store values and external memory addresses." Ex-1029, 000002. For purposes of this IPR, the parties' dispute is irrelevant because the references teach "registers." *Infra* §§V.A.3[1e], V.C.3[1e]; Ex-1003, ¶¶48-50.

PO took a similarly broad view of the other claim terms. Ex-1017, 00003-38; Ex-1018, 000012-13, 000016. For example, for the "input queue" and "output queue implemented in hardware," PO broadly alleged that these queues can be satisfied by an assortment of queues, including queues implemented in memory, such as DRAM. Ex-1017, 000021-36; Ex-1018, 000012, 000016; Ex-1003, ¶¶51-52.

C. Priority Date

Petitioner demonstrates invalidity as of December 6, 2013. Ex-1003, ¶47.

III. FIELD AND LEVEL OF SKILL

The '596 patent's field is computer architecture, including the use of queues and multiple processing elements. That is the focus of the claims and specification and how the patent describes its alleged invention. Ex-1001, 5:3-6, 3:30-33; Ex-1003, ¶54.

A POSITA would have had a bachelor's degree in electrical engineering, computer engineering, or comparable major with at least two years of experience in the field or a master's degree in electrical engineering, computer engineering, or similar major with a specialization in the field. A person with less education but more relevant practical experience may also meet this standard. Ex-1003, ¶¶55-57.

IV. CLAIM CONSTRUCTION

The following constructions were proposed in district court. The Challenged Claims are invalid irrespective of whether these constructions are adopted.

A. “[input/output] queue implemented in hardware”

In district court, Petitioners proposed that these terms are “input/output queue implemented in physical circuits without software intervention.” The specification states: “the hardware will then automatically, **without software intervention**, update the reserved free space when the command is actually

removed from the **queue for consumption.**” Ex-1001, 4:43-46.¹ This is an example of how the ’596 patent explains what it means when it describes something as implemented in hardware. Ex-1003, ¶¶58-62.

PO proposed plain and ordinary meaning. Ex-1029, 000002.

V. INVALIDITY GROUNDS

Petitioner requests *inter partes* review and cancelation of the challenged claims on the following grounds:

Grounds	Claims	Statutory Basis	Prior Art
1	1-2, 5-8, 11-14, 17-18	§103	Dongare and Gewirtz
2	3-4, 9-10, 15-16	§103	Ground 1 and Matsunami
3	1-2, 6-8, 12-14, 18	§103	Khawand and Rosenthal
4	1-2, 6-8, 12-14, 18	§103	Ground 3 and Ohkawa
5	3-4, 9-10, 15-16	§103	Ground 4 and Matsunami

Ex-1003, ¶¶63-66. The Grounds render the challenged claims obvious because any differences between the claimed subject matter and prior art are such that the subject matter, as a whole, was obvious at the time the invention was made to a POSITA. *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 406 (2007).

¹ All emphasis and annotations added unless otherwise specified.

A. Ground 1

Dongare teaches a RAID² ASIC architecture where an embedded processor manages several input queues. Ex-1005, 10:50-11:30, 14:30-55. Gewirtz teaches known techniques for managing queues. Ex-1006, 2:46-3:7. Ground 1 combines these teachings. The motivation is explained below. *Infra* §V.A.3[1e]; Ex-1003, ¶67.

1. U.S. Patent No. 8,559,439 (“Dongare”)

Dongare teaches an architecture where a host processor writes commands to input queues managed by an embedded processor. Ex-1005, Abstract, 1:21-22, Fig. 4. The embedded processor handles various aspects of the queues and is responsible for resequencing results from the RAID engines, if needed. Ex-1005, 14:30-55; Ex-1003, ¶¶70-72.

² RAID stands for redundant array of independent disks. Ex-1007, 7:1-17. RAID used multiple disks for reliability.

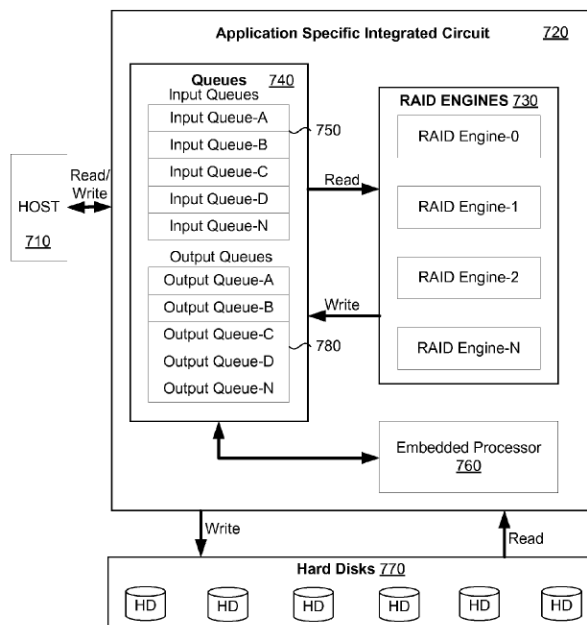
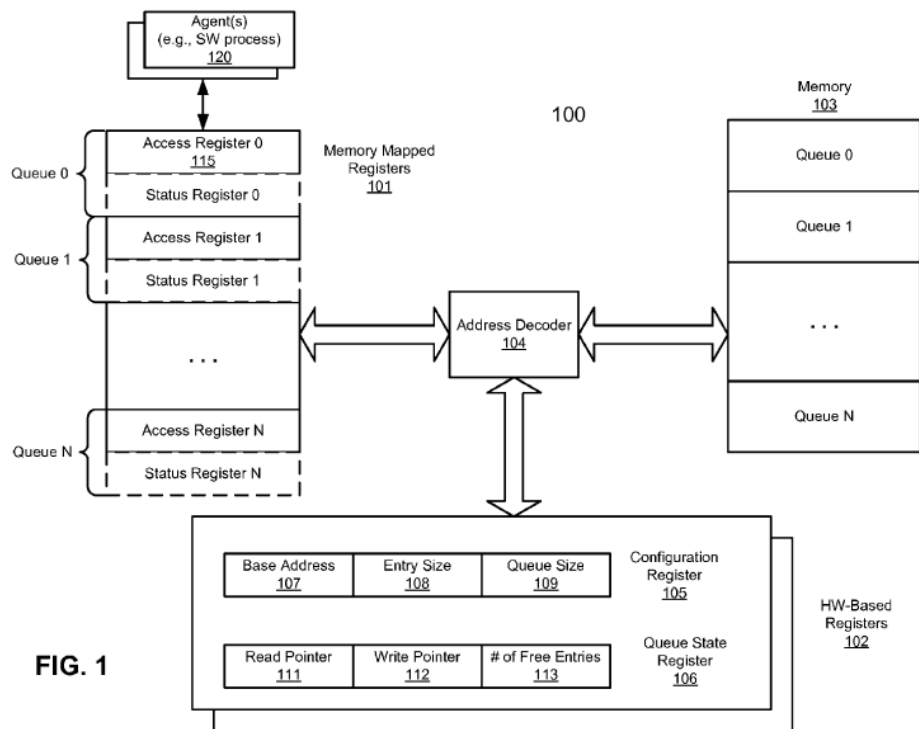


FIG. 7

Dongare is prior art under §§102(a)(1)-(2). Ex-1003 ¶¶68-69. Dongare is analogous art in the same field (*supra* §III) because Dongare describes a computer architecture and teaches embodiments with queues and queue ordering for increased efficiency. Ex-1005, 1:14-16, 1:36-60; Ex-1003, ¶73.

2. U.S. Patent No. 8,051,227 (“Gewirtz”)

Gewirtz teaches techniques for managing queues, including the use of a queue state register that contains the number of free entries available. Ex-1006, 1:12-64, 2:59-62, 4:51-58; Ex-1003, ¶¶76-78.



Gewirtz is prior art under §§ 102(a)(1)-(2). Ex-1003, ¶¶ 74-75. Gewirtz is analogous art in the same field (*supra* § III) because it teaches a computer architecture including atomic hardware operations for queue management. Ex-1006, 1:12-35, 2:46-3:7, 4:16-50; Ex-1003, ¶ 79.

3. Claim 1

[1pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:

Dongare teaches a system for dividing and synchronizing a processing task across a plurality of processing elements, including, e.g., a host processor, embedded processor, and RAID engines. Ex-1005, Fig. 7. For example, Dongare

teaches that “[m]ultiple engines or processors are used to increase the throughput of a device or a network system.” Ex-1005, 1:21-22. Dongare divides processing tasks to enable parallel execution across a plurality of RAID engines and then synchronizes the output as necessary to account for out-of-order execution. Ex-1005, 1:21-32, 4:49-55, 14:31-55; Ex-1003, ¶¶80-82.

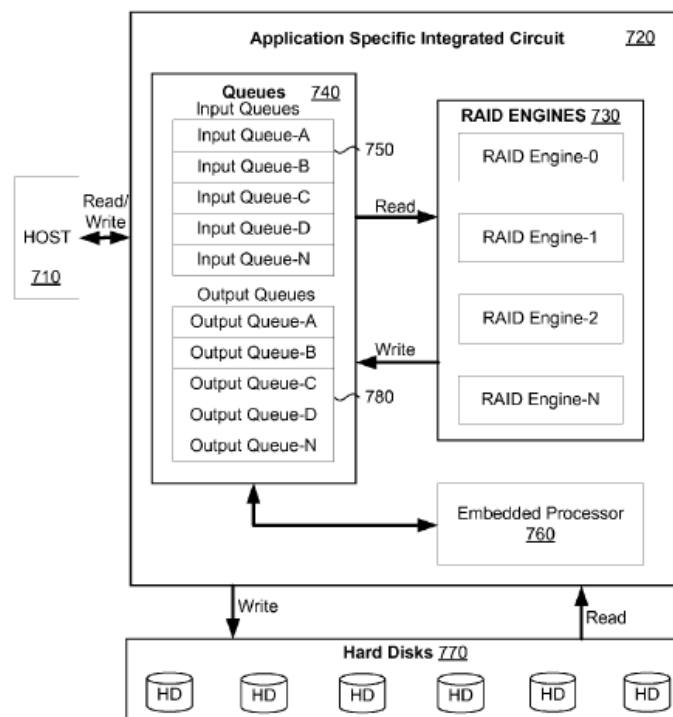


FIG. 7

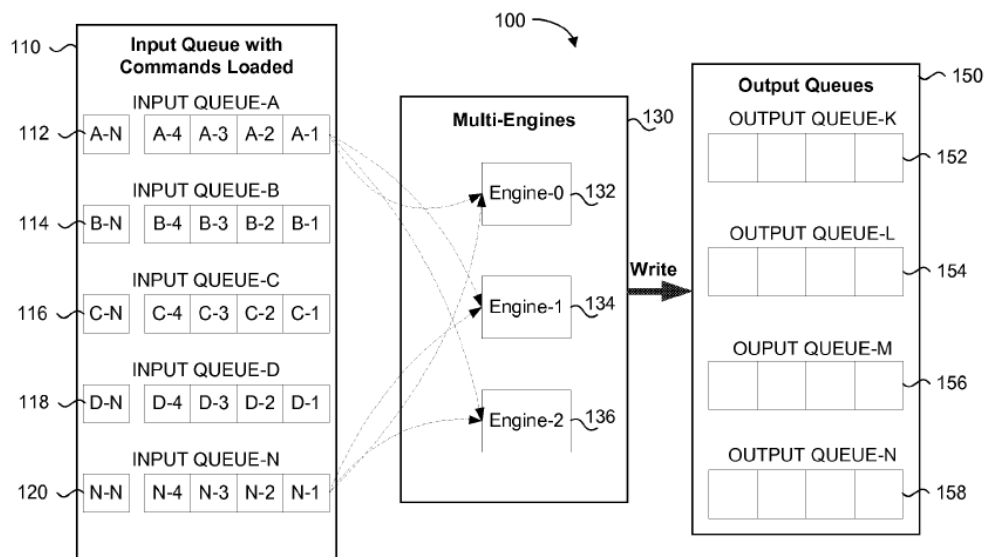


FIG. 1

The system further comprises the elements explained below. *Infra* §§V.A.3[1a]-[1j]; Ex-1003, ¶83.

[1a] an input queue implemented in hardware;

Dongare teaches input queues in the RAID ASIC for holding commands from the host to the processing engines. Ex-1005, 14:30-34. “FIG. 1 illustrates a multi-engine processing system 100 capable of handling a plurality of input queues[.]” Ex-1005, 5:3-36, 14:39-42 (“The queues 740 in the controller are classified as **input queues 750 from the Host**[.]”); Ex-1003, ¶¶84-87.

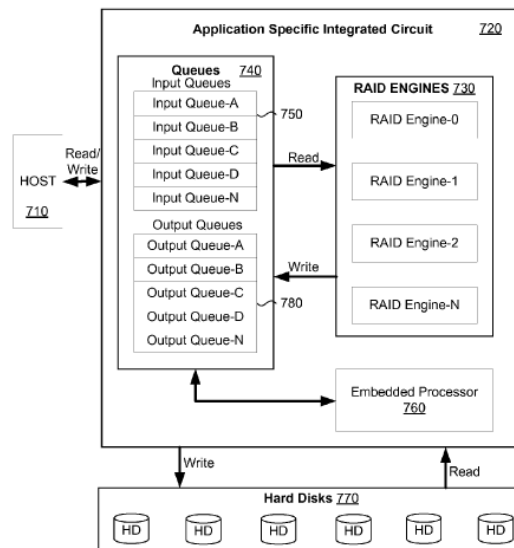


FIG. 7

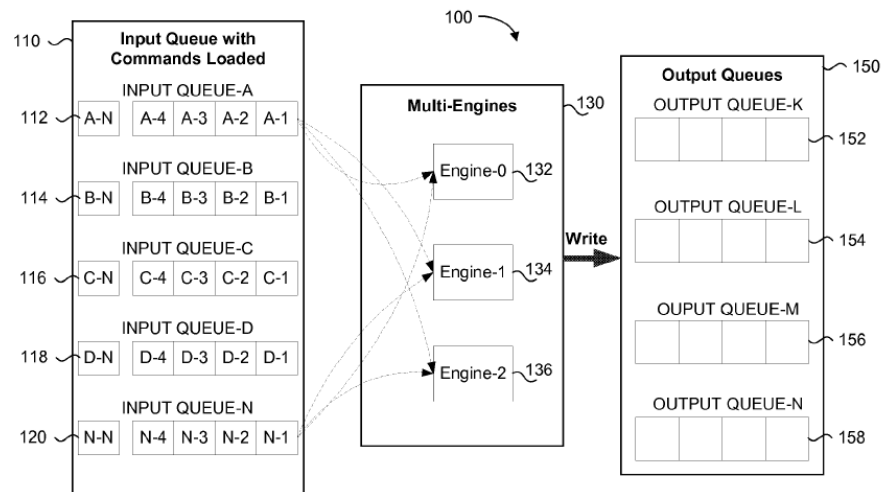


FIG. 1

Dongare teaches implementation in hardware including for input queues.

Ex-1005, 11:14-16. The input queue is implemented in physical circuits, without

software intervention, because Dongare teaches an implementation as a “hardware circuit.” Ex-1005, 11:14-16, 2:55-58; Ex-1003, ¶¶88, 90-91.

Additionally, to the extent PO somehow argues that Dongare or Gewirtz use memory, PO alleged in district court that a queue implemented in memory using software is a queue implemented in hardware. Ex-1017, 000003-14.

[1b] an output queue implemented in hardware;

Dongare teaches at least one output queue in the RAID ASIC for processed commands. “The processed commands are written out to the plurality of output queues 150.” Ex-1005, 5:3-20, 14:30-42 (“...output queues from the RAID engines 730...”), Fig. 7. “RAID engines 730... writes the processed commands back to output queues 780.” Ex-1005, 14:46-49; Ex-1003, ¶¶92-94.

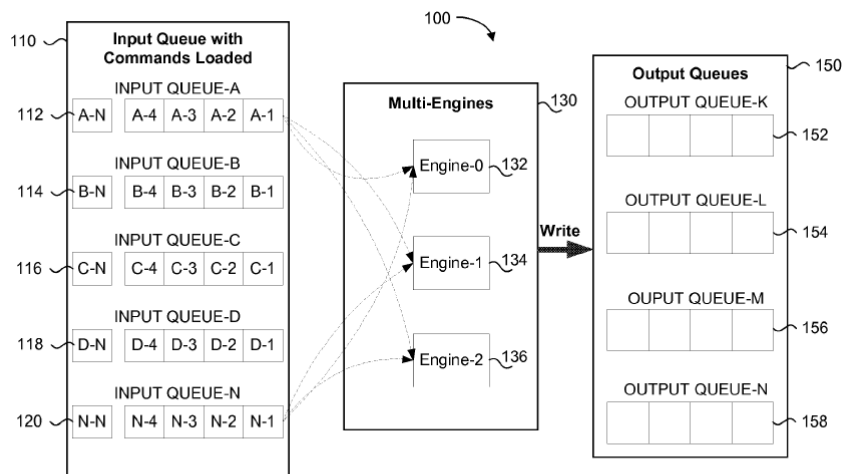


FIG. 1

Dongare teaches implementation in hardware including for output queues. Ex-1005, 11:14-16. The output queue is implemented in physical circuits, without software intervention, because Dongare teaches an implementation as a “hardware circuit.” Ex-1005, 11:14-16, 2:55-58; Ex-1003, ¶¶95, 97-98.

Additionally, to the extent PO somehow argues that Dongare or Gewirtz use memory, PO alleged in district court that a queue implemented in memory using software is a queue implemented in hardware. Ex-1017, 000003-14.

[1c] a first processing element having access to said input queue and said output queue;

Dongare’s host processor is a first processing element that has access to the input queue and output queue. The host processor is connected to the RAID ASIC using PCIe (Ex-1005, 14:31-34) and has access to the input queue because it writes commands to the input queue. The “Host 710 and RAID controller 720 communicate to each other by using a set of queues 740 on the RAID controller 720” and the “Host 710 writes commands for the RAID controller into the input queues 750.” Ex-1005, 14:37-42; Ex-1003, ¶¶99-100.

Dongare’s host processor accesses the output queue, for example by reading the output queue, after it is signaled by the embedded processor that the host’s command has been completed. Ex-1005, 14:46-54; Ex-1003, ¶¶101-02.

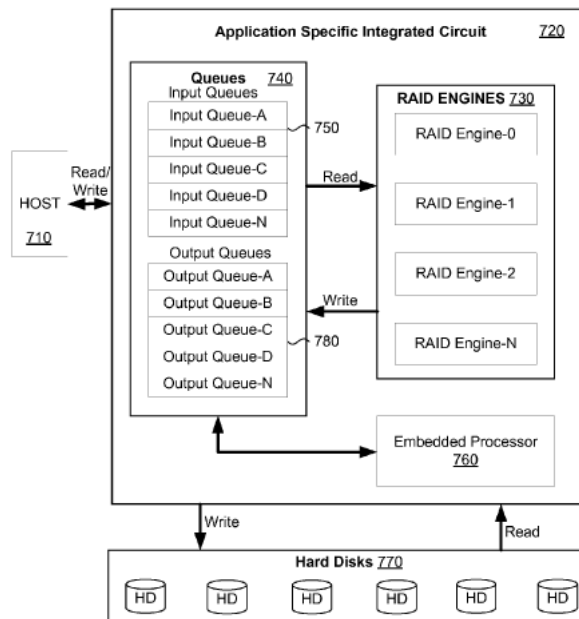


FIG. 7

[1d] at least one second processing element in communication with said first processing element;

Ground 1 teaches at least one second processing element (e.g., Dongare's Embedded processor) in communication with said first processing element (e.g., Dongare's Host processor). The embedded processor communicates with the host processor over PCIe, using input and output queues. Ex-1005, Fig. 7, 14:31-44. The embedded processor also signals and interrupts the host by writing to the outbound queue. Ex-1005, 14:52-55; Ex-1003, ¶¶103-106.

[1e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;
--

Dongare teaches that the embedded processor manages the input queues, e.g., through queue handling/scheduling, but leaves implementation details to the POSITA. Ex-1005, 10:50-11:30; *infra* V.A.3[1g]. One aspect of queue-management includes tracking available space. Gewirtz provides those teachings using a Queue State Register, which it also refers to as “the **reservation register**.” Ex-1006, 4:51-5:6; Ex-1003, ¶¶107-108.

Gewirtz’s Queue State Register is a reservation register that stores a value indicative of available space in the input queue (e.g., # of free entries). This register, also referred to as a state/status register, “holds the dynamic state of the queue, including... queue occupancy, and/or number of free writeable entries for flow control purposes.” Ex-1006, 2:59-64, 3:55-59, 4:5-15 (describing “reservation field... used to specify the number of entries available for writing into the queue.”).

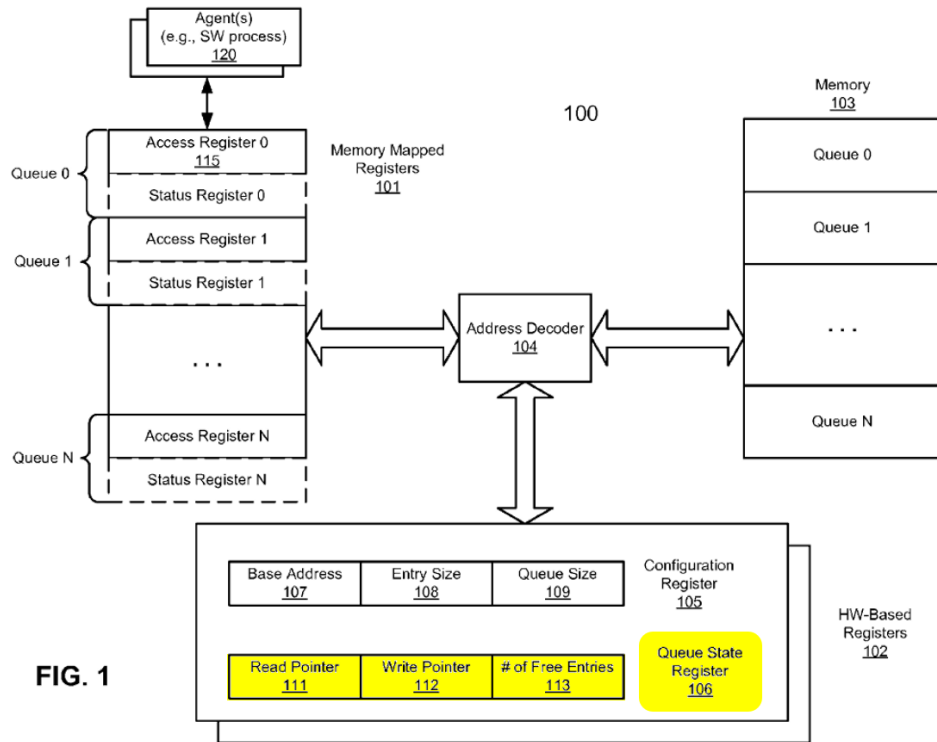


FIG. 1

Ex-1003, ¶109.

Software reads Gewirtz’s reservation register to see if space is available; on a successful reservation read request, hardware auto-decrements the “available

reservation count.” Ex-1006, 4:51-5:6, 1:30-64, 6:18-25, 6:66-7:6. Therefore, Gewirtz teaches a reservation register.^{3, 4} Ex-1003, ¶¶110-11.

Gewirtz’s Queue State Register is a “HW-Based Register[]” (Ex-1006, Fig. 1) that is “updated by hardware logic that manages the queues.” Ex-1006, 2:59-64. Gewirtz meets Petitioners’ district court construction of “register” (*supra* §II.B) because hardware registers have, for many decades, been implemented using a set of flip-flops.⁵ Therefore, Gewirtz’s “HW-Based Register” is a set of flip-

³ The ’596 patent confirms that these teachings fall within its scope. Ex-1001, 3:43-4:51 (“hardware automatically decrements the available free space...”).

⁴ A POSITA would have been motivated to use Gewirtz’s Queue State Register teachings because they provided a simple mechanism for flow control that improved efficiency. Ex-1006, 1:30-64, 4:55-5:6, 6:18-7:6; Ex-1003, ¶111.

⁵ A POSITA would have known about registers and understood that they comprise a set of flip-flops because this was taught in undergraduate logic design courses for decades. *See* Ex-1019, 000013 (“The flip-flops have all been combined into a single *register*, which is nothing more than a set of type *D* flip-flops[.]”) (*italics original*); Ex-1020, 000012 (“A collection of two or more *D* flip-flops with a common clock input is called a *register*. Registers are often used to store a collection of related bits, such as a byte of data in a computer.”) (*italics original*); Ex-1021, 000012

flops storing a set of related bits, i.e., fields related to queue status. Ex-1006, 4:5-15, Fig. 1; Ex-1003, ¶112.

The reservation register is accessible by both the first and second processing elements. Gewirtz makes the register accessible by software agents running on “multiple processor cores.” Ex-1006, 4:56-61. Dongare’s host and embedded processors are the processors in Ground 1. Therefore, software running on the host and embedded processors would obviously have had access to the reservation register, following Gewirtz’s teachings, to access and utilize the register. *See id.* A POSITA would have recognized the value in having this information be visible to multiple processors for synchronization purposes. *See id.*; Ex-1003, ¶113.

Additionally, Gewirtz’s queue state register is accessible to processors in the system—including Dongare’s host and embedded processors—because Gewirtz

(“Normally a byte containing N bits is stored in a set of N flip-flops. This set of flip-flops is then referred to as a *register*.”) (*italics original*); Ex-1022, 000008 (“The simplest of the storage components is a **register** which can be thought of as a bitwise extension of a flip-flop. Each register consists of n flip-flops driven by a common clock signal.”) (*italics and emphasis original*); Ex-1023, 000025 (“register a circuit formed from identical flip-flops or latches and capable of storing several bits of data.”); Ex-1024, 000026 (same definition); Ex-1003, ¶264.

teaches it as a memory-mapped register. Ex-1006, 3:10-19, 4:51-54. Gewirtz explains that the Queue State Register is one of the hardware-based registers 102 associated with a queue (Ex-1006, 3:19-27, Fig. 1), and the “[m]emory mapped registers of a particular queue may be mapped and/or translated by address decoder 104 to one or more of hardware-based registers 102” (Ex-1006, 3:27-31). Accordingly, Gewirtz teaches accessing the hardware Queue State Register via the “memory mapped registers 101 corresponding to the queue.”⁶ Ex-1006, 4:51-54. It would have been obvious for a memory-mapped register to be accessible by processors because, for decades, conventional multiprocessor systems have included hardware-based memory-mapped registers for this purpose, and students have learned how to access those registers through a computer’s memory space. Therefore, a POSITA with basic education/work experience in computer architecture would have understood and found it obvious, from Gewirtz’s teaching, that the queue state register is accessible to processors in the system, including

⁶ Gewirtz teaches PO’s district court construction of “register” (Ex-1029, 000002) because Gewirtz’s reservation register is accessible to the microprocessor via a memory address and stores a value, e.g., the number of free entries in the queue. Ex-1003, ¶114, n.2.

Dongare's host/embedded processors, because Gewirtz teaches it as a memory-mapped register. *See* Ex-1006, 4:51-54; Ex-1003, ¶114.

Motivation to Combine. Gewirtz teaches queue-management techniques for tracking queue space and flow control. Ex-1006, 1:12-35, 2:46-3:7, 4:16-50. Dongare's system uses many input and output queues, providing motivation to combine its teachings with known techniques for managing those queues. Ex-1005, 10:50-11:30. A POSITA would have been motivated to apply Gewirtz's teachings to Dongare; this was a combination of known prior art elements according to known techniques to yield a predictable result—management of Dongare's input queues using a register and Gewirtz's flow control. By 2013, registers, queue management, and flow control techniques were well-known, and their combination with Dongare would have been predictable and straightforward. Ex-1003, ¶115.

Dongare's architecture creates a need for synchronization and flow control with its queues as tasks are scheduled and divided among multiple processing engines. Ex-1005, 6:28-42, 7:34-39. Since different commands take different amounts of time to process (Ex-1005, 6:6-35, Abstract), commands are removed from input queues at different rates. This variability creates a need for synchronization and flow control because the number of entries in a queue at any given time is unpredictable, and processors need a mechanism to check the number

of entries available before adding more commands to the queue. *Infra* n.7 (explaining how the embedded processor manages input queues); Ex-1003, ¶116.

Gewirtz teaches a solution to this common problem of variable-rate data flows. Ex-1006, 2:46-3:7, 3:32-42, 8:7-17. Gewirtz teaches queue-management solutions—including a register—optimized for queue flow control. Ex-1006, 2:59-62, 6:66-7:20. Therefore, a POSITA would have been motivated to apply Gewirtz's teachings. Ex-1003, ¶117.

Gewirtz's teachings provide several known benefits including atomic hardware operations to avoid software spin locks (Ex-1006, 1:30-64, 6:18-7:6) and flow control using a register (Ex-1006, 4:51-61). A POSITA would have been motivated to use Gewirtz's Queue Status Register to manage queues in Dongare's architecture because register implementations were common in computer architecture and the prior art provides motivation to track available space using a register. Ex-1013, 1:11-37 (discussing a general purpose register used as a counter); Ex-1016, 000001-2 (discussing command queues and queue management including monitoring and scheduling); Ex-1006, 3:4-7 (describing queue maintenance functions). Gewirtz's teachings prevent queue overflow and increase throughput/efficiency. Ex-1006, 4:51-67, 6:66-7:14; Ex-1003, ¶¶118-119.

Dongare provides motivation to improve throughput/efficiency for multi-engine systems. Ex-1005, 4:49-55. Gewirtz provides increased throughput and

efficiency, using flow control, when multiple processors access queue structures.

Ex-1006, 6:8-7:14, 1:6-8; Ex-1003, ¶120.

Additionally, the references also provide explicit motivation to modify their teachings and combine them with other known techniques. Ex-1006, 12:1-4; *see also* Ex-1005, 15:48-52; Ex-1003, ¶121.

A POSITA would have had a reasonable expectation of success in combining Dongare's and Gewirtz's teachings. Gewirtz is directed at queue management and flow control. Ex-1006, 1:12-35, 2:46-3:7, 4:16-50. Dongare teaches an architecture with multiple queues that have variable rates of data flow—precisely the type of architecture that Gewirtz provide solutions for. Ex-1006, 1:12-35, 2:46-3:7, 4:16-50. The combination would have been a straightforward application of these known techniques and routine technologies like registers, queue management, and flow control to improve Dongare's architecture. A POSITA would have easily combined the teachings of Dongare and Gewirtz. For example, adding Gewirtz's memory-mapped register teachings would have been simple and straightforward to accomplish because essentially all modern processors uses memory mapping to make registers visible. Combining the references would have been the straightforward application of known prior art elements according to known methods and would have yielded a predictable result. Ex-1003, ¶122.

The combination would have been within the skills of a POSITA, with minimal modifications due to the use of well-known techniques. Ex-1006, 2:64-66. Registers, queue-management, and flow-control were undergraduate concepts, common building blocks in computer architecture. Dongare’s architecture uses multiple processors, which would naturally have used memory-mapped registers, which had been routine for decades. Therefore, a POSITA would have been motivated to combine with a reasonable expectation of success in doing so. Ex-1003, ¶123.

The combination would not have changed the principle of operation for any reference. Dongare would still have its RAID ASIC architecture with multiple processors and queues. Ex-1005, 14:30-55. Gewirtz’s queue management and flow control teachings would serve the same purpose of tracking queue space with a status register and managing queues. Ex-1006, 4:51-5:6. Gewirtz’s techniques apply equally to queues implemented in hardware or software; Gewirtz’s use of a reservation register for flow control would be the same. Ex-1003, ¶¶124-25.

[1f] computer storage storing instructions, which when executed by said at least one second processing element:

Dongare teaches using “a computer program product stored in a machine-readable medium” with “instructions” that “when executed, cause a processor to perform steps in a method according to an embodiment of the disclosure...”

Ex-1005, 15:27-47. Likewise, Gewirtz teaches implementation via “a computer program” “stored in a computer readable medium....” Ex-1006, 11:40-53, 11:14-16. Dongare and Gewirtz teach the second processing element performing steps [1g]-[1i]; therefore, they teach instructions executed by the second processing element for performing those steps. *See* Ex-1006, 11:40-53; *supra* §V.A.3[1e] (motivation subsection). It was obvious for Dongare’s embedded processor to access computer storage storing instructions because that is how processors have worked for decades. Ex-1003, ¶¶126-29.

[1g] accesses said reservation register and reads said stored value;

Dongare’s embedded processor manages the input queues. Ex-1005, 10:50-11:30, 14:30-55, Figs. 5, 7.⁷ One part of queue management is tracking the

⁷ Dongare teaches queue management functions in a “multi-engine processing system,” including input queue handling and other pre-processing steps for providing commands to the processing engines. Ex-1005, 10:50-11:30, Fig. 5. Dongare then teaches a RAID application with an embedded processor that performs preprocessing steps on the ASIC for the multi-engine processing system. Ex-1005, 14:30-55, Fig. 7. Here, the embedded processor obviously managed the input queue because it performed steps related to input-queue handling, scheduling, and

amount of free space in the queue. Gewirtz's Queue Status Register contains the "available entries for reservation" which enables a processor to track available space. Ex-1006, 4:5-7; Ex-1003, ¶¶130-131.

Ground 1 combines Dongare with Gewirtz, with Dongare's embedded processor managing the input queue by first reading the number of available entries for reservation from the Queue Status Register. *Supra* §V.A.3[1e] (motivation subsection); Ex-1006, 4:5-7, 4:56-61 ("...first **reads the reservation register** to see if an entry is available"). The "reservation field can be used to specify the number of entries available for writing into the queue."

Ex-1006, 4:12-13. It was obvious and natural for Dongare's embedded processor to perform a coordination role and read the reservation register due to its proximity to the queues and the register (everything is on-chip, avoiding cross-chip

reordering, which correspond to queue management functions described for Figure 5. Ex-1005, 14:30-55. Moreover, Dongare teaches a software implementation (Ex-1005, 10:50-11:30); in the RAID ASIC, the embedded processor would be the natural and obvious processor for running such software (Ex-1005, Fig. 7). Ex-1003, ¶130. Additionally, the embedded processor performs input queue handling (Ex-1005, 10:50-11:30, 6:28-35), which encompasses queue management and flow control. Ex-1003, ¶141.

communication delays), in contrast to the host processor. Therefore, it was obvious and more efficient for Dongare’s embedded processor to monitor the queue states via the reservation register and notify the host processor when there is space available for additional commands. *See* Ex-1006, 4:12-13, 4:56-58; Ex-1003, ¶¶131-32.

[1h] determines when said read value indicates available space in said input queue for said first processing element to issue a command;
--

Gewirtz’s Queue Status Register comprises a “reservation field” for “specify[ing] the number of entries available for writing into the queue.” Ex-1006, 4:12-13; *supra* §V.A.3[1e]. Gewirtz teaches that, before entries are written to the queue, a processor “first reads the reservation register to see if an entry is available.” Ex-1006, 4:56-64; Ex-1003, ¶¶133-34.

For Ground 1 (*supra* §V.A.3[1e] (motivation subsection)), Dongare’s embedded processor manages the input queue⁸ by reading the Queue Status Register’s reservation field to determine if the reservation field indicates space is available. Ex-1006, 4:12-13, 4:61-64; *supra* §V.A.3[1g]. A basic part of queue-management includes tracking the amount of free space in the queue, so it does not overflow. It was obvious for the embedded processor to be responsible for

⁸ *Supra* n.7 (explaining queue management by embedded processor).

preventing input queue overflow because, for Ground 1, the embedded processor, input queues, and reservation register are all on-chip for precisely this type of management function. *See* Ex-1005, 10:50-11:30. Dongare’s embedded processor has easier access because it does not have to go off-chip, through PCIe, to read the register. Ex-1005, 14:30-55; Ex-1006, 2:64-66. Therefore, it was obvious and most efficient for Dongare’s embedded processor to read the Queue Status Register for the input queue and determine whether the reservation field indicates available space for a command to be issued. Ex-1003, ¶135.

Dongare’s host processor issues commands for the RAID controller to the input queue, including for example, “read/write operations from Hard Disks 770.” Ex-1005, 14:41-49; Ex-1003, ¶¶136-37.

[1i] notifies said first processing element to issue said command to said input queue; and

For Ground 1, Dongare’s embedded processor reads the reservation register and notifies the host processor when space is available for additional commands to be issued. *Supra* §V.A.3[1g]-[1h], §V.A.3[1e] (motivation subsection). Dongare’s host processor then writes the commands to the input queue. Ex-1005, 14:41-49; Ex-1003, ¶¶138-39.

Dongare’s embedded processor schedules tasks for the RAID engines. Ex-1005, 10:50-11:2, 14:30-31.

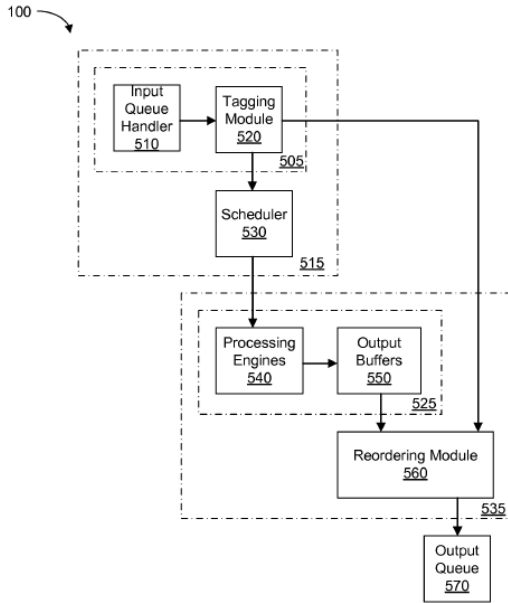


FIG. 5

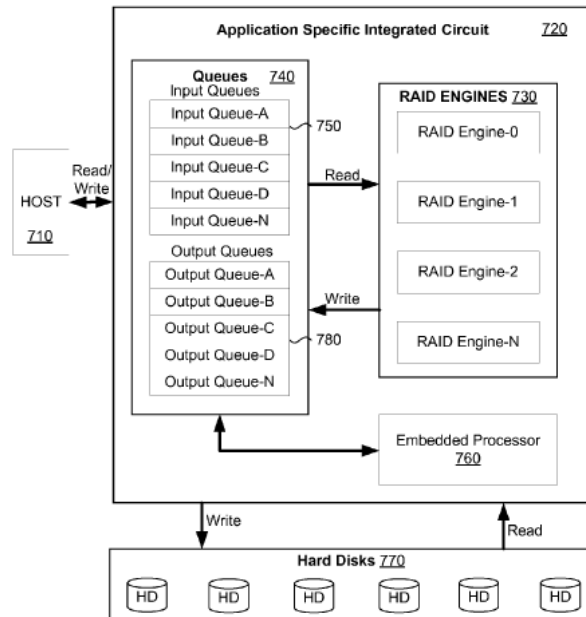


FIG. 7

Ex-1005, 11:20-23, 10:56-57; Ex-1003, ¶140.

Gewirtz's queues generate interrupts to a programmable target based on queue status/events, including when queue space is available. Ex-1006, 3:63-4:2 (“[A]n interrupt can be generated when a queue... drains below a predetermined threshold...”). Applied to Dongare, it was obvious for the programmable target to be on-chip with the queues and register, i.e., Dongare's embedded processor, which had better interrupt service routine performance due to its proximity.

Ex-1005, 14:30-55, Fig. 7. The Dongare-Gewirtz combination enables Dongare's

embedded processor to perform its queue-management and scheduling role⁹ by reading the reservation register (Ex-1006, 5:3-6) and notifying the host that space is available in the input queue. Ex-1006, 3:63-4:2. The prior art provides ample motivation for one processor to notify another when space is available. *See, e.g.*, Ex-1006, 3:63-4:2; Ex-1011, 10:3-26, 11:52-54; Ex-1003, ¶141.

Dongare's embedded processor notifies the host over the PCIe bus or by generating an interrupt to the host. Ex-1005, 14:30-34 (PCIe), 14:52-55 (interrupt); Ex-1003, ¶142.

In summary, when an input queue is full¹⁰—for example when the queue is full and waiting on RAID Engines to complete pending commands—the embedded processor notifies the host after the input queue drains below a predetermined threshold. Ex-1006, 3:63-4:2. The host processor can then issue commands to the input queue. *See* Ex-1006, 14:41-42. A POSITA would have been motivated to combine the references in this manner because the it enables the host to issue commands to the input queue without polling the Queue Status Register (or some other data structure) over PCIe. Ground 1 manages the queues locally with an on-

⁹ *Supra* n.7 (explaining queue management by embedded processor).

¹⁰ The host cannot write to a full queue and must wait until space is available. This was a basic undergraduate programming concept. Ex-1003, ¶143.

chip embedded processor, which can respond quickly to dynamic changes in the system's state, as compared to the external host processor. Therefore, the combination of Dongare and Gewirtz takes advantage of Gewirtz's teachings to monitor queue status and leverages Dongare's embedded processor to notify the host processor of queue space availability. Ex-1003, ¶¶143-44.

[1j] wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue.

The embedded processor notifies the host when there is space available in a previously full input queue. *Supra* §V.A.3[1i], §V.A.3[1e] (motivation subsection); *see* Ex-1006, 3:63-4:2. After the host receives this notification, it issues commands to the input queue. Dongare's host "writes commands for the RAID controller into the input queues 750." Ex-1005, 14:41-42. These commands are for example, "read/write operations from Hard Disks." Ex-1005, 14:46-49; Ex-1003, ¶¶145-46.

Ground 1 also teaches the first processing element receives a response corresponding to the command from the output queue. Dongare's "RAID engines 730 reads commands from input queues 750 and **after processing them (e.g., read/write operations from Hard Disks 770)**, writes the processed commands **back to output queues 780.**" Ex-1005, 14:46-49. These commands

were submitted by the host processor through the input queue to perform read/write operations from the RAID hard disks. Ex-1005, 14:41-42 (“The Host 710 writes commands for the RAID controller into the input queues 750.”). The RAID engines process these commands. Ex-1005, 14:35-37; Ex-1003, ¶147.

Dongare’s embedded processor signals the host when the RAID engines have finished processing a command and the output is available in the output queue. Ex-1005, 14:49-55. Dongare’s embedded processor uses the output queue to reorder resulting data due to possible out-of-order execution before signaling the host of completion. Ex-1005, 13:18-25. The host processor then reads the output queue over PCIe to receive the response corresponding to the command it submitted. Ex-1005, 14:31-39; Ex-1003, ¶¶148-49.

4. Claims 2, 8, 14

[2/8/14]. The system of claim [1/7/13], wherein said input queue is a first-in-first-out (FIFO) buffer.

Ground 1 teaches claims 1/7/13. *Supra* §V.A.3, *infra* §§V.A.7, V.A.8; *supra* §V.A.3[1e] (motivation subsection); Ex-1003, ¶¶150-51, 180, 200.

Dongare’s input queue was obviously a first-in-first-out (FIFO) buffer because Dongare teaches that input command queues were known to be “FIFO buffers.” Ex-1005, 2:61-67. Gewirtz also teaches that queues were “first-in-first-out (FIFO) buffers.” Ex-1006, 1:12-16, 11:7-9; Ex-1003, ¶¶151-52.

5. Claims 5, 11, 17

[5/11/17]. The system of claim [1/7/13], wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.

Ground 1 teaches claims 1/7/13. *Supra* §V.A.3, *infra* §§V.A.7, V.A.8; *supra* §V.A.3[1e] (motivation subsection); Ex-1003, ¶¶153-154, 181, 201.

Gewirtz teaches that “[t]he available reservation count is maintained by hardware: it is **decremented when a successful reservation read request is processed.**” Ex-1006, 4:56-5:6. Gewirtz’s “decrement logic” decrements the Gewirtz’s Queue State Register’s value for the number of free entries.

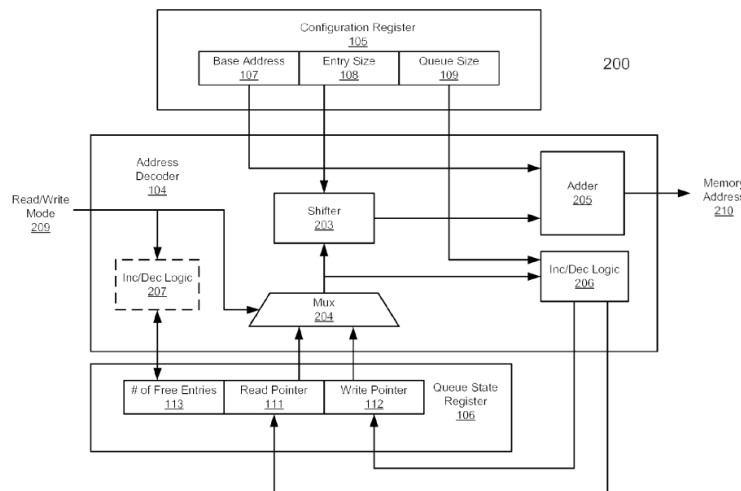


FIG. 2

Ex-1006, 6:53-54, Claim 4; Ex-1003, ¶¶154-157.

6. Claims 6, 12, 18

[6/12/18]. The system of claim [1/7/13], wherein at least one third processing element executes said command and notifies said output queue to issue said response.

Ground 1 teaches claims 1/7/13. *Supra* §V.A.3, *infra* §§V.A.7, V.A.8; *supra* §V.A.3[1e] (motivation subsection); Ex-1003, ¶¶158-159, 182, 202.

At least one third processing element (one or more RAID Engines) executes the command. Dongare's "**RAID engines 730 reads commands from input queues 750 and after processing them (e.g., read/write operations from Hard Disks 770), writes the processed commands back to output queues 780.**"

Ex-1005, 14:46-49. This write to the output queue is a notification to the embedded processor to initiate a process to issue a response (e.g., the fully complete and sequenced output) to the host. Ex-1005, 14:46-55. When the RAID engines finish the command and write the results to the output queue, the embedded processor re-sequences the results if necessary. The host processor then receives the response from the output queue. Ex-1005, 14:49-55; Ex-1003, ¶¶159-160.

7. Claim 7

Limitations [7pre]-[7c], [7e]-[7f], [7h]-[7i], and [7k] are substantively the same as corresponding limitations in claim 1 and taught for the same reasons. Any

differences are immaterial for determining patentability here. Ex-1003, ¶¶161-64, 168-69, 173-74, 179.

[7d] at least one second processing element in communication with said input queue;

Dongare teaches an embedded processor in communication with the input queues. “[T]he input queues 750 are read by the embedded processor 760 and commands are created for RAID engines 730 to process data.” Ex-1005, 14:42-44, Fig. 7. Dongare’s embedded processor receives the host processor’s commands via the input queue and schedules the commands for the RAID engines. Ex-1005, 7:40-64, 10:50-11:30, 14:39-55, 1:22-25, 9:5-9, 13:41-14:29; Ex-1003, ¶¶165-67.

[7g] receives a command from said first processing element;

Dongare teaches an embedded processor that receives a command from the host processor: “Host 710 writes commands for the RAID controller” that are then “read by the embedded processor 760” (Ex-1005, 14:41-44, Fig. 7), which performs subsequent steps as explained below (*infra* §V.A.7[7j]). Ex-1003, ¶¶170-72.

[7j] issues said command to said input queue on behalf of said first processing element; and

Dongare’s embedded processor receives a command from the host, tags it for scheduling purposes, and then writes (issues) the tagged command to the input

queue on behalf of the host, for execution by one or more RAID engines.

Ex-1005, 14:41-44, Fig. 7. **“The input queues 750 are read by the embedded processor 760 and commands are created for RAID engines 730 to process**

data.” *Id.* For longer commands, the embedded processor can divide the command into portions, tag the portions, and issue the command—including all of its portions—to the input queue. Ex-1005, 7:40-64, Fig. 3. The host writes the command, which is then processed and scheduled on the RAID engines by the embedded processor. Ex-1005, 7:40-64, 10:50-11:30, 14:39-55; Ex-1003, ¶¶175-178.

8. Claim 13

Limitations [13pre]-[13c] and [13e]-[13j] are substantively the same as corresponding limitations in claims 1 and 7 and taught for the same reasons. Any differences are immaterial for determining patentability here. Ex-1003, ¶¶183-86, 191-96.

[13d] at least one second processing element in communication with said input queue and said output queue;

Dongare’s embedded processor is in communication with the input queue (*supra* §V.A.7[7d]) and output queue. The “embedded processor 760 monitors output queues 780,” reorders output from the RAID engines as necessary, and signals the host “by writing into the outbound queue 780.” Ex-1005, 14:42-54, Fig. 7. The embedded processor is in communication with the output queue to

monitor and write to it. *Id.*; Ex-1005, 7:40-64; 10:50-11:30; 14:39-55.; Ex-1003, ¶¶187-90.

[13k] receives a response corresponding to said command from said output queue.

Dongare’s embedded processor receives a response corresponding to the Host’s command from the RAID engines. The “embedded processor 760 **monitors output queues 780** for completion of RAID commands in the same order as was loaded in input queues 750. **After completing the Host command, the embedded processor 760 will signal the Host 710 that the command is complete by writing into the outbound queue 780.**” Ex-1005, 14:49-54, Fig. 7; Ex-1003, ¶¶197-99.

B. Ground 2

Matsunami provides storage-area networking (“SAN”)/network-attached storage (“NAS”) teachings for dependent claims. The motivation to combine is explained below. *Infra* §V.B.2; Ex-1003, ¶203.

1. U.S. Patent No. 6,810,462 (“Matsunami”)

Matsunami teaches using “RAID” (Ex-1007, 7:13-17) in a combined “SAN”/“NAS” controller. Ex-1007, 2:13-15, Fig. 1; Ex-1003, ¶¶206-207.

Matsunami is prior art under §§102(a)(1)-(2). Ex-1003, ¶¶204-05.

Matsunami is analogous art in the same field (*supra* §III) because it describes a

storage controller system/architecture based on multiple processing elements.

Ex-1007, 3:52-65; Ex-1003, ¶208.

2. Claims 3-4, 9-10, 15-16

[3/9/15]. The system of claim [1/7/13], wherein said system is part of a storage area network (SAN) controller.

[4/10/16]. The system of claim [1/7/13], wherein said system is part of a network attached storage (NAS) controller.

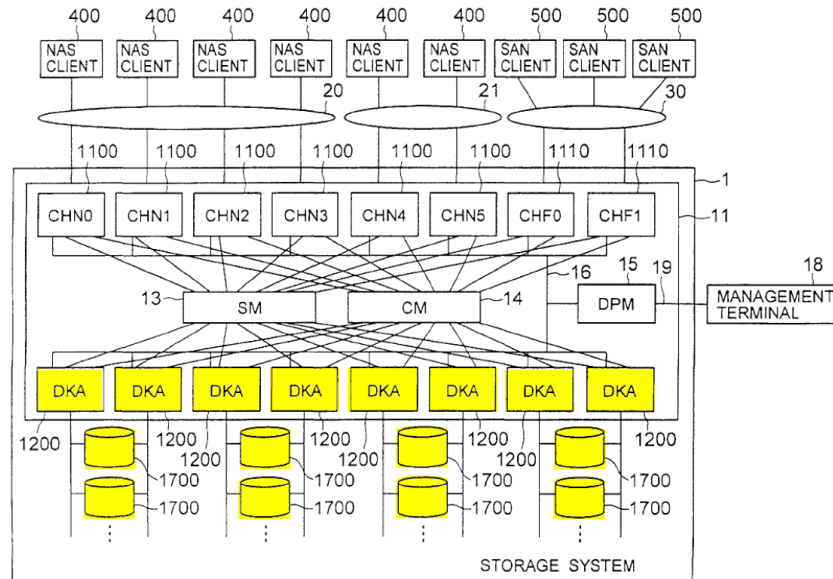
Ground 1 teaches claims 1/7/13. *Supra* §§V.A.3, V.A.7-8. Ground 2 applies Matsunami's teachings of a SAN/NAS controller. Ex-1007, Abstract (storage system with "NAS and SAN functions"), 1:18-23 (SAN), 1:30-34 (NAS), 2:1-4 (SAN/NAS complementary), 2:13-27 (combined SAN/NAS); Ex-1003, ¶209.

For Matsunami's "storage system having the NAS [and SAN] functions... the system user must manage the storage capacity for each storage system." Ex-1007, 2:20-23. Matsunami teaches using RAID for storage. Ex-1007, 7:1-17. Therefore, it was obvious to use RAID teachings from the Dongare-Gewirtz combination as part of Matsunami's SAN/NAS controller. Ex-1005, 14:30-55; Fig. 7; Ex-1003, ¶210.

For example, Matsunami's SAN/NAS controller comprises a Disk Pool Manager ("DPM") with a host processor and multiple disk adapters ("DKA"), each

providing RAID functionality. Ex-1007, 3:42-47 (combined SAN/NAS controller), 4:13-20 (disk pool manager), 3:66-4:1 (disk adapters), 7:13-17 (RAID).

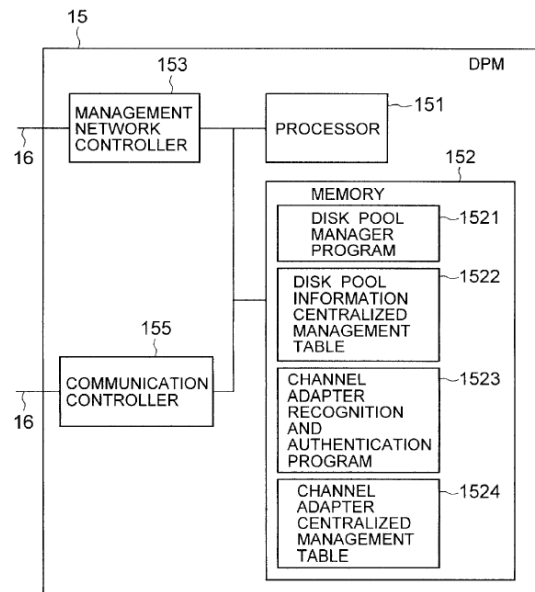
FIG. 1



Ex-1003, ¶211.

Matsunami's DPM contains a processor that "controls the overall operation of the disk pool manager[.]" Ex-1007, 3:66-4:1, 6:29-47.

FIG. 7



Matsunami teaches a SAN/NAS controller based on RAID hard disks (Ex-1007, 7:13-17) with a centralized DPM controller that would act as Dongare's host processor issuing read/write operations to the disks. See Ex-1005, 14:36-49; Ex-1003, ¶212.

It was obvious to apply Ground 1's teachings to Matsunami, with Dongare's host processor teachings applied to Matsunami's DPM and Dongare's RAID ASIC teachings applied to Matsunami's DKAs. Ex-1005, 14:30-37, Fig. 7; Ex-1003, ¶¶213, 222-23.

Motivation to Combine. The motivation to combine Dongare with Gewirtz was explained above. *Supra* §V.A.3[1e] (motivation subsection). Ex-1003, ¶214.

Matsunami teaches a SAN/NAS controller with RAID storage. Ex-1007, Abstract, 1:12-35, 6:66-7:17. Dongare teaches a RAID storage controller with improved efficiency and increased reliability. Ex-1005, 14:30-55 (for “applications such as RAID controllers”), Fig. 7. A POSITA would have been motivated to use Matsunami’s SAN/NAS teachings with Dongare’s RAID architecture; this was been a simple combination of prior art elements according to known methods to yield predictable results (a SAN/NAS storage controller using RAID). Ex-1003, ¶¶215-217.

A POSITA would have had a reasonable expectation of success, given Matsunami’s suggestion of including RAID (Ex-1007, 6:66-7:17), which would not have required specialized skills beyond a POSITA’s knowledge. By 2013, RAID was commonplace in servers, workstations, and data centers. Ex-1003, ¶¶218-20.

The combination would not have changed the principle of operation for any reference. Dongare’s RAID teachings would not fundamentally change the character of Matsunami’s teachings or the application domain of storage control systems. Ex-1003, ¶220.

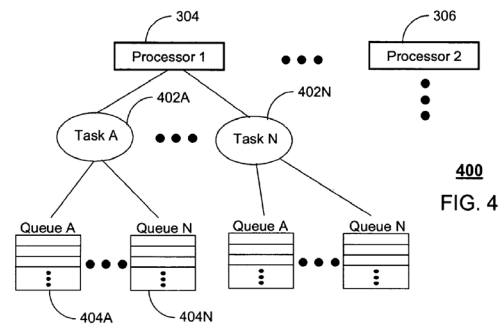
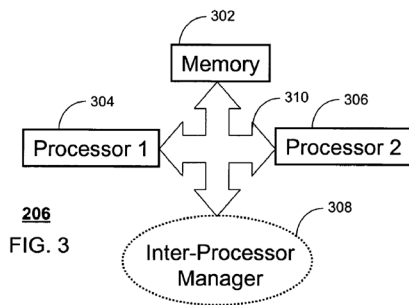
C. Ground 3

Khawand teaches a multiprocessor architecture that improves efficiency using delegation, queues, and flow control. Ex-1008, 1:8-48, 3:65-4:5. Rosenthal

provides details for using queues in multiprocessor systems with flow control, including a register that monitors available space. Ex-1009, 2:15-33, 16:33-17:18. Ground 3 combines Rosenthal with Khawand. The motivation is explained below. *Infra* §V.C.3[1e] (motivation subsection); Ex-1003, ¶¶224-25.

1. U.S. Patent No. 7,398,528 (“Khawand”)

Khawand teaches a multiprocessor architecture where service applications are delegated between the processors using queues; an inter-processor manager implements flow control between the processors. Ex-1008, 3:65-4:27.



Ex-1008, 3:18-22; Ex-1003, ¶228.

Khawand uses queues to delegate service applications from one processor to another when a processor runs out of processing capacity. Ex-1008, 3:52-4:5.

Khawand teaches using flow control to avoid queue overflow. Ex-1008, 4:15-22; Ex-1003, ¶¶229-30.

Khawand is prior art under §§102(a)(1)-(2). Ex-1003, ¶¶226-27. Khawand is analogous art in the same field (*supra* §III) because Khawand teaches a

computer architecture with multiple processors and queues to manage service application delegation among the processors. Ex-1008, 1:32-48; Ex-1003, ¶231.

2. U.S. Patent No. 5,805,930 (“Rosenthal”)

Rosenthal teaches “flow control for a computer system” “utilizing FIFO buffers[.]” Ex-1009, 2:4-7. Rosenthal teaches “a flow control register” that “stores a value which indicates the amount of space available in the FIFO buffer[.]” Ex-1009, 2:21-28.

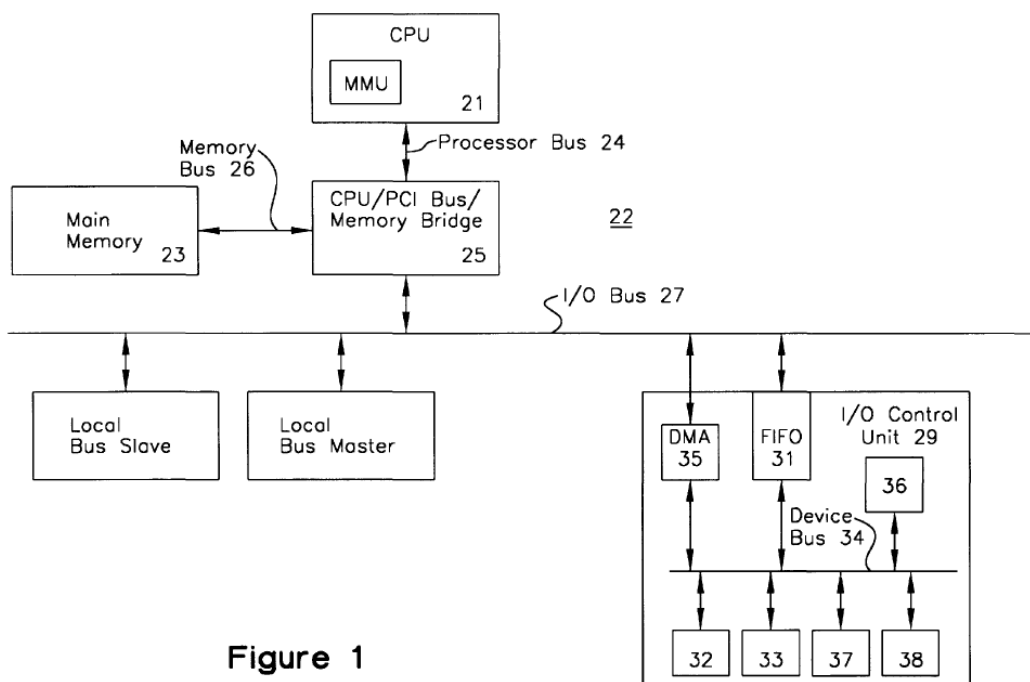


Figure 1

Ex-1003, ¶234.

Rosenthal is prior art under §§102(a)(1)-(2). Ex-1003, ¶¶232-33. Rosenthal is analogous art in the same field (*supra* §III) because Rosenthal teaches queue

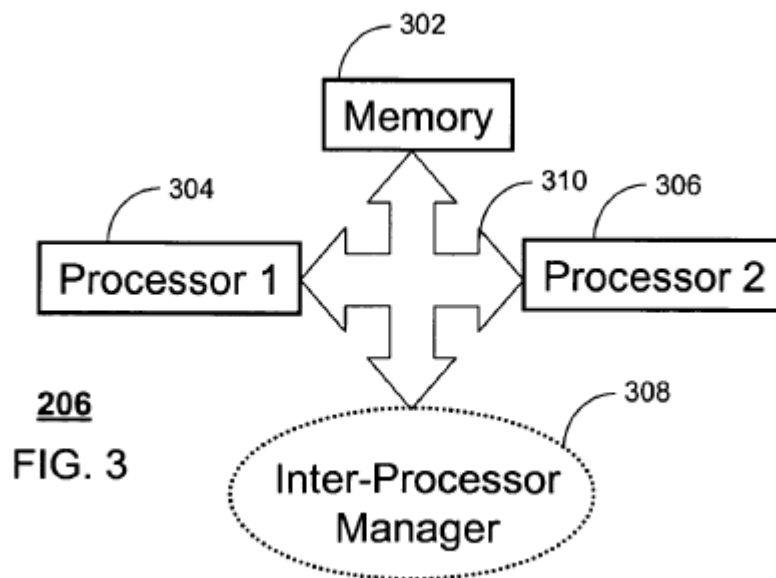
management and flow control for microprocessor systems. Ex-1009, 2:11-17;
Ex-1003, ¶235.

3. Claim 1

[1pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:

Khawand teaches a multiprocessor system for dividing and synchronizing a processing task across a plurality of processing elements. Khawand's “**multi-processor architectures**” includes multiple ARM processors and an Inter-Processor Manager that communicate using “conventional inter-processor protocols.”

Ex-1008, 1:20-24.



Khawand teaches distributing service applications and sub-tasks among the various processors for enhanced performance, with synchronization of service application delegation using queues. Ex-1008, 3:62-4:9, 4:60-67 (suspend command), 5:18-29

(resume command), Fig. 4. Khawand teaches “service applications” including “software algorithms or portions thereof **subdivided into tasks...**” Ex-1008, 3:4-17, Fig. 6; Ex-1003, ¶¶236-37.

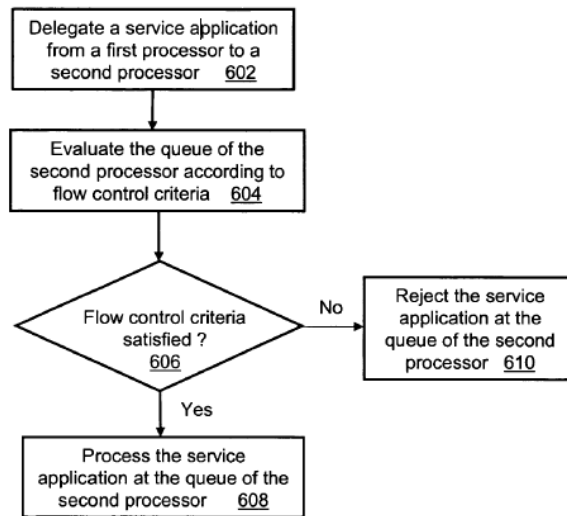
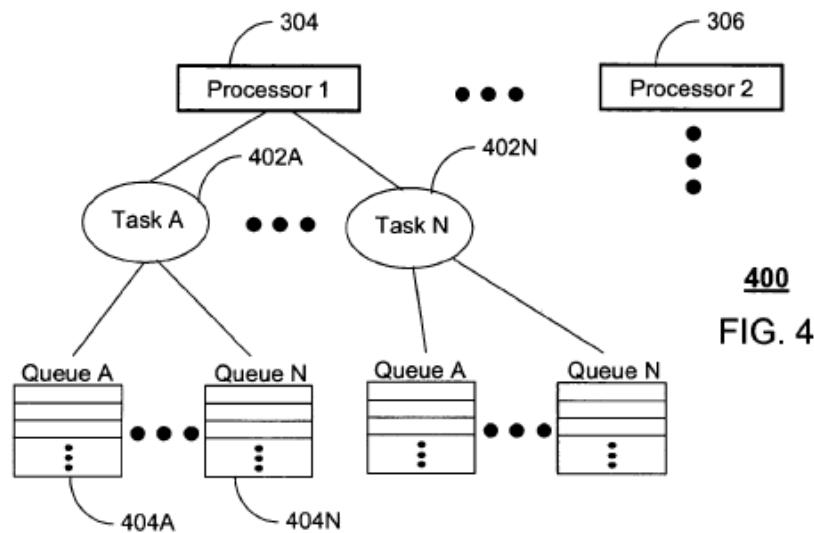


FIG. 6 600

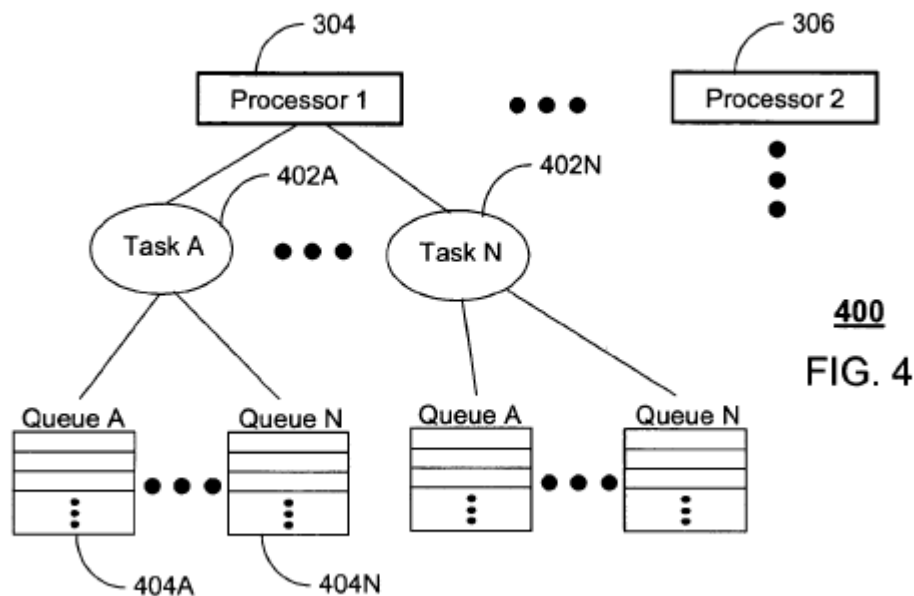


400
FIG. 4

Khawand's Inter-Processor Manager uses flow control to manage queues and keep the system synchronized. Ex-1008, 4:60-67 (“...synchronizes the first processor 304...”); Ex-1003, ¶¶236-37. The system further comprises the elements explained below. *Infra* §§V.C.3[1a]-[1j].

[1a] an input queue implemented in hardware;

Khawand teaches a hardware input queue (e.g., one of Processor 2's input queues) to hold service applications for a task:



“[E]ach task 402 comprises at least one queue 404 for processing at least one service application.” Ex-1008, 3:56-59. Khawand's queues are implemented “in **hardware**, software, or a combination[.]” Ex-1008, 6:14-31; Ex-1003, ¶¶238-39.

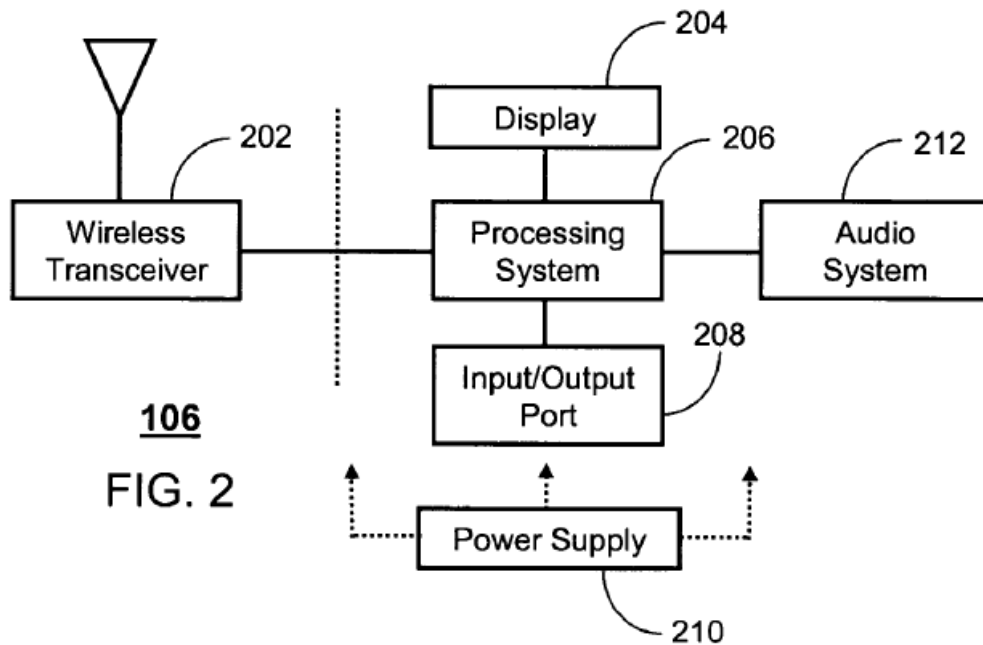
Rosenthal teaches using “hardware FIFO” queues for transferring commands. Ex-1009, 4:60-64, Fig. 1, 8:28-31, Fig. 3. It was obvious to use a “hardware FIFO” for Khawand’s input queue. *Infra* §V.C.3[1e] (explaining motivation to combine). The input queue is implemented in physical circuits, without software intervention, because both Khawand and Rosenthal teach hardware implementations. Ex-1008, 6:14-34; Ex-1009, 4:60-64; Ex-1003, ¶¶240-241.

Additionally, to the extent PO somehow argues that Khawand or Rosenthal use memory, PO alleged in district court that a queue implemented in memory using software is a queue implemented in hardware. Ex-1017, 000003-14.

[1b] an output queue implemented in hardware;
--

Khawand’s processors delegate service applications and therefore teach or suggest a mechanism for returning resulting data back, e.g., for service applications delegated from Processor 1 to Processor 2. *See* Ex-1008, Abstract, 1:53-2:9, 3:65-4:5, Fig. 6. Khawand’s processors execute various “service applications” where output is expected, including “multi-media processing,” “MPEG 4 video decoder,” “input/output drivers, display drivers, etc.” with return data for the display, audio system, and/or input/output port. Ex-1008, 3:4-17. Khawand’s mobile device contains multiple hardware elements that require buffering, e.g., file-storage device and network interfaces to handle data transfer correctly and

efficiently, in addition to buffering needed to pass data between service applications running on multiple processors.



Ex-1003, ¶¶244-45.

During EU prosecution, the Examiner stated that “[i]t is common general knowledge of the skilled person [] that an **output queue** [] is an implementation choice for buffering request results from a storage device.” Ex-1004, 000119. Output queues have been used for return data between functional units since the early days of computers; a POSITA with knowledge of basic logic/computer design would have found it obvious to make this simple implementation choice and use an output queue to return results for Khawand. Ex-1003, ¶246.

Additionally, it was obvious for Khawand to use output queues for buffering, as the returned data is either consumed by the delegating processor or destined for another function, such as the display, audio system, or I/O port. *See* Ex-1008, 3:4-17, Fig. 2. Buffers have been used for this purpose—to move data between computer components—for decades. Moreover, it was obvious to implement the output queue in physical circuits, without software intervention, because both Khawand and Rosenthal teach hardware implementations for queues. Ex-1008, 6:14-17; Ex-1009, 4:60-64; Ex-1003, ¶246.

The motivation to use Rosenthal’s hardware FIFO teachings is further explained below. *Infra* §V.C.3[1e] (motivation subsection); Ex-1003, ¶247.

Additionally, to the extent PO somehow argues that Khawand or Rosenthal use memory, PO alleged in district court that a queue implemented in memory using software is a queue implemented in hardware. Ex-1017, 000003-14.

[1c] a first processing element having access to said input queue and said output queue;
--

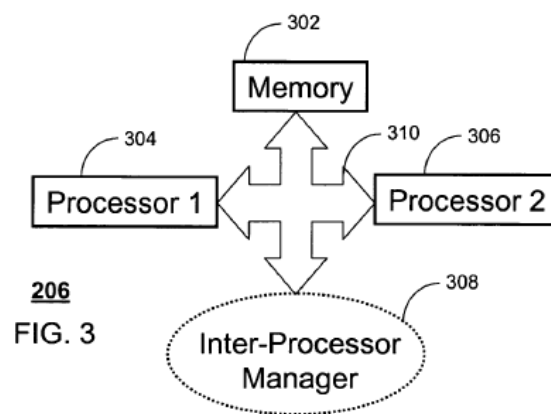
Khawand’s Processor 1 is a first “processing element[]” (Ex-1008, 3:37-39) that has access to the input queue and output queue. Processor 1 delegates service applications to Processor 2 by writing service applications into Processor 2’s input queue. Ex-1008, 3:52-4:5, Fig. 4. Khawand also teaches or suggests an output queue for returning results from delegated service applications. *Supra* §V.C.3[1b].

Processor 1 would obviously have had access to the output queue because the purpose of the delegation was for Processor 2 to process the service application and return the data back to Processor 1. That is how output queues have been used for decades. Ex-1003, ¶¶250-54.

[1d] at least one second processing element in communication with said first processing element;

Khawand's Inter-Processor Manager 308 is a second "processing element[]" (Ex-1008, 3:37-39) that comprises an ASIC or "microprocessor." Ex-1008, 3:29-36; Ex-1003, ¶¶255-56.

The Inter-Processor Manager is in communication with Processor 1 because "the processing elements 302-308 ... are interconnected by a conventional communication bus 310." Ex-1008, 3:37-47; Ex-1003, ¶255.



[1e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;
--

Khawand's Inter-Processor Manager implements flow control for the processors' queues. Ex-1008, Abstract, 4:6-27, 5:18-29, Fig. 6. Rosenthal teaches flow control details. For example, Rosenthal teaches a reservation register, called a "flow control register," "for each FIFO buffer." Ex-1009, 16:33-37. "The register 45 **stores an indication of the number of available spaces for commands** in the associated FIFO buffer 39." *Id.* Rosenthal's flow control register is implemented in hardware because it is a physical register that stores values, accessible by the CPU. Ex-1009, 16:33-49;¹¹ Ex-1003, ¶¶258-259.

Rosenthal teaches that the number of available spaces tracked in the flow control register must be read before any commands are written. Ex-1009, 16:37-49. Software reads the flow control register first to determine the amount of free space in the FIFO, and can only write up to that amount of data, before it must read the register again for subsequent writes in excess of the free space from the previous read. *Id.* Quite simply, Rosenthal teaches flow control using a register

¹¹ Rosenthal discloses an optional implementation using bit-fields. Ex-1009, 16:50-57; Ex-1003, n.6. It is not relied on for these Grounds.

that stores a value indicative of available space in the queue, and software is limited to writing that amount before it must check again. *Id.* Ex-1003, ¶260.

The flow control register is accessible by the first and second processing elements. Rosenthal teaches that the flow control register being located in a manner accessible by any unit attached to the I/O bus:

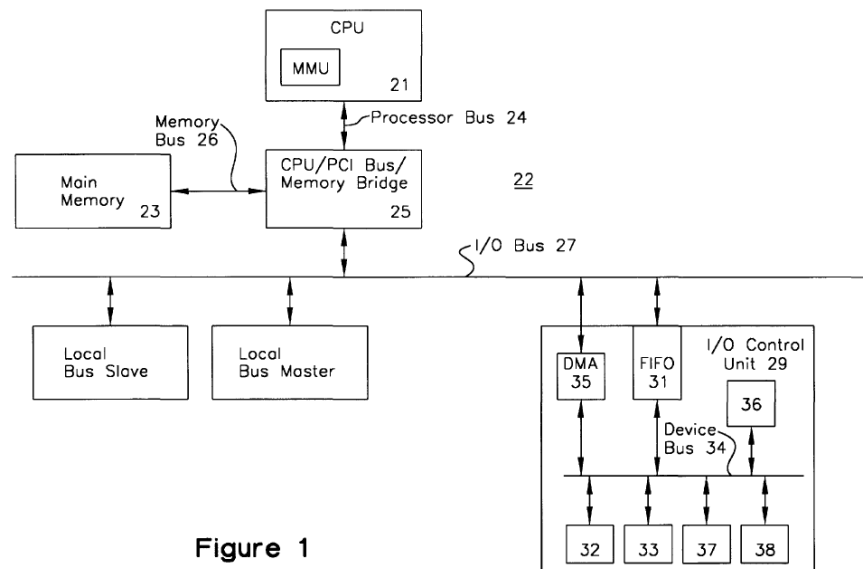


Figure 1

Ex-1009, 4:49-59. Rosenthal teaches that all devices on the I/O bus access the I/O control unit and use the flow control register to reserve space in the FIFO unit. *Id.* Similarly, Khawand's Processors and Inter-Processor Manager are connected to the "communication bus." Ex-1008, 3:37-39. Therefore, in the Khawand-Rosenthal combination, the Processors and Inter-Processor Manager would have had access to the flow control register, as Rosenthal teaches or at least suggests for the combination. Ex-1009, 4:49-59, Fig. 1; Ex-1003, ¶¶261-262.

Khawand's processing elements are processors (Ex-1008, 3:27-32), which obviously would have had access to the flow control register because its purpose was to provide flow control for the queue. Rosenthal explains that the flow control register is read by a processor the "central processing unit running the application" (Ex-1009, 2:21-26) and that after the value has been read, "the application program" knows how much free space is in the input queue. Ex-1009, 16:33-49; Ex-1003, ¶263.

Rosenthal's flow control register is a memory-mapped register, visible to and accessible to software applications running on processors. Ex-1009, 16:33-49. A POSITA would have understood this because that is how software applications access such registers; this has been a ubiquitous technique for decades in computer architecture. It was obvious for the flow control register to be similarly accessible by all processors in Khawand's multiprocessor system for flow control purposes, particularly since Rosenthal's express purpose is "a means for providing flow control for a computer system or similar system utilizing FIFO buffers to receive data so that the operation may proceed as rapidly as possible without loss of data." Ex-1009, 2:4-7. Conventional multiprocessor systems have included hardware-based memory-mapped registers for decades. Ex-1003, ¶263.

Ground 3 teaches both parties' district court constructions. For PO's construction (Ex-1029, 000002), Rosenthal's flow control register is accessible to

the microprocessor via a memory address and stores a value, e.g., the number of available spaces for commands in the associated FIFO buffer, as explained above. For Petitioner's district court construction, Rosenthal teaches a register as a set of flip-flops that store a set of related bits for indicating the number of available spaces for commands. Ex-1009, 22:11-13 (teaching that registers hold bits), 22:23-26 (same). A POSITA would have understood that a register was a set of flip-flops because this was taught in undergraduate logic design classes for decades. *See* Ex-1019, 000013; Ex-1020, 000012; Ex-1021, 000012; Ex-1022, 000008; Ex-1023, 000025; Ex-1024, 000026 (same definition); Ex-1003, ¶112. Rosenthal's flow control register is a set of flip-flops storing a set of related bits, e.g., indicating the number of available spaces for commands in the associated FIFO buffer. Ex-1009, 16:35-37, Fig. 3; Ex-1003, ¶¶263-264.

Motivation to Combine. Khawand teaches a multiprocessor architecture with flow control to prevent overflow of service application queues. Ex-1008, 4:15-27, Abstract. Khawand's Inter-Processor Manager evaluates "flow control criteria" including "capacity" for each queue. Ex-1008, 4:6-27. Rosenthal provides details for accomplishing this flow control, including teachings for hardware FIFOs and a "flow control register" that indicates the amount of space remaining in the queue. Ex-1009, Abstract. A POSITA would have been motivated to apply Rosenthal's teachings to Khawand. Ex-1003, ¶265.

Rosenthal explains that “any practical input/output device will have limited FIFO buffer storage for holding data written to it” and therefore “any architecture for input/output devices must include some technique for controlling the flow of data so that this storage is not exhausted” motivating the combination. Ex-1009, 1:66-2:7. Rosenthal’s solution to this known problem is flow control: “[i]t is another object of the present invention to provide a **flow control system** for an architecture which utilizes **FIFO buffers** for buffering write operations.” Ex-1009, 2:15-17. Rosenthal’s “flow control register” contains the number of available spaces in the FIFO. Reading this register before adding new data to the FIFO prevents overflow. Ex-1009, 16:33-37. The combination would have been a combination of known prior art elements according to known methods to yield a predictable result—queue flow control based on a register indicating the amount of space in the queue, as described by Rosenthal. Ex-1003, ¶266.

A POSITA would have been motivated by the benefits of the combination. Rosenthal’s register provides a simple mechanism that prevents processors from overflowing a queue, allowing Khawand’s processors to quickly verify that space is available before attempting to write to an input queue. Ex-1009, 16:33-49 (flow control register); Ex-1008, 4:15-27 (evaluating flow control register). The references provide motivation to combine their teachings. Ex-1008, 6:31-34; Ex-1009, 23:64-24:1; Ex-1003, ¶¶267-268.

A POSITA would have been motivated to use Rosenthal's hardware FIFO buffers as input queues with Khawand. Khawand teaches a hardware implementation for its queues. Ex-1008, 6:14-34. Rosenthal's hardware FIFO buffers provided faster access time while enabling atomic queue operations. Moreover, a mix of fast hardware queues would have provided flexibility, with the ability to exploit temporal locality in determining which queues for each task should be in hardware, while still allowing overhead queues in memory. Khawand itself suggests a hybrid approach. Ex-1008, 6:14-17. The combination simply combines known prior art elements according to known methods to yield a predictable and straightforward result—hardware FIFO buffers operating as input or output queues. Ex-1003, ¶¶268-69.

A POSITA would have had a reasonable expectation of success in combining Khawand and Rosenthal. Khawand uses queues and flow control. Ex-1008, 3:52-61 (queues), 3:62-4:5 (flow control), Fig. 6. Rosenthal teaches specific flow control details. Ex-1009, Abstract, 2:15-17, 2:21-26, 16:33-57. The combination would have been a straightforward application of these well-known techniques, e.g., hardware FIFOs, flow control, registers, and queue management to improve Khawand's architecture with a predictable result. These techniques were included in undergraduate education, and a POSITA would have been aware

of them. Adding Rosenthal’s teachings regarding hardware FIFOs and a flow control register would have been simple and straightforward. Ex-1003, ¶¶270-71.

Furthermore, the combination would not have changed the principle of operation for any reference. Khawand would still have multiple input queues for distributed service application processing. Rosenthal’s hardware FIFOs, flow control mechanisms, and flow control register would still serve the same purpose to buffer data and manage queues. Ex-1003, ¶¶272-73.

[1f] computer storage storing instructions, which when executed by said at least one second processing element:
--

Khawand teaches implementing the second processing element (e.g., Inter-Processor Manager) as a “microprocessor” (Ex-1008, 3:29-36) and implementing its embodiments generally “in hardware, **software**, or a **combination of hardware and software**.” Ex-1008, 6:14-17. Khawand’s Inter-Processor Manager executes software instructions to provide the functionality described, including for [1g]-[1j] below. These instructions were obviously stored on computer storage, e.g., memory, because that is how computers have operated since their inception. Ex-1003, ¶¶274-76.

[1g] accesses said reservation register and reads said stored value;

Khawand’s Inter-Processor Manager accesses the reservation register and reads the stored value to implement flow control for the input queue. The flow

control register contains the “number of available spaces for commands in the associated FIFO buffer.” Ex-1009, 16:35-49. “[T]he central processing unit read[s] the value in the flow control register 45 associated with the FIFO buffer 39 being utilized.” Ex-1009, 16:44-46; Ex-1003, ¶¶277-78.

In the combination of Ground 3 (*supra* §V.C.3[1e] (motivation subsection)), Khawand’s Inter-Processor Manager evaluates the flow control criteria of the second processor’s queue before the first processor can delegate a service application to it.

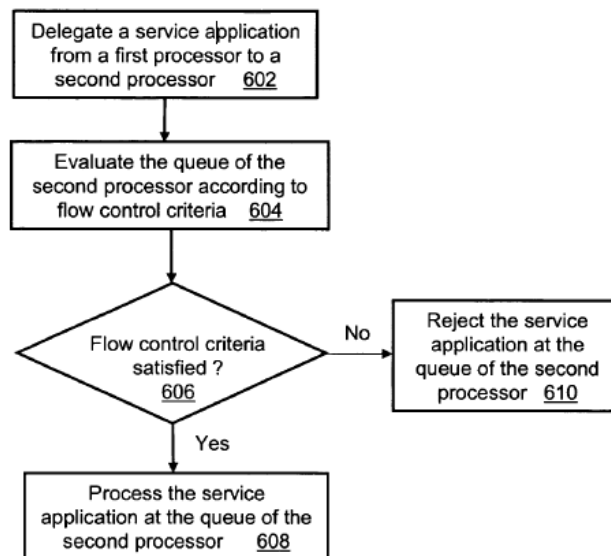


FIG. 6 600

Khawand’s Inter-Processor Manager executes step 604, where “the queue 404 of the second processor 306 is **evaluated according to flow control criteria.**”

Ex-1008, 4:6-9 (“In a first embodiment, the **evaluation step can take place at the**

inter-processor manager 308.”). Khawand’s flow control criteria is based on whether the queue has sufficient capacity. Ex-1008, 4:15-27, 5:50-54; Ex-1003, ¶279.

The flow control register provides a mechanism for Khawand’s Inter-Processor Manager to evaluate flow control criteria, including capacity. *Supra* §V.C.3[1e]. In the combination, Khawand’s Inter-Processor Manager accesses the flow control register, reads how many available entries are in the input queue (Ex-1008, 4:6-27; Ex-1009, 16:35-37), and makes a determination if there is enough space. This is Rosenthal’s stated purpose for the flow control register. Ex-1009, 16:35-46; Ex-1003, ¶¶280-81.

[1h] determines when said read value indicates available space in said input queue for said first processing element to issue a command;
--

In the combination of Ground 3 (*supra* §V.C.3[1e] (motivation subsection)), Khawand’s Inter-Processor Manager accesses the flow control register, reads the “number of available spaces for commands in the associated FIFO buffer” (*see* Ex-1009, 16:35-46; *supra* §V.C.3[1g]), and then determines if there is enough space in the input queue for the command as Rosenthal teaches. Ex-1009, 16:35-49; Ex-1003, ¶¶282-83.

Khawand’s Inter-Processor Manager evaluates flow control on the basis of available space in Processor 2’s input queue. Ex-1008, 4:15-27; Ex-1008, 4:7-9

(“... the evaluation step can take place at the inter-processor manager.”). In the combination, this is performed by reading the value from the flow control register, evaluating the flow control criteria (capacity not exceeded), and determining if there are available spaces for commands to be added to the input queue for Processor 2 (*see* Ex-1009, 16:35-46), so that Processor 1 can delegate service applications to Processor 2. Ex-1008, 3:62-4:5; *infra* §V.C.3[1i]; Ex-1003, ¶¶284-85.

[1i] notifies said first processing element to issue said command to said input queue; and

The combination of Ground 3 was explained above. *Supra* §V.C.3[1e] (motivation subsection).

Khawand’s Processor 2 sends a suspend command to the Inter-Processor Manager when its queue is full, indicating that the associated queue is not available. Ex-1008, 4:60-67, 5:1-17 (“... suspend command corresponds to... all tasks 402 and associated queues 404 would no longer be available to process service applications...”). When the input queue becomes available again, the Inter-Processor Manager notifies the first processor of the availability through a resume command. Ex-1008, 5:1-29. Khawand’s Inter-Processor Manager executes this algorithm to “provide[] a stable environment for managing queues

404 using suspend and resume commands[.]” Ex-1008, 5:30-41; Ex-1003, ¶¶286-288.

Additionally, Khawand’s Inter-Processor Manager sends queue capacity information to Processor 1, so that it can issue commands. Ex-1008, 5:50-63; *infra* §V.C.3[1j]. Therefore, Khawand teaches, or at least suggests, that the Inter-Processor Manager notifies the first processor to issue the command to the input queue, when the queue is no longer full and is available to receive service applications. *See* Ex-1008, 5:50-63; Ex-1003, ¶¶288-89.

[1j] wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue.
--

Khawand’s processors “are programmed to delegate a service application from the first processor 304 to a queue 404 of the second processor 306.” Ex-1008, 3:65-4:2. As explained above, Khawand’s Inter-Processor Manager sends a suspend command to the first processor when the input queue is full, followed by a resume command when space becomes available. *Supra* §V.C.3[1i]. After receiving the notification (e.g., resume command), Khawand’s Processor 1 delegates its service application by issuing the command to the input queue of Processor 2. Ex-1008, 3:65-4:22; Ex-1003, ¶¶290-92.

Khawand's Processor 1 receives a response corresponding to said command from said output queue. Khawand's Processor 1 "delegat[es] a service application from a first processor" to a "queue of a second processor." Ex-1008, 1:39-48, 1:49-61 (same), 1:62-2:9 (same). It was obvious to include an output queue to return processed data from the delegated service application—that is the purpose of delegation. *Supra* §V.C.3[1b]. Therefore, a POSITA would have found it obvious for Processor 1 to receive a response corresponding to its delegated command from the output queue. Ex-1003, ¶¶293-94.

4. Claims 2, 8, 14

Ground 3 teaches claims 1/7/13. *Supra* §V.C.3; *infra* §§V.C.6-7; *supra* V.C.3[1e] (motivation subsection); Ex-1003, ¶¶295-96, 324, 344.

Rosenthal's input queue is a "first-in first-out (FIFO)" buffer, used as a write buffer. Ex-1009, 1:44-50, 3:66-4:11, 4:61-64. Rosenthal teaches using FIFO write buffers in CPU, bridges, and I/O devices. Ex-1009, 1:50-53; Ex-1003, ¶¶297-98.

5. Claims 6, 12, 18

Ground 3 teaches claims 1/7/13. *Supra* §V.C.3; *infra* §§V.C.6-7; *supra* V.C.3[1e] (motivation subsection); Ex-1003, ¶¶299-300, 325, 345.

Khawand teaches wherein at least one third processing element (e.g., Khawand's Processor 2) executes said command and notifies said output queue to issue said response. Khawand explains that "the **service application is processed**

in step 608 by the second processor 306...” Ex-1008, 4:18-22. Khawand’s Processor 2 processes service applications delegated to it by processing commands from its input queue. Ex-1003, ¶301.

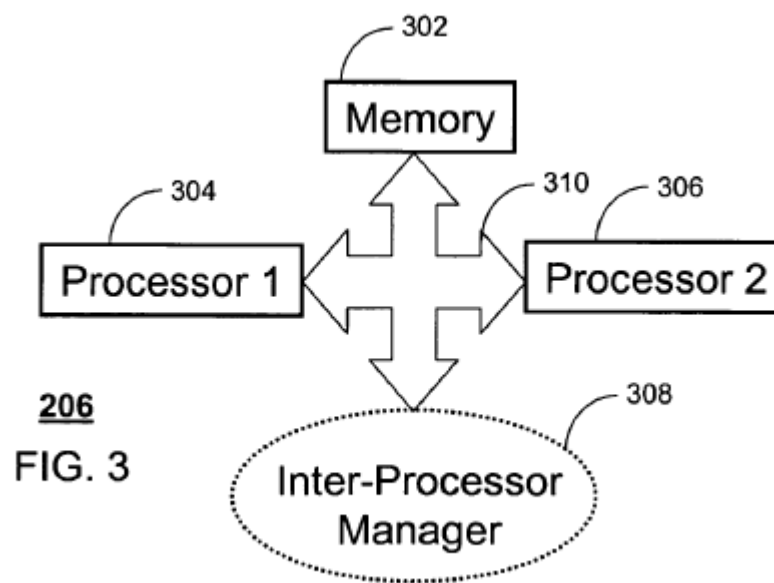
When done, the second processor provides its results back to the first processor—that is the point of delegation. Ex-1008, 3:65-4:22. It would have been obvious to use the output queue to return the results. *Supra* §§V.C.3[1b], [1j]. Accordingly, it was obvious for Processor 2 to provide the results from the delegated command to the output queue, and then notify the output queue to provide the response back to the first processor. Khawand teaches or suggests a mechanism for the processors to communicate with each other and to notify each other after completing their delegated service applications because they are connected by a communication bus. Ex-1008, 3:37-51, Fig. 3. Additionally, a POSITA would have also found it obvious for the output queue to notify the first processor of a result through a simple ready or queue not empty flag. Ex-1008, 3:37-51, Fig. 3; Ex-1003, ¶¶302-03.

6. Claim 7

Limitations [7pre]-[7c], [7e]-[7f], [7h]-[7i], and [7k] are substantively the same as corresponding limitations in claim 1 and taught for the same reasons. Any differences are immaterial for purposes of determining patentability here. Ex-1003, ¶¶304-07, 313-14, 318-319, 323.

[7d] at least one second processing element in communication with said input queue;

Khawand teaches an Inter-Processor Manager in communication with the input queues using a communication bus. Ex-1008, 3:65-4:22 (delegation), 3:37-51 (communication bus), Fig. 3. Additionally, the Inter-Processor Manager is in communication with the input queue (implemented in hardware *supra* §V.C.3[1a]) because it delegates service applications to the second processor by writing to the input queue. *Infra* §V.C.6[7j]; Ex-1003, ¶¶308-12.



[7g] receives a command from said first processing element;

Khawand teaches an Inter-Processor Manager performing the steps of Figure 6. Ex-1008, 4:10-14 (explaining how Figure 6 “can be... centralized in the inter-processor manager 308”). Figure 6 is a method to “delegate a service

application from the first processor 304 to a queue 404 of a second processor.”

Ex-1008, 3:65-4:2. When Figure 6 is centralized on the Inter-Processor Manager, it submits the command it received from the first processor (step 602 delegation) to the input queue of the second processor if the flow control criteria is satisfied (step 606). Ex-1008, Fig. 6 (step 602), 4:6-27. Before Khawand’s Inter-Processor Manager submits the command from the first processor, it must first receive the command from the first processor.

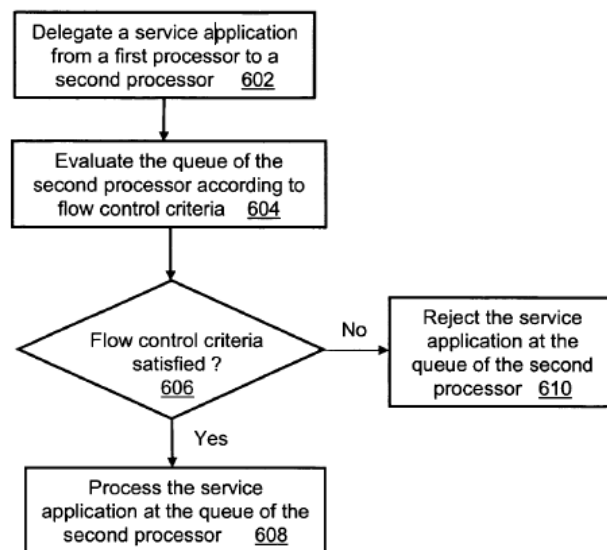


FIG. 6 600

Ex-1003, ¶¶315-17.

[7j] issues said command to said input queue on behalf of said first processing element; and

Khawand's Inter-Processor Manager performs the steps of Figure 6.

Ex-1008, 4:10-14 (explaining how Figure 6 “can be... centralized in the inter-processor manager 308”). Figure 6 teaches a method to “**delegate a service application from the first processor 304 to a queue 404 of a second processor.**”

Ex-1008, 3:65-4:2. Khawand's Inter-Processor Manager receives the command that Processor 1 wants to delegate to Processor 2. *Supra* §V.C.6[7g]. Khawand's Inter-Processor Manager then issues the delegated command it received, on behalf of Processor 1, to the input queue of Processor 2 when the flow control criteria is met. Ex-1008, Fig. 6, 3:65-4:27.

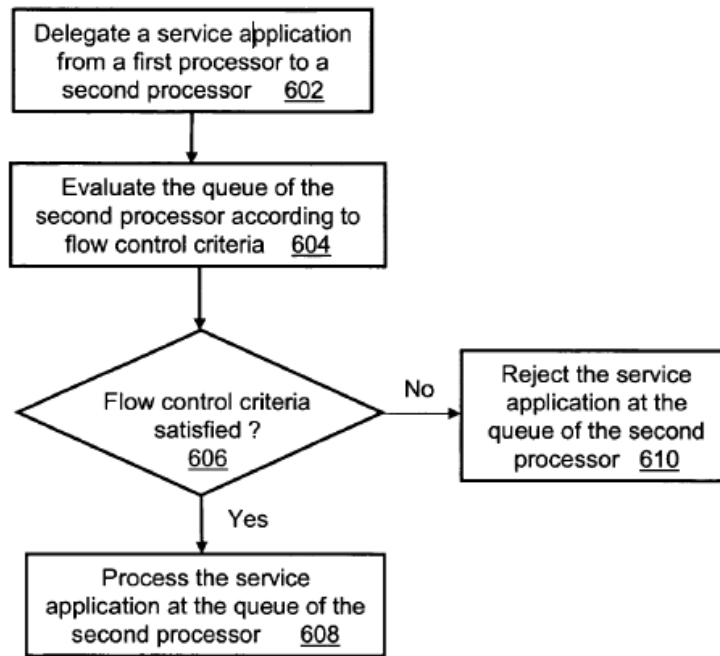


FIG. 6 600

Ex-1003, ¶¶320-22.

7. Claim 13

Limitations [13pre]-[13c] and [13e]-[13j] are substantively the same as corresponding limitations in claims 1 and 7 and are satisfied for the same reasons.

Any differences are immaterial for purposes of determining patentability here.

Ex-1003, ¶¶326-29, 335-40.

[13d] at least one second processing element in communication with said input queue and said output queue;

Khawand teaches an Inter-Processor Manager that uses the communication bus to delegate service applications to a given processor via input queues, including the input queue for Processor 2. *Supra* §V.C.6[7j]; Ex-1008, 3:65-4:22 (delegation), 3:37-51 (communication bus), Fig. 3. The Inter-Processor Manager communicates with the input queue using a communication bus (Ex-1008, Fig. 3) and is in communication with the input queue (implemented in hardware, *supra* §V.A.3[1a]) because it writes the delegated commands to the input queue. Ex-1003, ¶¶330-32.

Khawand's Inter-Processor Manager is also in communication with the output queue. Khawand teaches or suggests an output queue. *Supra* §§V.C.7[13b], V.C.3[1b], V.C.3[1j]. Khawand's Inter-Processor Manager has a centralized role in the delegation, synchronization, and coordination of service applications. *Supra* §V.C.6[7g]; Ex-1008, 4:10-14, 5:30-41. A POSITA would have found it obvious for Khawand's Inter-Processor Manager to communicate with the output queue from Processor 2 that is used to return data to the delegating processor due to its centralized role in facilitating service application delegation. *Infra* §V.C.7[13k]; Ex-1003, ¶¶333-34.

[13k] receives a response corresponding to said command from said output queue.
--

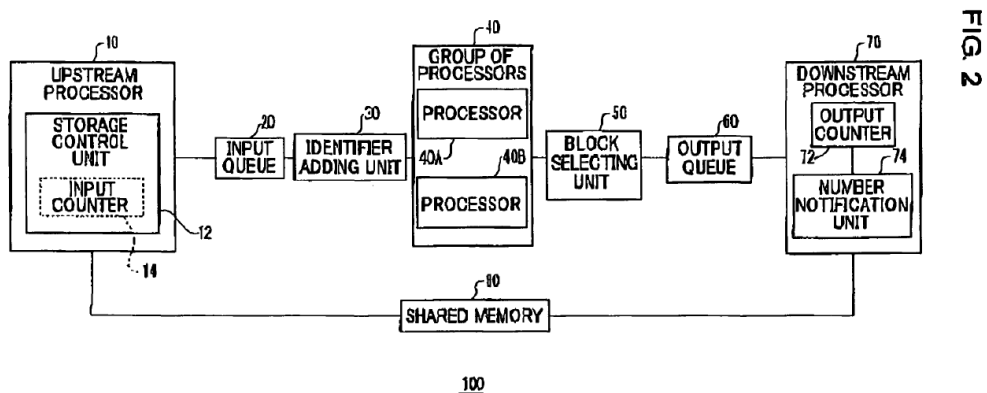
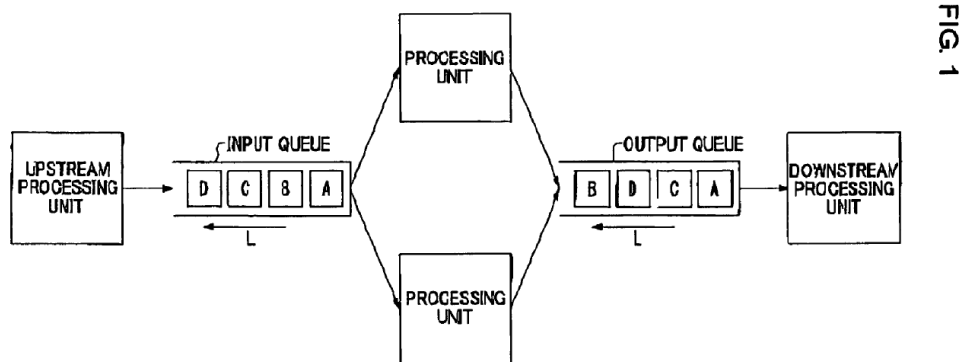
Khawand teaches delegation of service applications, and it was obvious to include an output queue for return data from delegated service applications. *Supra* §V.C.3[1b]. Accordingly, it was obvious for Khawand's Inter-Processor Manager to receive a response for the delegated command from the output queue, particularly when the Inter-Processor Manager plays a centralized role in the delegation, synchronization, and coordination of service applications. *Supra* §V.C.6[7g]; Ex-1008, 4:10-14, 5:30-41. Khawand's Inter-Processor Manager directly requests updates from each input queue, which teaches or at least suggests that the it would also be able to access the corresponding output queues and receive a response corresponding to the issued command. Ex-1008, 5:50-54; Ex-1003, ¶¶341-42.

D. Ground 4

To the extent PO argues that hardware output queues were somehow not obvious by 2013, Ohkawa teaches hardware output queues. Ex-1010, 4:15-17. Ground 4 combines the same teachings explained above for Ground 3 with Ohkawa's teachings for output queues. The motivation to combine is explained below. *Infra* §V.D.2[1b]; Ex-1003, ¶346.

1. U.S. Patent No. 8,028,284 (“Ohkawa”)

Ohkawa teaches a multiprocessor system with input queues and output queues for buffering between processing units.



Ex-1010, 5:7-13; Ex-1003, ¶349.

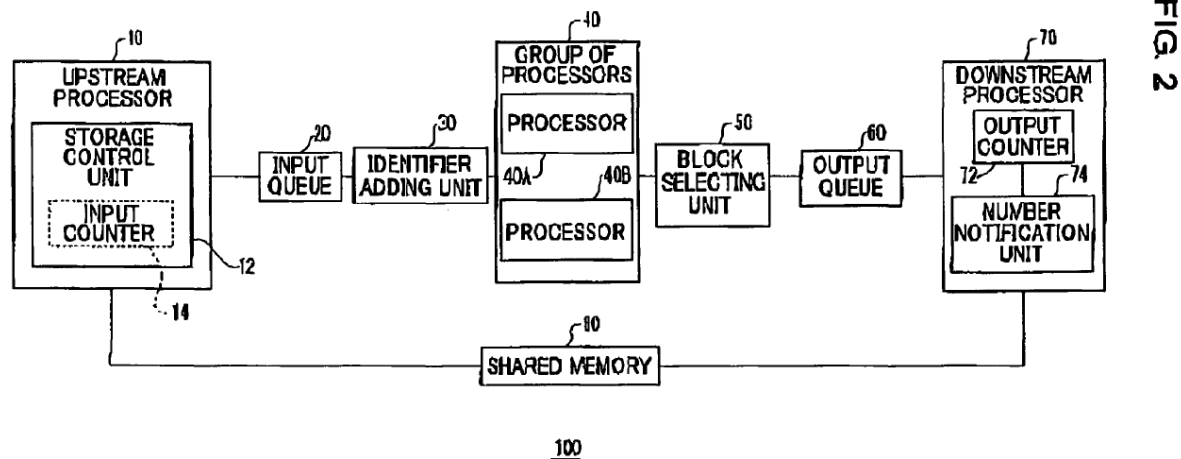
Ohkawa is prior art under §§102(a)(1)-(2). Ex-1003, ¶¶347-48. Ohkawa is analogous art in the same field (*supra* §III) because it teaches a multiprocessor architecture with coordinated processing and queue structures. Ex-1010, 1:30-40; Ex-1003, ¶350.

2. Claim 1

Beyond the additional teachings discussed below, Ground 4 relies on the same teachings as Ground 3. *Supra* §V.C.3[1pre]-[1a], V.C.3[1d]-[1i]; Ex-1003, ¶¶351-52, 369-74.

[1b] an output queue implemented in hardware;

Ground 4 includes additional teachings from Ohkawa for an output queue implemented in hardware. Ohkawa teaches “output queue” buffers for data passed between processors in a multiprocessor system:



“The output queue 60 stores the data processed by the group of processors 40 temporarily before this data is read by the downstream processor 70.” Ex-1010, 5:11-13; Ex-1003, ¶¶353-54.

Ohkawa’s output queue is implemented in hardware. As Ohkawa explains, the elements of Fig. 2 “can be constituted... in terms of **hardware**.” Ex-1010,

4:52-61 (“[T]hese functional blocks may be achieved by various forms, including hardware alone, software alone...”). The output queue is implemented in physical circuits, without software intervention, because Ohkawa teaches a hardware implementation. *Id.*; Ex-1003, ¶355.

Motivation to Combine. Ground 3’s motivation was explained above. *Supra* §V.C.3[1e] (motivation subsection). Ex-1003, ¶356.

Given Khawand’s delegation of service applications from one processor to another, it was obvious to use an output queue to buffer and return data from a delegated service application. *Supra* §V.C.3[1b]. Khawand’s architecture suggests the use of output queues to pass data, and Ohkawa teaches output queues for this express purpose. Ex-1010, 4:15-17, 4:31-33, 5:11-13, Fig. 2. Therefore, a POSITA would have been motivated to use Ohkawa’s output queues to buffer return data from delegated processes (e.g., for data sent from Processor 2 to Processor 1 in the examples described above for Ground 3). *Supra* §V.C.3[1c]; Ex-1003, ¶¶357, 359.

Moreover, Khawand teaches a multiprocessor architecture, and Ohkawa teaches output queues for multiprocessor systems. Ex-1008, Fig. 3; Ex-1010, Abstract, 1:6-40, 5:11-13. Adding Ohkawa to the combination would have been straightforward as it simply combines known prior art elements according to known methods to yield a predictable result—the use of output queues in a

multiprocessor system. The combination would have been the application of a known technique (output queues) to improve a similar device in the same way. Ex-1003, ¶358.

The references provide additional motivation because they teach using queues for inter-processor communication. Ex-1008, 3:65-4:22 (delegating service applications from one processor to another); Ex-1009, 8:33-38 (discussing advantage of using a FIFO to facilitate asynchronous operations between the processor and I/O devices); Ex-1003, ¶360.

A POSITA would have been motivated to use Ohkawa's hardware output queues because they were widely-used building blocks in computer architecture, known to provide efficient buffering between processors. A mix of fast hardware queues would have enabled the processors to exploit temporal locality by allocating tasks to hardware queues, with flexibility to have additional overhead queues in memory, like cache of sorts. Both Khawand and Ohkawa suggest this type of hybrid approach. Ex-1008, 6:14-17; Ex-1010, 4:52-61; Ex-1003, ¶361.

A POSITA would have had a reasonable expectation of success because the combination uses conventional and well-known techniques (output queues) to yield a predictable result (buffers between processors). A POSITA would have expected success in combining given the similarity of Khawand's and Ohkawa's multiprocessor architecture—particularly since using output queues had been well-

known for many decades. Applying the teachings of Ohkawa would not have required specialized skills beyond what a POSITA would have had at the time of the alleged invention. Ex-1003, ¶¶362-363.

The combination would not have changed the principle of operation for any reference. Khawand's multiprocessor architecture already suggests the need for output queues. *Supra* §V.C.3[1b]. Ohkawa's output queues would be used to buffer data between processors, as described by Ohkawa. And Rosenthal's teachings would still be used in the manner taught by Rosenthal, as explained for Ground 3. Ex-1003, ¶364.

[1c] a first processing element having access to said input queue and said output queue;

Ground 4 includes additional teachings from Ohkawa for an output queue. *Supra* §V.D.2[1b]. Ground 3 teaches a first processing element (e.g., Processor 1) having access to the input queue and the output queue. *Supra* §V.C.3[1c]. The motivation to combine was explained above. *Supra* §V.D.2[1b] (motivation subsection); Ex-1003, ¶¶366-67.

Ohkawa teaches the first processing element having access to both the input and output queues. For example, Ohkawa teaches a general multiprocessor architecture, where each processing unit has access to an input queue (to the processor) and output queue (from the processor):

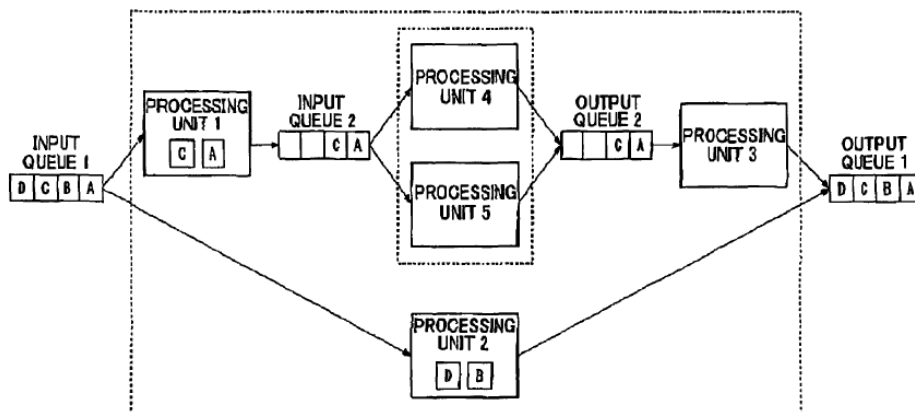


FIG 14

Ohkawa's queue teachings "may also be applied to a system where a plurality of processors perform coordinate processing using a hierarchical configuration."

Ex-1010, 9:64-66. It was obvious to use Ohkawa's queue structure, with each of Khawand's processing elements accessing an input and output queue because, like Ohkawa, Khawand includes multiple interconnected processing engines that pass data among them. Ex-1008, Fig. 3; Ex-1003, ¶368.

[1j] wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue.

Ground 4 includes additional teachings from Ohkawa for the output queue.

Supra §§V.D.2[1b]-[1c]. The motivation to combine was explained above. *Supra* §V.D.2[1b] (motivation subsection); Ex-1003, ¶¶375-76.

Ohkawa’s “output queue 60 stores the data processed by the group of processors 40 temporarily before this data is read by the downstream processor 70.” Ex-1010, 5:11-13. Therefore, when combined with Khawand, it was obvious for Processor 2 to store processed data in the output queue, and for Processor 1 to receive that data, in response to its delegated service application, by reading from the output queue. *See id.*; Ex-1003, ¶¶377-78.

3. Claims 2, 8, 14

These claims were explained for Ground 3. *Supra* §V.C.4; *supra* §V.D.2[1b] (motivation subsection).

Ohkawa provides additional teachings that “data stored in the output queue is read in the order stored”—i.e., first-in-first-out (“FIFO”). Ex-1010, 4:41-43; Ex-1003, ¶¶379-80, 396, 415.

4. Claims 6, 12, 18

As explained for Ground 3, a third processing element (e.g., Khawand’s Processor 2) executes the delegated command and notifies the output queue to issue the response. *Supra* §V.C.5; *supra* §V.D.2[1b] (motivation subsection); Ex-1003, ¶¶381-82.

Ohkawa provides additional teachings for Processor 1 receiving a response from the output queue. Ohkawa teaches that “[t]he **output queue 60 stores the data** processed by the group of processors 40 temporarily **before this data is read**

by the downstream processor 70.” Ex-1010, 5:11-13. Therefore, when combined with Khawand, it was obvious for Processor 2 to use the output queue to store processed data from the executed command, and to notify the output queue to issue the response to Processor 1 after the service application is complete. *See* Ex-1010, 5:11-13; *supra* §V.C.5; Ex-1003, ¶¶383, 397, 416.

5. Claim 7

Ground 4 relies on the disclosures of Ground 3 with the additional Ohkawa teachings explained above. *Supra* §V.C.6[7pre]-[7k] (Ground 3), §V.D.2[1pre]-[1j] (Ohkawa’s additional teachings); Ex-1003, ¶¶384-395.

6. Claim 13

Ground 4 relies on the disclosures of Ground 3 with the additional Ohkawa teachings, explained above (*supra* §V.C.7[13pre]-[13k] (Ground 3), §V.D.2[1pre]-[1j] (Ohkawa’s additional teachings)) and below. Ex-1003, ¶¶398-401, 405-10.

[13d] at least one second processing element in communication with said input queue and said output queue;

Ground 4 includes additional teachings from Ohkawa regarding the output queue. *Supra* §§V.D.2[1b]-[1c]; Ex-1003, ¶¶375-78. The motivation to combine was explained above. *Supra* §V.D.2[1b] (motivation subsection); Ex-1003, ¶402.

Ohkawa teaches a general multiprocessor architecture, where each processing unit is in communication with an input queue (to the processor) and output queue (from the processor):

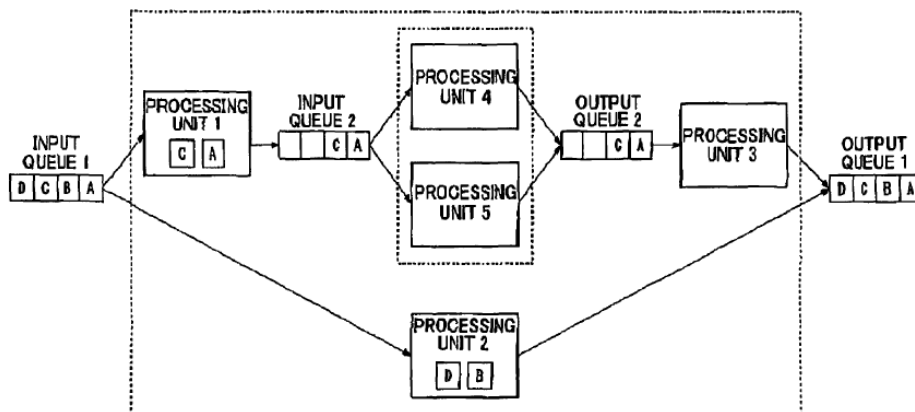


FIG 14

Ex-1010, 9:64-66. In Figure 14, processing unit 4 is in communication with Output Queue 2 and Input Queue 2. It was obvious to use Ohkawa's queue structure in Khawand's multiprocessor architecture where each of Khawand's processing elements accesses an input and output queue, like Ohkawa teaches, because Khawand similarly includes processing engines that pass data between them. Ex-1008, Fig. 3; Ex-1003, ¶¶403-04.

[13k] receives a response corresponding to said command from said output queue.

Ground 4 includes additional teachings from Ohkawa for the output queue. *Supra* §§V.D.2[1b]-[1c]; Ex-1003, ¶¶375-78. The motivation to combine was explained above. *Supra* §V.D.2[1b] (motivation subsection); Ex-1003, ¶¶411-12.

Ohkawa provides additional teachings for a processor to receive a response from the output queue. For example, Ohkawa's Processing Unit 3 receives data

from an output queue corresponding to a command; “[t]he output queue 60 stores the data processed by the group of processors 40 temporarily before this data is read by the downstream processor 70.” Ex-1010, 5:11-13. Therefore, when combined with Khawand, it was obvious for Khawand’s Processor 2 to store processed data in the output queue, and for Khawand’s Inter-Processor Manager to receive that data, in response to its delegated service application, by reading from the output queue (*see* Ex-1010, 5:11-13), particularly in view of the central role played by the Inter-Processor Manager in coordinating Khawand’s service application delegation process. *See, e.g.*, Ex-1008, 4:10-14; *supra* §V.C.6[7g], [7j]; Ex-1003, ¶¶412-14.

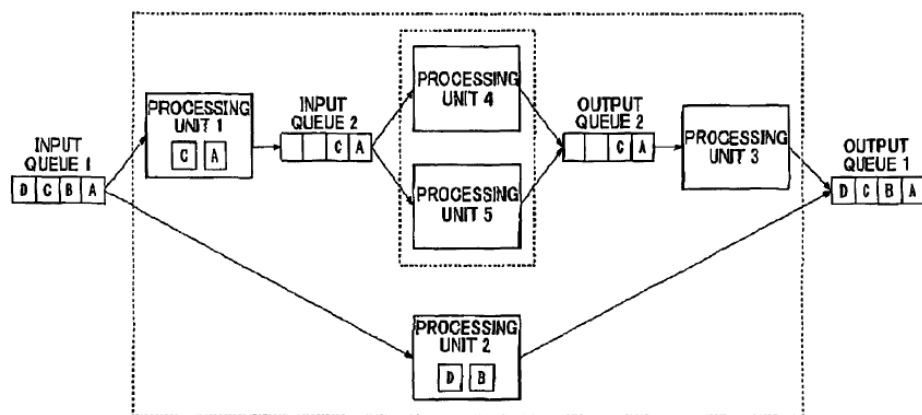


FIG. 14

E. Ground 5

Ground 4 teaches the independent claims. Matsunami teaches SAN and NAS. *Supra* §V.B.2. Ground 5 adds Matsunami’s teachings for dependent

SAN/NAS claims. The motivation to combine is explained below. *Infra* §V.E.1 (motivation subsection); Ex-1003, ¶417.

1. Claims 3-4, 9-10, 15-16

Ground 4 teaches claims 1/7/13. *Supra* §§V.D.2, V.D.5-6. Ground 5 further applies Matsunami’s teachings of a SAN/NAS controller. Ex-1007, Abstract (storage system with “NAS and SAN functions”), 1:18-23 (SAN), 1:30-34 (NAS), 2:1-4 (SAN/NAS complementary), 2:13-27 (combined SAN/NAS); *supra* §V.B.2 (explaining SAN/NAS teachings); Ex-1003, ¶¶417-18.

Matsunami suggests that the processors on the SAN/NAS channel interface controllers are multiprocessor systems. Ex-1007, 5:35-38 (“The processor 11001 may be a single processor or a **set of a plurality of processors**. For example, the processor 11001 may be configured as a **symmetric multiprocessor for horizontal load balancing** of control processing.”), 6:10-12 (“Like the processor 11001 of the network channel adapter 1100, the processor 11101 may also be configured as a single processor or a **multiprocessor**.”).

FIG. 4

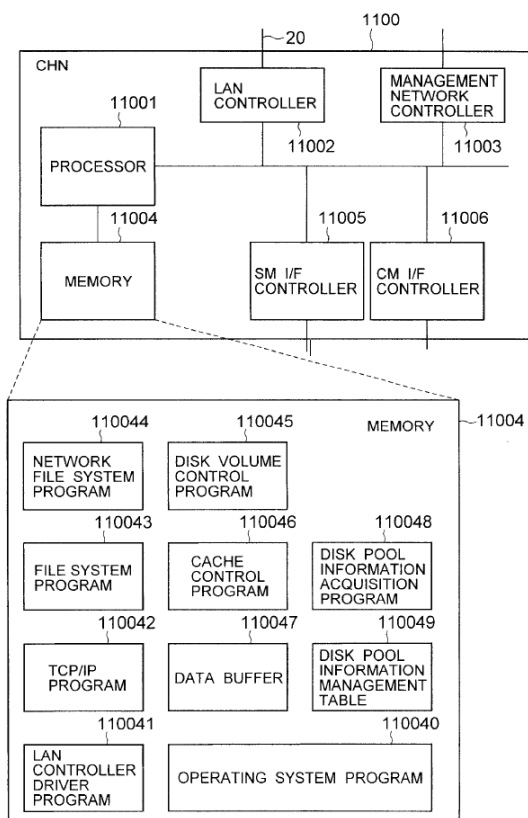
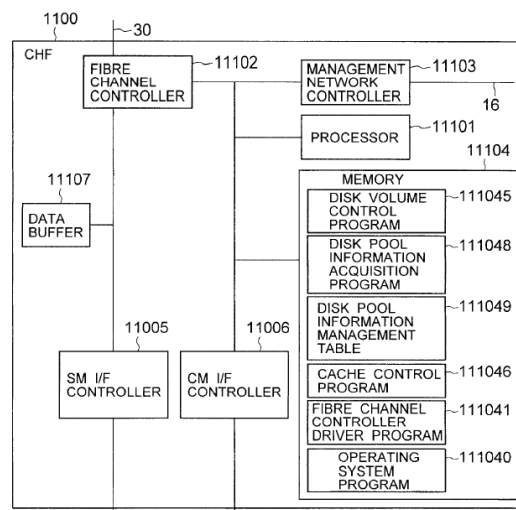


FIG. 5



Ex-1003, ¶419.

For Ground 5, the combination is a simple substitution of the processors in Matsunami's channel interface control cards with the multiprocessor system taught by Ground 4. Therefore, the multiprocessor system is part of a SAN/NAS controller. Ex-1003, ¶¶420, 427-28.

Motivation to Combine. The motivation to combine Khawand, Rosenthal, and Ohkawa was explained above. *Supra* §V.C.3[1e], §V.D.2[1b] (motivation subsections). Ground 4 teaches a multiprocessor system. *Supra* §V.D.2 (claim 1). Matsunami teaches a SAN/NAS controller and suggests using a multiprocessor

system for the processors on the channel interface cards. Ex-1007, 5:35-45, 6:10-12. A POSITA would have been motivated to combine Ground 4's multiprocessor system with Matsunami, replacing the channel interface control card processors with the multiprocessor system of Ground 4. Ex-1007, 5:35-45, 6:10-12. The combination would have been a simple combination of known prior art elements according to known methods to yield a predictable result (a multiprocessor SAN/NAS storage controller). Moreover, it would have been a simple substitution (explained above) that applied a known technique (Khawand's multiprocessor system) to improve a device (Matsunami's SAN/NAS controller) in the same way as explained for Ground 4. Ex-1003, ¶421.

Khawand's teachings are broadly applicable for general multiprocessor architectures. Khawand teachings are directed at "substantially improving the efficiency of a multiprocessor system." Ex-1008, 1:32-34. For example, Khawand's delegation of service applications is explained at a computer architecture level, and is applicable to multiprocessor architectures in general.¹²

¹² ARM-based multiprocessor architectures described in Khawand were becoming increasingly popular in servers and data centers. Ex-1008, 1:20-24; Ex-1031, 000001-4. SAN/NAS was commonly used for servers and in data centers. Ex-1007, 1:10-35; Ex-1003, ¶423.

E.g., Ex-1008, Fig. 3. A POSITA would have been motivated to apply those teachings to the processors in Matsunami's channel interface control cards to improve efficiency, including for shared service applications, based on Khawand. Ex-1008, 1:25-34; Ex-1003, ¶422.

The references provide motivation. Both references provide teachings related to I/O operations and storage. Khawand's multiprocessor system handles service applications and software applications divided into tasks including "input/output." Ex-1008, 3:13-17. Khawand uses solid-state flash memory. Ex-1008, 3:11-26, 2:63-67. Rosenthal teaches I/O operations, including access to hard disk drives. Ex-1009, Fig. 3; Ex-1009, 4:49-55; *see also* Ex-1009, 21:6-10. Matsunami describes "a storage system used in a computer system" and provides various teachings for I/O storage. Ex-1007, 1:6-7, 2:44-52, 2:13-19; Ex-1003, ¶424.

A POSITA would have had a reasonable expectation of success because the combination uses conventional and well-known techniques to yield a predictable result. Indeed, Matsunami already suggests that its interface control cards are based on multiprocessor systems. Ex-1007, 5:35-45, 6:10-12. Ground 4 teaches a multiprocessor system. Ex-1003, ¶425.

Combining the references would not have changed the principle of operation for any of the teachings of the references relied upon in the combination. The

application of a multiprocessor system would not have changed Matsunami's SAN/NAS teachings. Ground 4 still teaches a multiprocessor system. Ex-1003, ¶425.

SAN and NAS were well known by the time of the alleged invention. Multiprocessor systems were widely used in mobile, desktop, and server computers, and they were widely used in data centers where SAN and NAS were prevalent. Applying the teachings to Matsunami would not have required specialized skills beyond what a POSITA would have had at the time of the alleged invention. Ex-1003, ¶426.

VI. STANDING

Petitioners certify that Petitioners are not barred or estopped from requesting this inter partes review, and the '596 patent is eligible for it.

VII. DISCRETIONARY ANALYSIS

Discretionary denial is not warranted.

Fintiv. The factors favor institution. The parallel district court case is in its early stages. Magistrate Judge Lane recently issued a Report & Recommendation recommending that the case be transferred to the Northern District of California. Fact discovery is currently stayed.

Claim construction has not been briefed and no trial date is set. The court has not ruled on Petitioner's pending motion to dismiss and a hearing is yet to be

scheduled. This is the only IPR filed against the challenged claims to date.

General Plastic factors do not weigh against institution.

This Petition's merits are also compelling, and "if unrebutted in trial, would plainly lead to a conclusion that one or more claims are unpatentable by a preponderance of the evidence." June 2022, Director's Memo, 3-5.

Advanced Bionics. Denial under §325(d) is likewise unwarranted because Dongare, Gewirtz, Khawand, Ohkawa, and Matsunami were not cited nor considered during prosecution.

While Rosenthal was cited, it was not used in any office actions. The Office has never considered the allowability of the '596 patent in view of the combinations asserted here, including the combinations involving Rosenthal (Grounds 3-5).

BD Factors (a), (b), and (d) weigh against denial because the entirety of Grounds 1-2 (Dongare, Gewirtz, and Matsunami) were not considered during prosecution of the '596 patent. Dongare and Gewirtz are materially different from the cited art because they teach a system with multiple processing engines, queues, and a reservation register (*supra* §§V.A, V.B), which was never before the Examiner during prosecution of the '596 patent. For Grounds 3-5, Rosenthal was never discussed and the specific combinations with Khawand, Ohkawa, or

Matsunami were never considered. Grounds 3-5 are not the same nor substantially the same as art or arguments that were previously presented.

Factor (d) also weighs against denial because the arguments made during prosecution do not overlap with Petitioners' arguments. Grounds 1-2 were not previously considered. There were no rejections by the Examiner, and none of the references, including Rosenthal were ever used either alone or in any combination for a rejection.

BD Factors (c), (e), and (f) weigh strongly against denial. The allowance of the '596 patent was in error because each of the Grounds teach a system that use queues for task management along with a reservation register (*supra* §§V.A, V.C); but none of the references relied upon in this Petition were the basis of an Office Action rejection. Thus, the Examiner erred by overlooking the material teachings of these references. *See Snap, Inc. v. Playvuu, Inc.*, IPR2023-00848, Paper 12, 12-16 (PTAB December 6, 2023) (finding error when Examiner "overlooked, or did not appreciate, the materiality of [prior-art] references"); *Samsung Electronics Co., Ltd. v. Evolved Wireless LLC*, IPR2021-00943, Paper 9 at 10-12 (PTAB Dec. 1, 2021) (no denial when file history provides little insight into the Examiner's evaluation of art). Each of the prior art references are directly relevant, and therefore not cumulative of the art (notwithstanding Rosenthal's citation) cited during prosecution.

Factor (f) weighs against denial because the Petition presents testimony from Dr. Russ explaining Dongare, Gewirtz, Khawand, Rosenthal, Ohkawa, and Matsunami, which was not before the Examiner.

VIII. MANDATORY NOTICES AND FEES

1. **Real Party In Interest:** The real parties-in-interest here are Advanced Micro Devices, Inc. and Pensando Systems, Inc. No other parties directed, controlled, or funded this Inter Partes Review proceeding (IPR).

2. **Related Matters:**

- *Concurrent Ventures, LLC and XtreamEdge, Inc.. v. Advanced Micro Devices, Inc. and Pensando Systems, Inc., Case No. 1:24-00335-ADA (W.D. Tex.)*

3. **Lead and Backup Counsel:** Pursuant to 37 C.F.R. §42.8(b)(3) and 42.10(a), Petitioner designates Harper Batts (Reg. No. 56,160) as lead counsel and Chris Ponder (Reg. No. 77,167), Jeffrey Liang (Reg. No. 69,043), and James Young Hurt (Reg. No. 79,121) as back-up counsel, each of Sheppard, Mullin, Richter & Hampton LLP.

4. **Service Information:** Petitioner consents to service by electronic mail addressed to:

HBatts@sheppardmullin.com;

CPonder@sheppardmullin.com;

JLiang@sheppardmullin.com;

JHurt@sheppardmullin.com and

legaltm-amd-concurrent@sheppardmullin.com

Petitioner's counsel may also be served by mail or hand delivery at Sheppard, Mullin, Richter & Hampton LLP, 1540 El Camino Real Suite 120, Menlo Park, CA 94025. Petitioner's counsel may be reached by telephone at (650) 815-2673 and by facsimile at (650) 815-4668.

5. **Fees**: The Office is authorized to charge fees for this Petition to Deposit Account 50-4561, Ref. 71EY-388593.

Respectfully submitted,

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP

/Harper Batts/
Harper Batts (Reg. No. 56,160)
Attorneys for Petitioner

CLAIM APPENDIX

[1pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:

[1a] an input queue implemented in hardware;

[1b] an output queue implemented in hardware;

[1c] a first processing element having access to said input queue and said output queue;

[1d] at least one second processing element in communication with said first processing element;

[1e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;

[1f] computer storage storing instructions, which when executed by said at least one second processing element:

[1g] accesses said reservation register and reads said stored value;

[1h] determines when said read value indicates available space in said input queue for said first processing element to issue a command;

[1i] notifies said first processing element to issue said command to said input queue; and

[1j] wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue.

2. The system of claim 1, wherein said input queue is a first-in-first-out (FIFO) buffer.

3. The system of claim 1, wherein said system is part of a storage area network (SAN) controller.

4. The system of claim 1, wherein said system is part of a network attached storage (NAS) controller.

5. The system of claim 1, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.

6. The system of claim 1, wherein at least one third processing element executes said command and notifies said output queue to issue said response.

[7pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:

[7a] an input queue implemented in hardware;

[7b] an output queue implemented in hardware;

[7c] a first processing element having access to said input queue and said output queue;

[7d] at least one second processing element in communication with said input queue;

[7e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;

[7f] computer storage storing instructions, which when executed by said at least one second processing element:

[7g] receives a command from said first processing element;

[7h] accesses said reservation register and reads said stored value;

[7i] determines when said read value indicates available space in said input queue to issue said command;

[7j] issues said command to said input queue on behalf of said first processing element; and

[7k] wherein said first processing element receives a response corresponding to said command from said output queue.

8. The system of claim 7, wherein said input queue is a first-in-first-out (FIFO) buffer.

9. The system of claim 7, wherein said system is part of a storage area network (SAN) controller.

10. The system of claim 7, wherein said system is part of a network attached storage (NAS) controller.

11. The system of claim 7, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.

12. The system of claim 7, wherein at least one third processing element executes said command and notifies said output queue to issue said response.

[13pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:

[13a] an input queue implemented in hardware;

[13b] an output queue implemented in hardware;

[13c] a first processing element having access to said input queue and said output queue;

[13d] at least one second processing element in communication with said input queue and said output queue;

[13e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;

[13f] computer storage storing instructions, which when executed by said at least one second processing element:

[13g] receives a command from said first processing element;

[13h] accesses said reservation register and reads said stored value;

[13i] determines when said read value indicates available space in said input queue to issue said command;

[13j] issues said command to said input queue on behalf of said first processing element; and

[13k] receives a response corresponding to said command from said output queue.

14. The system of claim 13, wherein said input queue is a first-in-first-out (FIFO) buffer.

15. The system of claim 13, wherein said system is either part of a storage area network (SAN) controller.

16. The system of claim 13, wherein said system is part of a network attached storage (NAS) controller.

17. The system of claim 13, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.

18. The system of claim 13, wherein at least one third processing element executes said command and notifies said output queue to issue said response.

CERTIFICATE OF SERVICE

The undersigned hereby certifies that on March 7, 2025, true and correct copies of the foregoing Petition for *Inter Partes* Review of U.S. Patent No. 8,924,596 and Exhibits 1001-1031 were served in entirety on the Patent Owner at the following addresses of record as listed on Patent Center, via Priority Mail Express®:

Michael Gencarella
12875 Lucero Lane
Los Altos Hills, CA 94022
Official Correspondence Address

Concurrent Ventures LLC
c/o Beeson, Jesse D.
240 Woodscape Court
Johns Creek, GA, 30022-3244
Patent Owner Correspondence Address

XtreamEdge, Inc.
c/o The Corporation Trust Company
Corporation Trust Center
1209 Orange Street
Wilmington, DE 19801
Patent Owner Correspondence Address

The undersigned certifies that true and correct copies of the Petition for *Inter Partes* Review of U.S. Patent No. 8,924,596 and Exhibits 1001-1031 were also sent via electronic mail to the attorneys of record for Plaintiff in the concurrent litigation matter(s):

Michael Matulewicz-Crowley
mmatulewicz-
crowley@reichmanjorgensen.com
Christine E. Lehman
clehman@reichmanjorgensen.com;
myohannes@reichmanjorgensen.com

Ariane S. Mann
amann@reichmanjorgensen.com
Laura Carwile
lcarwile@reichmanjorgensen.com
Scott Cole
scole@reichmanjorgensen.com

Date: March 7, 2025

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP
/Harper Batts/
Harper Batts (Reg. No. 56,160)
Attorneys for Petitioner

1540 El Camino Real Suite 120
Menlo Park, CA 94025
Tel: (650) 815-2673

**CERTIFICATE OF COMPLIANCE WITH
TYPE-VOLUME LIMITATION, TYPEFACE REQUIREMENTS,
AND TYPE STYLE REQUIREMENTS**

1. This Petition complies with the type-volume limitation of 14,000 words, comprising 13,549 words, as counted using the Microsoft Word software that was used to prepare this paper, excluding the parts exempted by 37 C.F.R. § 42.24(a).

2. This Petition complies with the general format requirements of 37 C.F.R. § 42.6(a) and has been prepared using Microsoft® Word in 14-point Times New Roman.

Respectfully submitted,

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP

/Harper Batts/
Harper Batts (Reg. No. 56,160)
Attorney for Petitioner

Date: March 7, 2025

1540 El Camino Real Suite 120
Menlo Park, CA 94025
Tel: (650) 815-2673