

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.

Case No. IPR2025-00478

U.S. Patent 8,924,596

Declaration of Dr. Samuel Russ

TABLE OF CONTENTS

	Pages
I. OVERVIEW OF THE TECHNOLOGY.....	6
A. Multiprocessor Architectures	8
B. Memory Mapping.....	10
C. Multi Threading and Semaphores	12
II. THE '596 PATENT	18
A. Prosecution History	19
B. Priority Date	21
C. District Court Allegations	22
D. Challenged Claims	23
III. LEVEL OF ORDINARY SKILL IN THE ART	24
IV. CLAIM CONSTRUCTION	25
A. “input/output queue implemented in hardware”	26
V. INVALIDITY	27
A. Ground 1: Claims 1-2, 5-8, 11-14, and 17-18 are Rendered Obvious in View of Dongare and Gewirtz.....	28
1. Dongare (Ex-1005)	29
2. Gewirtz (Ex-1006)	32
3. Independent Claim 1	36
4. Dependent Claim 2	79
5. Dependent Claim 5	80
6. Dependent Claim 6	82
7. Independent Claim 7	83
8. Dependent Claim 8	91
9. Dependent Claim 11	91
10. Dependent Claim 12	92
11. Independent Claim 13	92
12. Dependent Claim 14	99

13.	Dependent Claim 17	99
14.	Dependent Claim 18	100
B.	Ground 2: Claims 3-4, 9-10, and 15-16 are Rendered Obvious in view of Dongare, Gewirtz, and Matsunami	100
1.	Matsunami (Ex-1007)	100
2.	Dependent Claims 3-4.....	102
3.	Dependent Claims 9-10.....	109
4.	Dependent Claims 15-16.....	110
C.	Ground 3: Claims 1-2, 6-8, 12-14, and 18 are Rendered Obvious by Khawand and Rosenthal	110
1.	Khawand (Ex-1008).....	111
2.	Rosenthal (Ex-1009)	113
3.	Independent Claim 1	115
4.	Dependent Claim 2	147
5.	Dependent Claim 6	148
6.	Independent Claim 7	150
7.	Dependent Claim 8	158
8.	Dependent Claim 12	158
9.	Independent Claim 13	159
10.	Dependent Claim 14	165
11.	Dependent Claim 18	165
D.	Ground 4: Claims 1-2, 6-8, 12-14, and 18 are Rendered Obvious by Khawand, Rosenthal, and Ohkawa.....	166
1.	Ohkawa (Ex-1010).....	166
2.	Independent Claim 1	168
3.	Dependent Claim 2	179
4.	Dependent Claim 6	179
5.	Independent Claim 7	180
6.	Dependent Claim 8	184
7.	Dependent Claim 12	184

8.	Independent Claim 13	185
9.	Dependent Claim 14	191
10.	Dependent Claim 18	191
E.	Ground 5: Claims 3-4, 9-10, and 15-16 are Rendered Obvious in view of Khawand, Rosenthal, Ohkawa, and Matsunami	191
1.	Dependent Claims 3-4.....	192
2.	Dependent Claims 9-10.....	197
3.	Dependent Claims 15-16.....	197
VI.	BACKGROUND AND QUALIFICATIONS	198
A.	Compensation.....	200
B.	Materials and Other Information Considered	200
VII.	UNDERSTANDING OF THE LAW	200
VIII.	RESERVATION OF RIGHTS	201
	APPENDIX 1: CURRICULUM VITAE OF DR. SAMUEL RUSS.....	202
	APPENDIX 2: MATERIALS CONSIDERED IN THE PREPARATION OF THIS DECLARATION	216
	APPENDIX 3: CHALLENGED CLAIMS.....	219
	APPENDIX 4: UNDERSTANDING OF THE LAW	225

I, Samuel Russ declare as follows:

1. My name is Samuel Russ. I am an Associate Professor of Electrical Engineering at the University of South Alabama. I have prepared this report as an expert witness retained by Advanced Micro Devices, Inc. and Pensando Systems, Inc. In this report I give my opinions as to whether certain claims of U.S. Patent No. 8,924,596 (“the ’596 patent”) are invalid. I provide technical bases for these opinions as appropriate.

2. This report contains statements of my opinions formed to date and the bases and reasons for those opinions. I may offer additional opinions based on further review of materials in this case, including opinions and/or testimony of other expert witnesses. I make this declaration based upon my own personal knowledge and, if called upon to testify, would testify competently to the matters contained herein.

3. The ’596 patent seeks to avoid software synchronization (e.g., with semaphores and mutexes) when accessing command or input queues. But the alleged invention—a “reservation register” that stores the amount of space in the input queue—was trivial and already known in the art, for example as taught by Gewirtz (Ground 1) and Rosenthal (Ground 3). Grounds 1 and 3 simply combine these references with conventional computer architecture arrangements that

describe the various processing elements and queues recited in the Challenged Claims.

I. OVERVIEW OF THE TECHNOLOGY

4. One of the basic data structures used in computers is called a queue. A queue is a line, like a queue of people in line to see a movie. In the computing arts, a queue is a linear data structure in which the first element put into the queue is the first element that comes out. In other words, like a line of people at a movie, the first one entering the line is also the first one to exit. Queues, in life and in computing, maintain the “first come, first served” policy or, stated differently, the first one in is the first one out. First-in-first-out is often abbreviated “FIFO.”

5. If a queue is implemented in ordinary memory, it has a “write pointer” (a pointer to the place where elements enter the queue) and a “read pointer” (a pointer to the place where elements exit the queue).

6. Queues are a powerful way to synchronize concurrently running processes or threads. For example, if one thread has access to the write pointer and another has access to the read pointer, there is not a thread safety issue because only one thread can update the write pointer (write data to the queue) and only one thread can update the read pointer (read data from the queue). This is why queues are often placed by design between concurrently running processes.

7. Because queues are so useful, chip manufacturers have created dedicated chips called FIFOs. FIFOs have a write port, a read port, and flags that indicate how full the FIFO is and whether there is space available, and how much space is available. Examples include the Texas Instruments SN74V273 FIFO. The datasheet is dated 2001 and we used it recently in our (successful) satellite design at the University of South Alabama.

8. Queue have finite size. 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.

9. A common mechanism for reporting queue space was with registers, which have been one of the basic building blocks of computers for many decades. For example, Gewirtz described a reservation register that tracked the “available entries for reservation” in a queue. Ex-1006, 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 also* 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).

A. Multiprocessor Architectures

10. By 2013, it was common for computers to have multiple processors inside them. For example, multiprocessor systems consisting of CPUs, dedicated hardware engines, DSPs, embedded processors, and other types of processors were widely known and used. Queues were often used to facilitate communications between these processors. *See* Ex-1005, Abstract (describing multi-engine processing system with queues); Ex-1006, 4:51-5:6 (describing software running on multiple processor cores using a reservation register to determine if it can write to a queue); Ex-1007, 6:10-12 (describing multiprocessor used in a network channel adapter); Ex-1008, Abstract (describing a multiprocessor system with queues that delegates tasks from processor to another); Ex-1009, 4:49-59 (describing multiple bus masters and bus slaves connected to an I/O bus); Ex-1010, Abstract (describing coordinated processing using queues for groups of processors).

11. Another example of a multiprocessor system with queue-based synchronization among three processors is found in a 2008 article by Ou. Ex-1016. Ou's Fig. 3 is illustrative of the architecture:

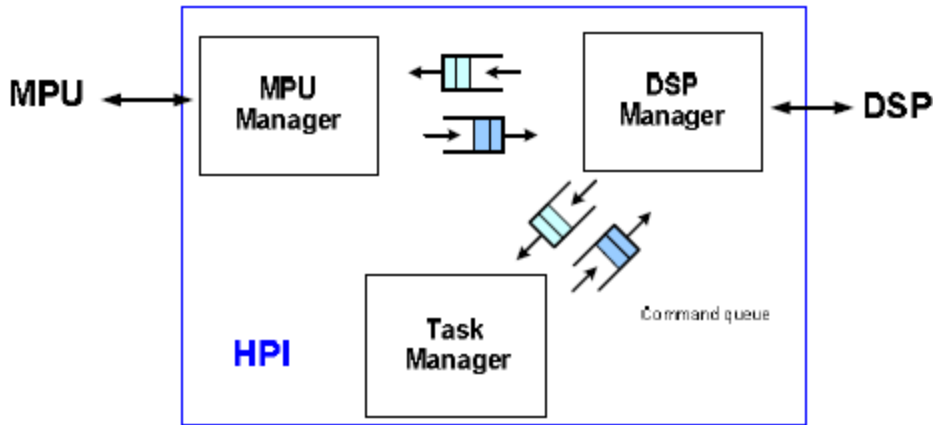


Fig. 3 HPI block diagram

Ex-1016, Fig. 3.

12. Ou's Figure 3 illustrates a task manager that coordinates the MPU (processor) and DSP (digital signal processor) by use of command queues. As described:

The proposed HPI consists of three interactive controllers (FSMs) including the MPU manager, the DSP manager, and the task manager as depicted in Fig. 3. Communication among those controllers is through command queues. The MPU manager takes care of commands such as data push/pull from the MPU or the DSP. The DSP manager will receive commands from the MPU or the DSP. Some commands such as "free memory page request/release" are handled by the DSP manager itself while others like "dequeue/enqueue" are bypassed to the task manager. The main function of the task manager is to monitor all task status and schedule any ready-to-run task to idle DSP thread for execution.

Ex-1016, 000001-2.

13. The Ou article also describes an embodiment, in which a third “task manager” processor manages enqueueing and dequeueing commands going to/from a DSP and interfacing with an MPU manager running on the main processor.

Ex-1016, 000001-2. In general, well before the filing of the '596 patent, it was well known in the art that there were several ways to arrange multiple processors to interact with and to manage queues, and Ou's embodiment is one example.

By 2013, a particular arrangement of tasks among processors would not be novel, as many possible arrangements would have been well known in the art. To restate, in Ou's embodiment the addition of a separate task manager co-processor enabled the MPU and DSP both to operate more efficiently. The MPU is freed from having to create and assign tasks (processes) to the DSP and the DSP is able to be kept busier because the dedicated task manager can assign another task as soon as one finishes.

B. Memory Mapping

14. One common way to allow a processor to access internal control registers is by mapping the registers to addresses in memory space, where accesses to that address will be directed to the register. Consider, for example, the Atmel 8-bit microcontroller. This is a common, inexpensive microcontroller. Consider this excerpt from its datasheet:

(0xC6)	UDR0	USART I/O Data Register								194
(0xC5)	UBRR0H	USART Baud Rate Register High								198
(0xC4)	UBRR0L	USART Baud Rate Register Low								198
(0xC3)	Reserved	–	–	–	–	–	–	–	–	
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 / UDORD0	UCSZ00 / UCPHA0	UCPOL0	196/207
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	195
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	194
(0xBF)	Reserved	–	–	–	–	–	–	–	–	
(0xBE)	Reserved	–	–	–	–	–	–	–	–	



Ex-1025, 000008.

15. This shows that the “UCSR0C” register is accessed via address location 0xC2 (hexadecimal C2). A read from address 0xC2 reads the values found inside that register. This is why all modern processors (that I am aware of) use this technique—a CPU can easily read and write control registers simply by reading and writing to specific addresses.

16. Examining the table in the datasheet, it is clear that the defined memory range goes from memory address 0x00 to memory address 0xFF. Ex-1025, 000008-11. In other words, it would be a simple matter for a hardware designer to add more registers and map them into the processor’s memory space at an address higher than 0xFF without disrupting the operation of any other registers. A processor’s memory map is determined by the address decoder that selects and activates the specific memory location or device that corresponds to the given address. By using different address ranges for various memory components and devices (like RAM, ROM, I/O ports, registers), the address decoder determines which device to access based on the address being used.

C. Multi Threading and Semaphores

17. Binary electronic computers are only approximately 75 years old.

The first electronic computers, based on vacuum tubes, came along during World War 2 and their better-known transistorized versions came out shortly after. Even in my childhood, computers were roughly the size of refrigerators.

18. Early computers contained a single central processing unit (CPU) that fetched and executed instructions. In that world, programs were relatively straightforward—one CPU ran one program. Even then problems arose.

Computers performed input/output operations and, as storage technology evolved, access to input/output soon became necessary to store a non-trivial amount of information.

19. In the emerging world where bulk data storage was much slower than the CPU, CPUs migrated from polling input/output (waiting for the I/O operation to complete before continuing) to interrupt driven to improve overall performance. In the interrupt-driven model, a CPU assigns an operation to an I/O unit and then resumes its main program. Later, when the I/O unit is finished, the CPU is notified via an interrupt. Importantly, an interrupt can occur at any time while the program is running. At this point, programming becomes significantly more complicated.

20. I teach interrupts in my sophomore-level undergraduate microprocessor class. The important feature is that the main program can be

interrupted at any moment. The only way that the main program can run reliably is if the interrupt software maintains the illusion, to the main program, that nothing is happening behind the program's back. For example, if an interrupt could overwrite the variable "x" then the main program would live in a world where "x" could spontaneously change with no warning. It rapidly becomes clear that something fundamental has shifted.

21. In the history of computing, this issue came to the surface rather quickly. For example, it was discussed in a seminal paper by Djiskstra in 1966 (as cited in a paper by Wirth in 1969). In other words, the problem of two pieces of software that can execute in any order was recognized as a problem almost 60 years ago.

22. In short, the main program and the interrupt service routine are separate threads (to be clear, the name "thread" did not come along until later, but the concept is clear). In the computing arts, a thread is a single schedulable piece of software. Each thread has its own set of instructions, access to its own set of data (variables), and its own program state. Importantly, one thread cannot alter the instructions, data, or state of another thread, or else the result will be chaotic and unpredictable execution.

23. This situation was explained in a 1969 paper by Niklaus Wirth.

Ex-1012. Wirth's paper explains the following:

Most modern data processing systems consist of several machines which can operate concurrently, and most programs make use of many of these concurrently operating units. This is particularly the case for input and output (I/O) activities. ...

The difficulties arising with multiprogramming compared with uniprogramming (programming a single sequential algorithm) are such that every effort is made to save the programmer from having to specify concurrent operations.

On the machine code level, however, these difficulties cannot be circumvented: the activities of every unit must be specified in detail. Unfortunately this level offers the least safety against programming mistakes, because there is no redundancy as provided by higher level languages, which can be used by compilers to check against errors. It is therefore particularly desirable that at the machine code level the programmer be able to work on the basis of clear and systematic concepts of concurrent activities--their initiation, termination, and synchronization.

Ex-1012, 000001.

24. In other words, programmers needed access to a simple set of concepts to enable concurrently running threads to be able to work together harmoniously on a single computer. The paper goes on to explain the concept of semaphores.

Synchronization among concurrent processes is performed by means of variables common to those processes. Dijkstra has thoroughly

investigated the problems arising from access of common variables (and the reader is encouraged to study his treatise at this point [3, 7]). He postulates two basic operations which constitute an interlocking mechanism, named P and V. These operations act on variables which assume integral values (0, 1, 2, •••). Dijkstra appropriately calls these variables semaphores, and it is subsequently postulated that P and V be the only operations applicable to variables designated as semaphores. (The observation of this postulate will be crucial for the implementation discussed below.) The operation V(s) (3) increments the value of the semaphore S by 1. The operation P(S) (4) can only be performed when the value of S is positive, which is decremented by 1. From this it follows that P may effectuate a delay of program execution until another process performs a V operation on the semaphore S. Thereby a synchronization of the two processes is obtained.

Ex-1012, 000002.

25. To recap, by the mid-1960s, it was clear that concurrently running pieces of software need a mechanism to synchronize their actions. For example, one purpose of a semaphore is to make sure that only one thread has access to a resource at a time.

26. Consider what happens when two threads have write access to the same area of memory without coordination between them. When one thread writes to a variable in memory and reads back the variable, normal programming practice demands that the variable have the same value. The analogy I use in my class is

this: imagine a world where you drive to work and when you return home, it has been replaced by an elephant sanctuary. Like threads, we expect the world to remain the same when we are not observing it.

27. The semaphores described by Dijkstra serve as a safeguard. A thread can only obtain write access if it successfully performs a P operation with a positive result.

28. A subtle side effect of semaphores is that a single semaphore has to be updated atomically. That is, the process of reading and updating (and specifically the internal steps of the P operation) must be executed without any interruptions. In computing circles, this is sometimes called atomic test and set.

29. Circling back to the P and V example in the Wirth paper, a common approach was to extend the P and V concept incrementally, e.g., to modify P and V such that the integer value of the semaphore represents the amount of empty space available for storage. If a thread requested “100” and the value of the integer was 110, the integer would be lowered to 10 and the 100 bytes would be made available to the thread. If a thread requested “200” and the value of the integer was 110, the thread would block until more space was available. In other words, semaphores are very useful tools for providing concurrent access to hardware resources.

30. It is worth noting that semaphores were implemented in software or hardware. In software, semaphores were maintained as long as the “P” process can occur without interrupts. In other words, the heart of the “P” process contains a critical section, a piece of code that runs with interrupts disabled. In hardware, many mechanisms existed to permit atomic (that is, uninterruptible or indivisible) access. For example, U.S. Patent No. 5,276,886 to Dror describes a hardware semaphore. Ex-1013. A 2006 article describes the creation of a set of operating-system-accessible hardware semaphores to permit more rapid and more efficient access to hardware co-processors. Ex-1014. As explained in this article, hardware implementations were more efficient, and included the ability to arbitrate as necessary and to perform atomic operations on shared resources, such as a read of a queue status register, or writing or reading to queue, and updating the queue status register accordingly based on read (increment) or write (decrement). *See generally* Ex-1014.

31. Modern CPUs, such as the commonly used ARM CPU, have instructions in their instruction set architecture specifically for accessing memory locations with mutual exclusion, a foundational step for creating a semaphore. For example, the LDREX (“load register exclusive”) and STREX (“store register exclusive”) instructions perform these steps. Ex-1015, 000004. In other words, ARM processors include instructions to maintain variables in memory that can be

“locked” and “unlocked” by running processes. This enables processes to maintain semaphores or to track other information without concerns of another process overwriting it without permission.

32. To recap, the notion of communicating processes is quite old, as is the notion of a simple set of constructs to manage their interaction. One common way to manage their interaction is with a semaphore, a construct that can exist in software or hardware.

33. To summarize, semaphores, hardware status registers, and/or memory locations accessed atomically by multiple processors, queues, multiple independent processors / processes, and concurrent queue access were all extremely well known in the art by 2013.

II. THE '596 PATENT

34. The '596 patent does not purport to invent FIFO buffers or queues; it admits these existed in the prior art. Ex-1001, 1:22-25. The '596 patent also admits that software techniques used for synchronization such as semaphores and mutexes existed in the prior art. Ex-1001, 1:25-27.

35. According to the '596 patent, “[i]t can become a challenge to ensure the module command queues (FIFOs) are not overfilled, thus synchronization is required amongst the multiplicity of processing elements.” Ex-1001, 1:22-32. “In the prior art, such synchronization is performed using software techniques, such as

semaphores and mutexes to a shared available space count variable in memory.”

Ex-1001, 1:25-27. “However, this takes additional processing time, making this process non-optimal.” Ex-1001, 1:27-29.

36. To address these supposed shortcomings of software semaphores, the ’596 patent purports to invent a “shared counter resource (e.g. a register) in the hardware representing how much free space there is in the command queue, accessible to one or more processing elements.” Ex-1001, 3:30-34.

37. However, as explained above, the prior art already included multiprocessor architectures with queues, along with registers indicating the amount of free space in those queues. *Supra* §I.

A. Prosecution History

38. I understand that the ’596 patent is the start of a family of patents, including patents in the United States and China, and an application filed in Europe.

39. On August 19, 2014, the ’596 patent received a first action allowance. Ex-1002, 000060-74. Prior to the allowance, an examiner-initiated interview occurred, in which the applicant agreed to the Examiner’s claim amendments as reflected in the claims of the issued patent. Ex-1002, 000075.

40. In the Examiner’s reasons for allowance, the Examiner stated the reason as:

the inclusion of the specific details of a reservation register that is accessed by the second processing element and the feature of the second processing element notifying the first processing element to issue a command to the input queue if the reservation register value indicates that there is available space in the input queue as are now included in independent claim 1

...

the inclusion of the specific details of a reservation register that is accessed by the second processing element and the feature of the second processing element receiving a command from the first processing element and the second processing element issuing said command to the input queue if the reservation register value indicates that there is available space in the input queue, as are now included in independent claims 7 and 13.

Ex-1002, 000072-73.

41. I understand that there was no substantive discussion of any prior art references, including the cited prior art references, during the prosecution of the '596 patent.

42. Before the '596 patent issued in the United States, a European counterpart application, EP14867963 (the "EU Counterpart") was filed. Ex-1004, 000080-83. During prosecution of the EU Counterpart, on June 7, 2016, an International Search Report and Written Opinion on Patentability were issued.

Ex-1004, 000084-88. On June 23, 2016, the application Entered into the European phase of prosecution. Ex-1004, 000089-93.

43. On May 19, 2017, a Supplementary European Search Report was issued that identified U.S. Patent No. 8,051,227 to Gewirtz and U.S. Patent Application Publication No. 2002/069245 to Kim. Ex-1004, 000111-113.

44. On June 13, 2017, an EU Search Opinion was issued rejecting all claims on the basis of the Gewirtz and Kim. Ex-1004, 000114-123. 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.

45. On April 23, 2018, the Applicant responded to the EU Search Opinion, by amending the claims to address the issues identified in the EU Search Opinion and argued against the rejections based on Gewirtz and Kim. Ex-1004, 000128-144.

46. On July 21, 2021, the EU Counterpart was considered withdrawn for non-payment of the 7-year renewal fee. Ex-1004, 000159.

B. Priority Date

47. The ’596 patent was filed on December 6, 2013, and did not claim priority to any earlier filed application. I have not conducted any analysis as to whether the ’596 patent is entitled to any claim of earlier priority. For the purposes

of this declaration, I have applied a December 6, 2013, priority date for my opinions.

C. District Court Allegations

48. In the District Court proceedings, I understand that Patent Owner has asserted that Petitioner infringes claims 1-4, 6-10, 12-16, and 18 (the “Asserted Claims”) of the ’596 patent. Ex-1017, 000001.

49. Patent Owner alleged that a memory location in DRAM can be the claimed “reservation register.” Ex-1018, 000016 (highlighting “DRAM-based qstate record”). To address this dispute over whether a register must be a register or whether it can also include DRAM, the parties proposed competing constructions for “register” in District Court. Petitioners proposed a construction that “register” refers to a “set of flip-flops that store a set of related bits.” Ex-1030, 000009. Patent Owner proposed “a memory location within a microprocessor used to store values and external memory addresses.” Ex-1029, 000002.

50. This claim construction dispute is not relevant to this IPR because the prior art explicitly teaches “registers.” *Infra* §§V.A.3[1e], V.C.3[1e]. Therefore, the parties’ proposed constructions—which are directed at whether a DRAM location could also be a register as Patent Owner contends—are simply irrelevant to the prior art grounds for this IPR. Regardless, I have explained below why both

constructions are nonetheless taught by the prior art. *See infra* §§V.A.3[1e], V.C.3[1e].

51. Regarding other claim terms, for the “input queue implemented in hardware” and “output queue implemented in hardware” limitations, Patent Owner broadly alleged that these queues are met by an assortment of queues, including queues implemented in memory, such as DRAM. Ex-1017, 000021-36; Ex-1018, 000012, 000016.

52. As another example, for the first and second processing element limitations, Patent Owner broadly alleged that the claimed processing elements do not require processors or processor cores, and can instead be met by a wide variety of functional blocks, including the P4 Processing Pipelines, table engines, match processing units, scheduler, doorbell engine, parser, deparser, DMA, splitter, ARM processors, and the host processor, among others. Ex-1017, 000036-55; Ex-1018, 000012-13.

D. Challenged Claims

53. I understand that Petitioner is challenging the validity of claims 1-18, of the '596 patent in the Petition for *Inter Partes* Review to which this declaration will be attached. Those claims are reproduced in Appendix 3. I have considered all claims 1-18 of the '596 patent, as well as portions of the '596 patent prosecution history in forming my opinions.

III. LEVEL OF ORDINARY SKILL IN THE ART

54. I have analyzed the '596 patent and determined that the field of the patent is computer architecture, including the use of queues and multiple processing elements. For example, the '596 patent characterizes its alleged invention as “a system for dividing and synchronizing a processing task across a plurality of processing elements.” Ex-1001, 5:3-6. More specifically, the '596 patent characterizes its invention as providing “a shared counter resource (e.g., a register) in the hardware representing how much free space there is in the command queue, accessible to one or more processing elements.” Ex-1001, 3:30-33.

55. In determining the characteristics of a hypothetical person of ordinary skill in the art (“POSITA”) of the '596 patent at the time of the claimed invention, I considered several things, including various prior art techniques relating to the field, the type of problems that such techniques gave rise to, and the rapidity with which innovations were made. I also considered the sophistication of the technologies involved, and the educational background and experience of those actively working in the field at the time. I also considered the level of education that would be necessary to understand the '596 patent. Finally, I placed myself back in the relevant period of time and considered the engineers and programmers that I have worked with and managed in the field of art of the '596 patent.

56. I came to the conclusion that a person of ordinary skill in the field of art of the '596 patent would have been a person with (1) a bachelor's degree in electrical engineering, computer engineering, or a comparable major, and (2) approximately two to three years of experience in the field. Additional experience can substitute for level of education and vice-versa. A person with less education but more relevant practical experience may also meet this standard.

57. I at least qualify under the definition of a POSITA because I had a BS degree in electrical engineering by 1986, with many years (more than three) of experience in the field (stated above) by 2013, as explained in my qualifications section. *Infra* §VI.

IV. CLAIM CONSTRUCTION

58. For purposes of this inter partes review, I have considered the claim language, specification, and portions of the prosecution history, to determine the meaning of the claim language as it would have been understood by a person of ordinary skill in the art at the time of the invention. The “plain and ordinary meaning” or *Phillips* standard has traditionally been applied in district court litigation, where a claim term is given its plain and ordinary meaning in view of the specification from the view point of a person of ordinary skill in the art.

59. I have applied the *Phillips* standard in my analysis. Unless otherwise stated, I have applied the plain and ordinary meaning to claim terms.

A. “input/output queue implemented in hardware”

60. I understand that Petitioners have proposed a construction for “input/output queue implemented in hardware” in district court. I analyzed the Petitioners’ proposed construction below and have applied it to my analysis in this Declaration, although the proposed construction is not necessary for this IPR because the prior art teaches this claim term regardless of whether this construction is adopted.

61. Based on my review of the claims and specification of the ’596 patent, it is my opinion that a POSITA would have understood an “input/output queue implemented in hardware” to mean an input/output queue implemented in physical circuits without software intervention.

62. The specification of the ’596 patent describes that “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. This is an example of how the ’596 specification explains what it means when it describes something as implemented in hardware. A POSITA would have viewed the discussion of “input/output queue implemented in hardware” in the ’596 patent as being consistent with this understanding of “input/output queue implemented in hardware” which was an input/output queue implemented in physical circuits without software intervention.

V. INVALIDITY

63. There are a number of patents and publications that constitute prior art to the '596 patent. I have reviewed and considered the prior art discussed in this section, along with the materials listed in Appendix 2.

64. Based on my review and analysis of the materials cited herein, my opinions regarding the understanding of a POSITA in the 2013 timeframe, and my training and experience, it is my opinion that a POSITA would have found the challenged claims of the '596 patent obvious in view of the invalidity grounds identified below. The reasons for my conclusions are explained more fully in the following sections.

Grounds	Challenged 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	Dongare, Gewirtz, and Matsunami
3	1-2, 6-8, 12-14, 18	§ 103	Khawand and Rosenthal
4	1-2, 6-8, 12-14, 18	§ 103	Khawand, Rosenthal, and Ohkawa
5	3-4, 9-10, 15-16	§ 103	Khawand, Rosenthal, Ohkawa, and Matsunami

65. I understand that Dongare, Gewirtz, Matsunami, Khawand, and Ohkawa were not cited nor considered during prosecution of the '596 patent.

Rosenthal was cited by the Examiner, but never relied upon by the Examiner for a rejection. Ex-1002, 000060-74. In addition, I reviewed the file history for the '596 patent, and am not aware of these references being discussed in any office action or in the prosecution history generally.

66. I understand that after the '596 patent issued in the United States, a European counterpart, EP 14867963, was prosecuted in the EP. Ex-1004, 000080-93. Additionally, I understand the EP Counterpart never issued and was considered withdrawn due to non-payment of renewal fees. *Supra* §II.A.

A. Ground 1: Claims 1-2, 5-8, 11-14, and 17-18 are Rendered Obvious in View of Dongare and Gewirtz

67. It is my opinion that a POSITA would have found the Challenged Claims obvious based on Ground 1, i.e., the teachings of Dongare in combination with Gewirtz. Dongare teaches an architecture where an embedded processor manages several input queues for a plurality of RAID engines. Ex-1005, 10:50-11:30; 14:30-55. Gewirtz provides details of known techniques for managing queues. Ex-1006, 2:46-3:7. The combination teaches all limitations of the Challenged Claims. The motivation to combine and reasonable expectation of success are explained below. *Infra* §V.A.3[1e] (Motivation Subsection explaining how and why Dongare and Gewirtz would have been combined).

1. Dongare (Ex-1005)

68. Dongare is U.S. Patent No. 8,559,439 to Dongare, et al., entitled “Method and Apparatus for Queue Ordering in a Multi-Engine Processing System.” Dongare was filed on November 3, 2011, claiming priority to provisional application No. 61/409,860 filed on November 3, 2010. Dongare issued on October 15, 2013. I understand that Dongare is prior art to the ’596 patent under at least AIA §102(a)(1)-(2).

69. I have reviewed the Dongare reference. I understand that the Dongare reference was not cited nor considered during the prosecution of the ’596 patent based primarily on the fact that Dongare is not cited on the face of the ’596 patent and was not discussed in the prosecution history of the ’596 patent.

70. Dongare teaches a host processor that writes to a plurality of input queues managed by an embedded processor for multiple RAID engines.

Specifically, Dongare’s Abstract discloses:

[c]ommands from single/multiple queues and multi-flows are processed by **multi-engines with different processing time and/or out of order**, which breaks sequential order of commands from same input queue and commands are distributed across multiple engines’ output buffer after processing. Processed commands are stored in dedicated command output buffer associated with each engine temporarily. The processed commands are re-ordered while writing out.

Ex-1005, Abstract; *see also* Ex-1005, 1:21-22 (“Multiple engines or processors are used to increase the throughput of a device or a network system.”).

71. Dongare’s Figure 4 illustrates the multiple processing engines architecture, input queues, output queues, and potential resequencing.

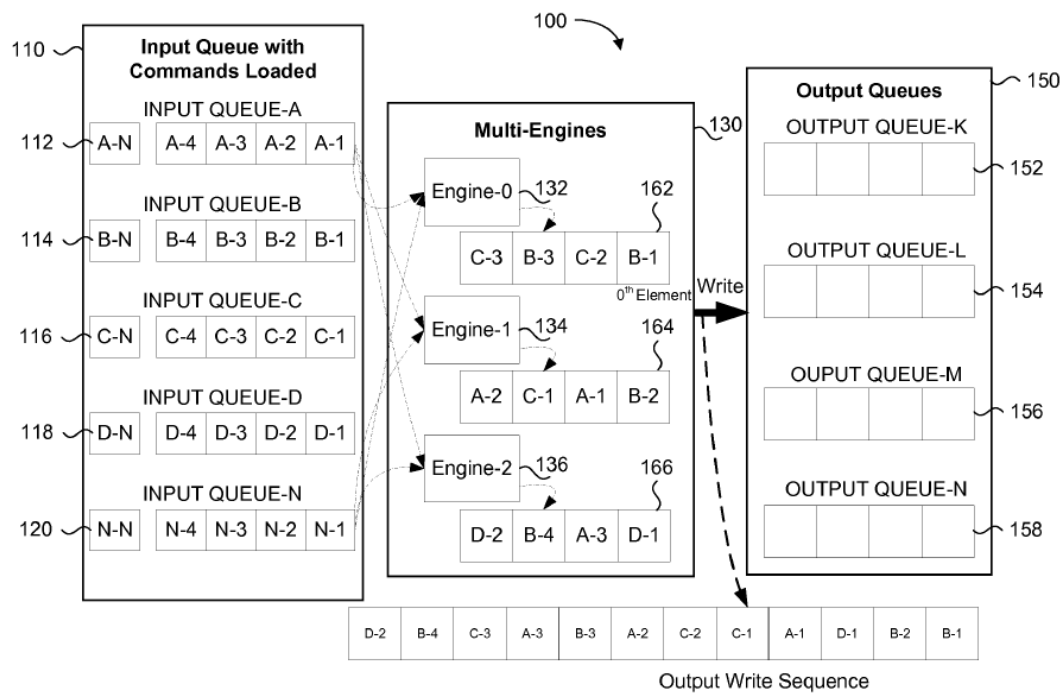


FIG. 4

Ex-1005, Fig. 4.

72. Dongare also teaches that its multiple processing engines architecture can be used as a controller for a RAID storage system.

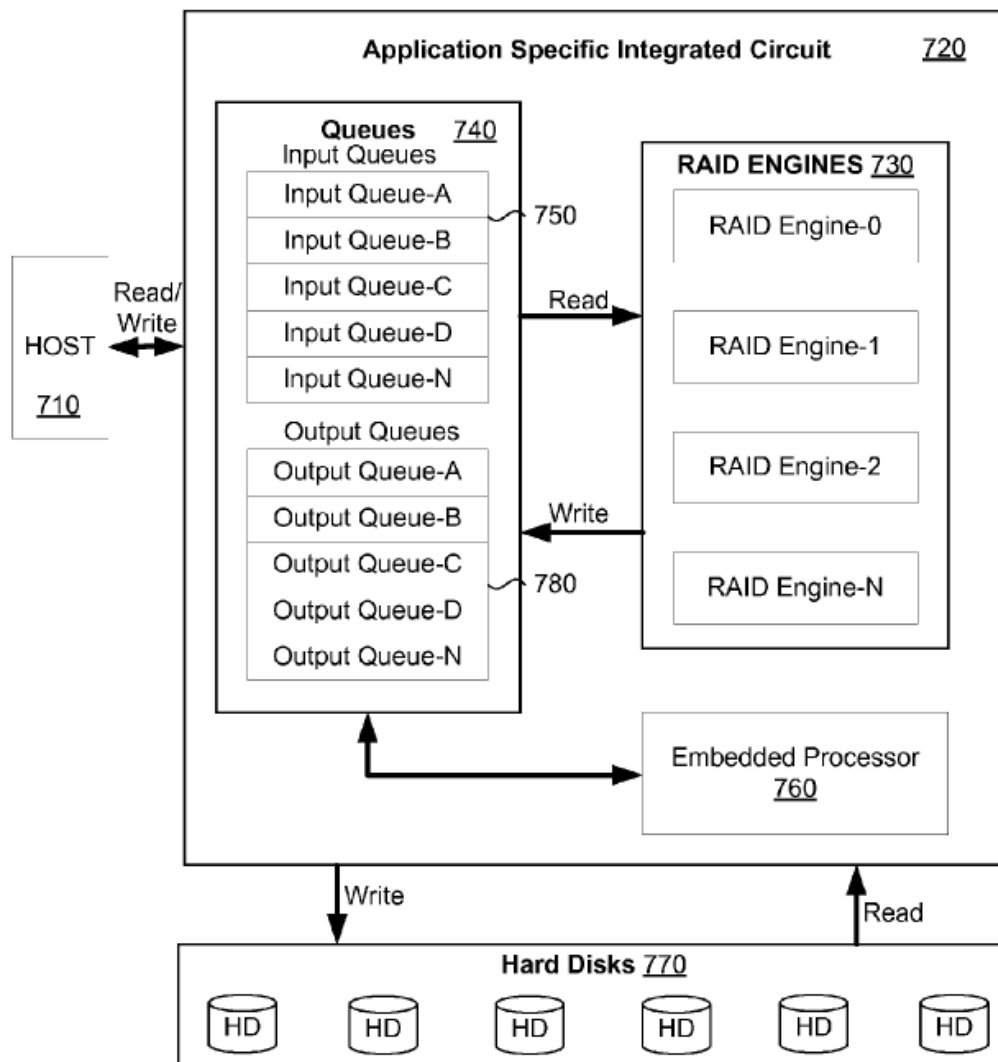


FIG. 7

Ex-1005, Fig. 7. Specifically, Dongare describes that “FIG. 7 provides an illustration of an application of the apparatus of FIG. 5 in a multi-engine processing system. In applications such as RAID controllers, the Host

computer 710 is connected to a RAID ASIC 720 via a PCIe protocol bus.”

Ex-1005, 14:30-34. Dongare describes a multiple processing engines architecture for use in storage systems that has input queues, output queues, multiple processors, and RAID processing engines. Commands to be processed by the RAID processing engines are sent from the host processor to the embedded processor via the input queues and results are returned to the host processor via the output queues with the embedded processor responsible for any necessary resequencing to maintain queue order to account for any out-of-order completion by the RAID engines. Ex-1005, 14:34-55.

73. Dongare is analogous art in the same field as the ’596 patent (computer architecture, including the use of queues and multiple processing elements, *supra* §III) because Dongare describes a computer architecture and teaches embodiments related to queues and queue ordering for increased efficiency. Ex-1005, 1:14-16; 1:36-60.

2. Gewirtz (Ex-1006)

74. Gewirtz is U.S. Patent No. 8,051,227 to Gewirtz, et al., entitled “Programmable Queue Structures for Multiprocessors.” Gewirtz was filed on May 10, 2010, and issued on November 1, 2011. I understand that Gewirtz is prior art to the ’596 patent under at least AIA §102(a)(1)-(2).

75. I have reviewed the Gewirtz reference. I understand that the Gewirtz reference was not cited nor considered during the prosecution of the '596 patent based primarily on the fact that Gewirtz is not cited on the face of the '596 patent and was not discussed in the prosecution history of the '596 patent.

76. Gewirtz teaches techniques for managing queues. For example, Gewirtz explains that:

In a multi-processor system, it is frequently desirable to build queues, or first-in-first-out (FIFO) buffers, in memory. A queue is a data structure with FIFO properties where agents push elements onto the queue (writes) and pop elements off of the queue (reads). This is normally accomplished by allocating storage for the queue elements, and separately allocating storage for the queue state which includes read and write pointers, and any other desired information.

Ex-1006, 1:12-19. Gewirtz explains one issue with queues in multiprocessor systems:

[s]ince there are usually multiple processors using a queue, this requires some forms of data locking of the queue states to ensure that the access to the queue states and the access to the queue element are atomic and consistent. Data locking, and particularly contention over the lock state memory locations, is often a performance-limiting operation in multiprocessor systems. In particular, if there are both multiple producers and multiple consumers of queue data, the performance implications can be severe.

Ex-1006, 1:27-35. Accordingly, Gewirtz teaches techniques for managing queues which were particularly useful when managing numerous queues.

77. Gewirtz teaches specific techniques to improve the performance of systems that interfaced with queues. For example, Gewirtz teaches memory-mapped software accessible registers that perform atomic operations performed by hardware related to queue management. Ex-1006, 1:39-60. Gewirtz explains that “[a]s a result, ... the software process represented by the first agent is not required to obtain an exclusive access control of the first queue from the Operating System.” Ex-1006, 1:60-64.

78. Gewirtz also discloses memory-mapped software accessible registers that contain various queue state information, including the number of free entries available in the queue.

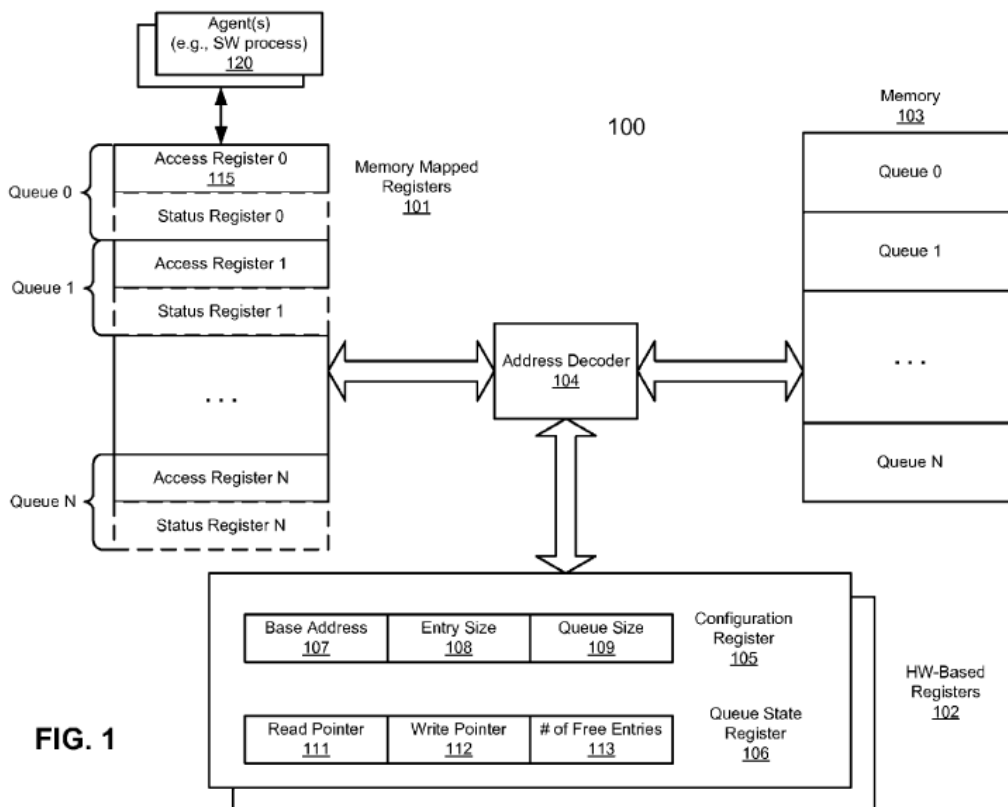


FIG. 1

Ex-1006, Fig. 1. Gewirtz explains that

[a]nother register (also referred to as a state register or status register) holds the dynamic state of the queue, including a read pointer and a write pointer, **queue occupancy, and/or number of free writeable entries** for flow control purposes.

Ex-1006, 2:59-62. Additionally, Gewirtz teaches software using the queue state register as a reservation register.

In one embodiment, state register 106 includes an optional built-in credit counter represented by the number of free entries 113 which can be accessed via a status register of memory mapped registers 101

corresponding to the queue. The credit counter originally contains the number of entries in the queue. When an agent wants to write an element into the queue, it first **reads the reservation register to see if an entry is available.**

Ex-1006, 4:51-58.

79. Gewirtz is analogous art in the same field as the '596 patent (computer architecture, including the use of queues and multiple processing elements, *supra* §III) because it teaches a computer architecture including queue management techniques and atomic hardware operations related to queue management. Ex-1006, 1:12-35; 2:46-3:7; 4:16-50.

3. Independent Claim 1

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

80. To the extent that the preamble is limiting, Ground 1 teaches limitation [1pre], as explained below.

81. 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.

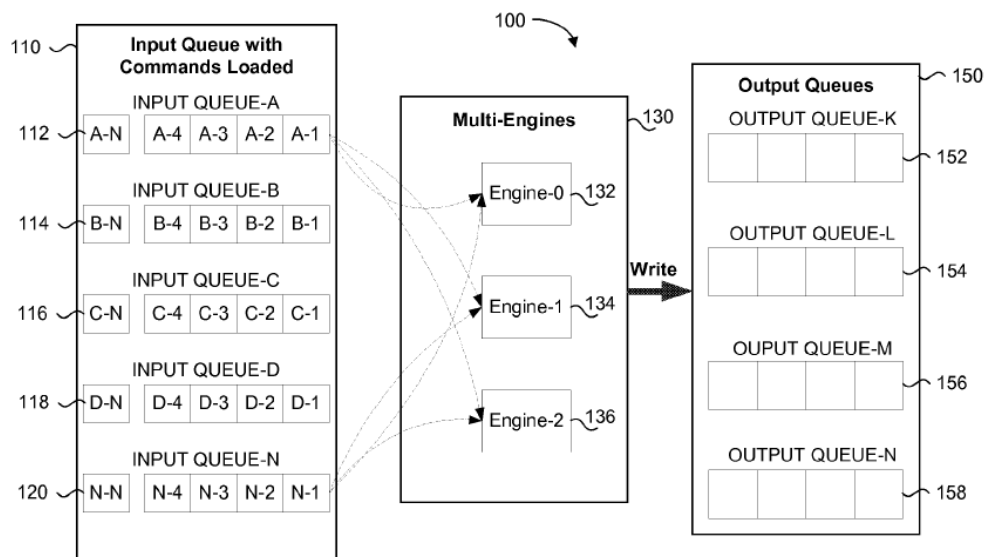


FIG. 1

Ex-1005, Fig. 1.

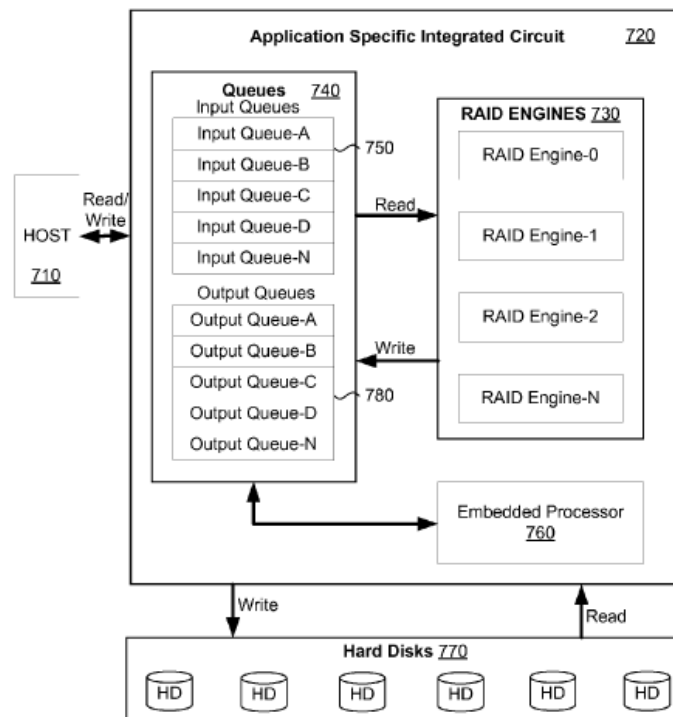


FIG. 7

Ex-1005, Fig. 7.

82. Dongare teaches dividing processing tasks to enable parallel execution (concomitantly) on multiple processing engines; and then reordering or synchronizing the completed portions as necessary:

In a multi-engine processor or a multi-threading system, **tasks from same program can be executed concomitantly** on different processors to achieve maximum throughput. However, the completed task has to be **provided in the order it was posted** to the multi-processor system. The method and apparatus of the present disclosure **provide for reordering completed tasks efficiently**.

Ex-1005, 4:49-55; *see also* Ex-1005, 1:22-25 (“Typically, segments of one or more input streams are parallel processed in multiple processors to increase the processing efficiency or throughout of the system.”). Dongare teaches that out-of-order execution includes “[a]dditional processing [that] is then required to restore the order of the segments of the output stream relative to the input stream resulting in a compromise of the efficiencies gained by multi-processing.” Ex-1005, 1:29-32; 4:49-55.

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

[1a] an input queue implemented in hardware;

84. Ground 1 teaches limitation [1a], as explained below.

85. Ground 1 teaches an input queue implemented in hardware, e.g., Dongare’s command input queue. Dongare teaches using its multiple processing engines architecture for applications like a RAID controller. Ex-1005, 14:30-34. Dongare’s Figure 7 discloses a RAID controller architecture with input queues, output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

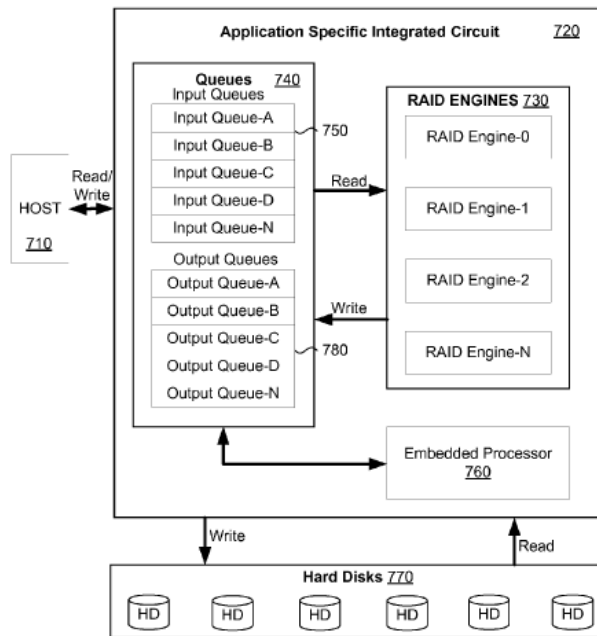


FIG. 7

Ex-1005, Fig. 7.

86. Dongare teaches at least one input queue for holding commands to be processed by the processing engines.

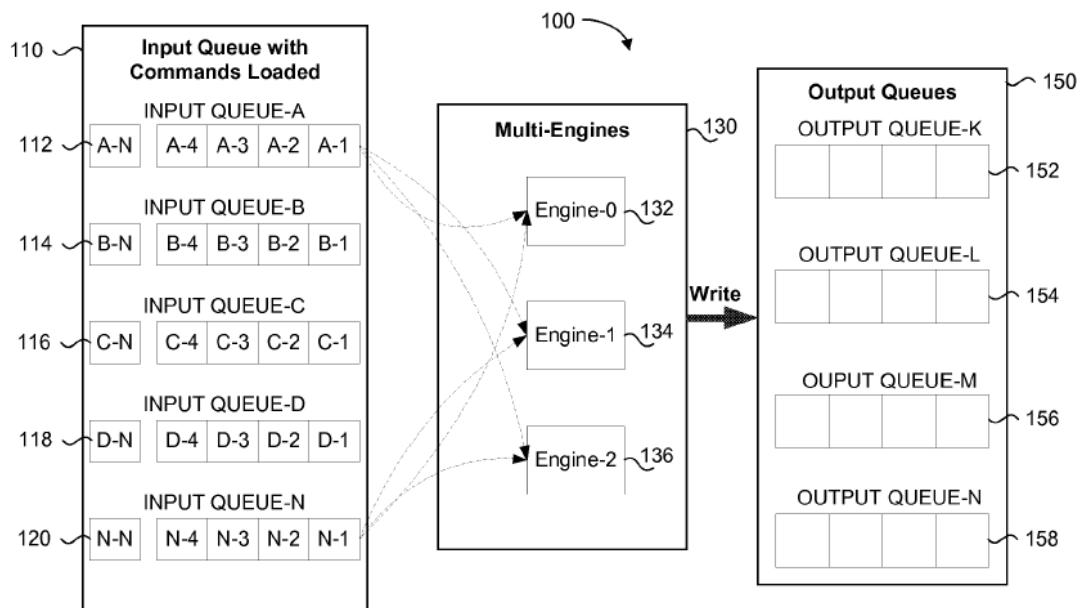


FIG. 1

Ex-1005, Fig. 1.

FIG. 1 illustrates a multi-engine processing system 100 capable of handling a plurality of input queues or input streams 110. The input queues are sequentially ordered commands or segments to be processed by the plurality of engines 130. The processed commands are written out to the plurality of output queues 150. In the example shown in FIG. 1, the plurality of input queues are received and stored in **input queue buffer-A 112**, **input queue buffer-B 114**, **input queue buffer-C 116**, **input queue buffer-D 118**, and **input queue buffer-N 120**. The plurality of processing engines 130 includes Engine-0 132, Engine-1 134, and Engine-2 136. The plurality of output queues 150 are written out and stored in output queue buffer-K 152, output queue buffer-L 154, output queue buffer-M 156, and output

queue buffer-N 158. Thus, command streams A, B, C, D through N can be received and stored in the respective input queue buffers and the processed commands can be written out and stored in output queue buffers.

Ex-1005, 5:3-20. Dongare teaches that “the command input queues may be buffered in one or more input queue buffer and one input queue buffer may hold commands from more than one command input queue.” Ex-1005, 5:33-36.

87. The input queues provide input to the RAID controller. With respect to the RAID controller application, Dongare discloses that “[t]he queues 740 in the controller are classified as **input queues 750 from the Host**, and output queues from the RAID engines 730. The Host 710 writes commands for the RAID controller into the input queues 750.” Ex-1005, 14:39-42.

88. Dongare teaches implementing the input queues as hardware. Ex-1005, 11:14-16 (“as a software routine, **hardware circuit**, firmware, or a combination thereof”) (emphasis added). 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.

89. INTENTIONALLY LEFT BLANK.

90. Gewirtz also teaches programmable queues for use as input or output queues.

Once a queue is set up, it can be used concurrently by any number of agents in the system, both producers (e.g., agents to write to the queue) and **consumers (e.g., agents to read from the queue)**.

Ex-1006, 2:55-58. A consumer agent reading from the programmable queue is using the queue for input, and thus the queue is an input queue.

91. Based on the foregoing, Ground 1 teaches limitation [1a].

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

92. Ground 1 teaches limitation [1b], as explained below.

93. Ground 1 teaches an output queue implemented in hardware, e.g.

Dongare's output queue for processed commands. Dongare teaches using its multiple processing engines architecture for applications like a RAID controller.

Ex-1005, 14:30-34. Dongare's Figure 7 discloses a RAID controller architecture with input queues, output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

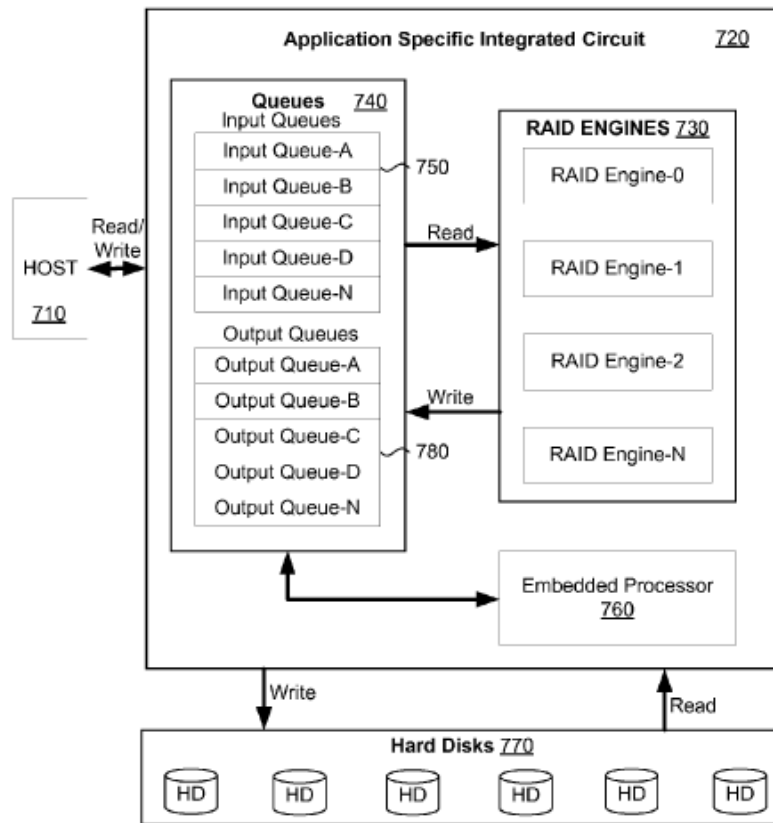


FIG. 7

Ex-1005, Fig. 7. Dongare teaches at least one output queue for returning results from the processing engines.

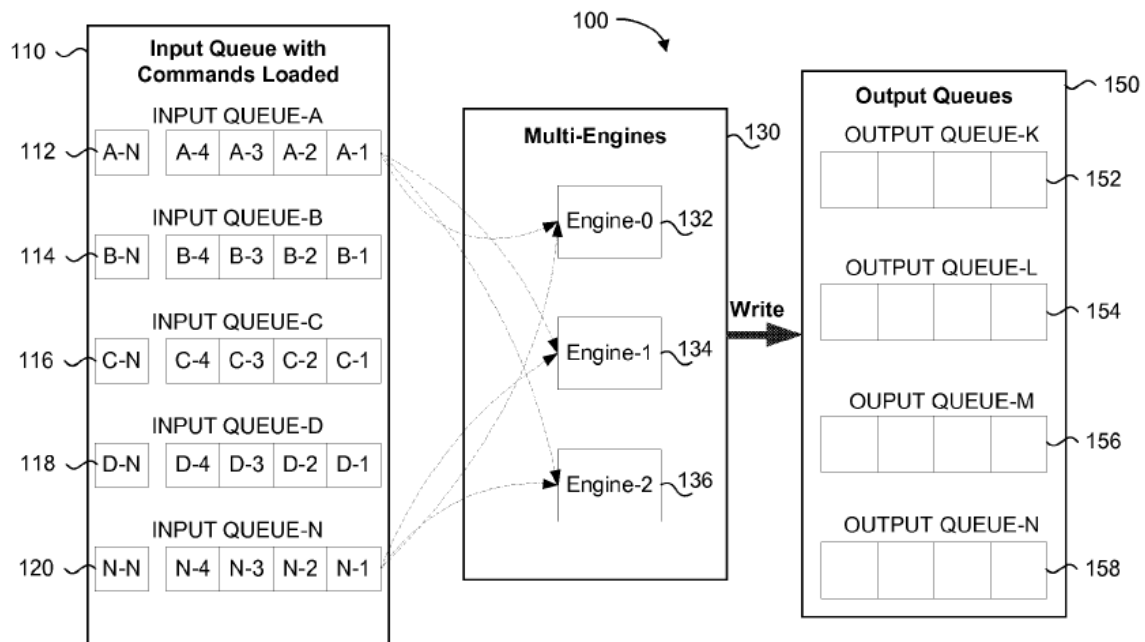


FIG. 1

Ex-1005, Fig. 1.

FIG. 1 illustrates a multi-engine processing system 100 capable of handling a plurality of input queues or input streams 110. The input queues are sequentially ordered commands or segments to be processed by the plurality of engines 130. **The processed commands are written out to the plurality of output queues 150.** In the example shown in FIG. 1, the plurality of input queues are received and stored in input queue buffer-A 112, input queue buffer-B 114, input queue buffer-C 116, input queue buffer-D 118, and input queue buffer-N 120. The plurality of processing engines 130 includes Engine-0 132, Engine-1 134, and Engine-2 136. **The plurality of output queues 150 are written out and stored in output queue buffer-K 152, output**

queue buffer-L 154, output queue buffer-M 156, and output queue buffer-N 158. Thus, command streams A, B, C, D through N can be received and stored in the respective input queue buffers and the processed commands can be written out and stored in output queue buffers.

Ex-1005, 5:3-20.

94. The output queues provide output from the RAID controller to the Host. With respect to the RAID controller application, Dongare discloses that “[t]he queues 740 in the controller are classified as input queues 750 from the Host, and **output queues** from the RAID engines 730. The Host 710 writes commands for the RAID controller into the input queues 750.” Ex-1005, 14:39-42. 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.

95. Dongare also teaches implementing the output queues in hardware Ex-1005, 11:14-16 (“as a software routine, **hardware circuit**, firmware, or a combination thereof.”) (emphasis added). 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.

96. INTENTIONALLY LEFT BLANK.

97. As explained above, *supra* §V.A.3[1a], Gewirtz teaches programmable queues for use in multiprocessor systems using memory to implement the storage for the queues. Gewirtz teaches that these programmable queues are output queues:

Once a queue is set up, it can be used concurrently by any number of agents in the system, both **producers (e.g., agents to write to the queue)** and consumers (e.g., agents to read from the queue).

Ex-1006, 2:55-58. A producer agent writing to the programmable queue is using the queue for output, and thus the queue is an output queue.

98. Based on the foregoing, Ground 1 teaches limitation [1b].

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

99. Ground 1 teaches limitation [1c], as explained below.

100. Ground 1 teaches a first processing element (e.g., Dongare's host processor) having access to said input queue and said output queue. For example, Dongare's Figure 7 discloses a RAID controller architecture with input queues, output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

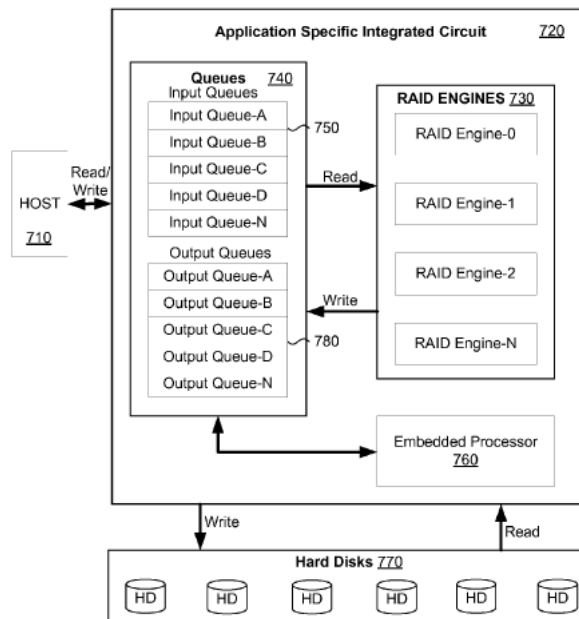


FIG. 7

Ex-1005, Fig. 7. Dongare's host processor is connected to the RAID controller via a PCIe protocol bus. Ex-1005, 14:31-34. Specifically, the "Host 710 and RAID controller 720 communicate to each other by using a set of queues 740 on the RAID controller 720"; the "Host 710 writes commands for the RAID controller into the input queues 750." Ex-1005, 14:37-42. In other words, Dongare's host processor has access to the input queue because it writes commands to the input queue. Ex-1005, 14:37-42.

101. Similarly, Dongare's host processor has access to the output queue. As Dongare explains, the host processor has the capability of 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.

102. Based on the foregoing, Ground 1 teaches limitation [1c].

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

103. Ground 1 teaches limitation [1d], as explained below.

104. 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). For example, Dongare's Figure 7 discloses a RAID controller architecture with input queues, output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

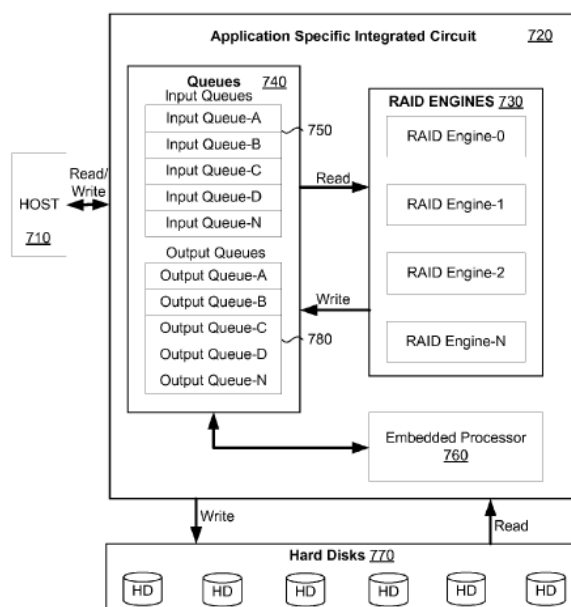


FIG. 7

Ex-1005, Fig. 7.

105. The embedded processor is in communication with the host processor. Dongare explains that the host 710 “is connected to a RAID ASIC 720 via a PCIe protocol bus.” Ex-1005, 14:31-34. Dongare’s RAID ASIC 720 comprises an embedded processor 760; communications between the embedded and host processors travel over the PCIe interface. Ex-1005, Fig. 7; 14:42-44. Dongare teaches that the embedded processor communicates with the host through the input and output queues: the “Host 710 writes commands ... into the input queues 750” which “are read by the embedded processor 760 and commands are created for RAID engines 730 to process data.” Ex-1005, 14:41-44. The embedded processor

also “signal[s] the Host 710 that the command is complete by writing into the outbound queue 780.” Ex-1005, 14:52-54. Dongare also discloses the embedded processor interrupting the host by writing to the outbound queue (Ex-1005, 14:54-55), which is an example of the second processing element communicating with the first processing element.

106. Based on the foregoing, Ground 1 teaches limitation [1d].

[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;

107. Ground 1 teaches limitation [1e], as explained below.

108. As explained for limitation [1g], *infra*, Dongare teaches that the embedded processor manages the input queues, for example through input queue handling and scheduling, but leaves implementation details to the knowledge of a POSITA. Ex-1005, 10:50-11:30. One aspect of queue-management includes tracking the amount of space in the queue. Gewirtz provides those teachings using a Queue State Register, which it also refers to as “the **reservation register**.” Ex-1006, 4:51-5:6.

109. Ground 1 teaches a reservation register implemented in hardware (e.g., Gewirtz’s Queue State Register) storing a value indicative of available space in said input queue (e.g., # of free entries). For example, Gewirtz teaches

programmable structures to manage queues. These structures include a programmable configuration register that holds information such as queue depth, base memory address, and other options. Ex-1006, 3:55-59. Gewirtz also teaches a “register (also referred to as a state register or status register)” that “holds the dynamic state of the queue, including a read pointer and a write pointer, queue occupancy, and/or number of free writeable entries for flow control purposes.” Ex-1006, 2:59-64; 4:5-15 (describing “reservation field... used to specify the number of entries available for writing into the queue.”). Gewirtz describes the contents of Queue State Register:

In one embodiment, state register 106 includes read pointer 111, write pointer 112, a number of occupied entries 113, and **available entries for reservation**. Read pointer 111 points to a head of a queue for reading a next queue element or data entry. Write pointer 112 points to a tail of the queue for writing new data into the queue. Number of occupied entries 113 includes data representing the number of entries currently stored in the queue. **The reservation field can be used to specify the number of entries available for writing into the queue.** All fields of state register 106 are maintained by hardware logic, for example, atomically.

Ex-1006, 4:5-15.

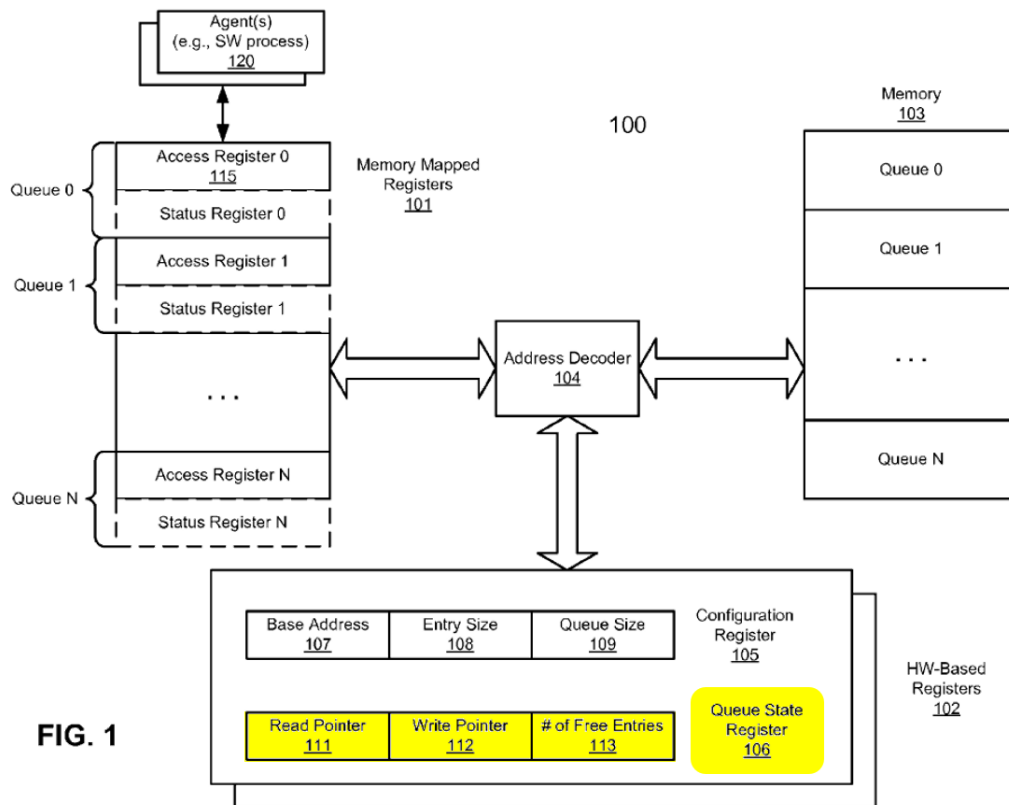


FIG. 1

Ex-1006, Fig. 1 (annotated).

110. Specifically, Gewirtz teaches a reservation register that includes the number of free entries for a queue, which software agents use:

In one embodiment, state register 106 includes an optional built-in credit counter represented by **the number of free entries 113** which can be accessed via a status register of memory mapped registers 101 corresponding to the queue. The credit counter originally contains the number of entries in the queue. When an agent wants to write an element into the queue, **it first reads the reservation register to see if an entry is available**. An agent may be one of multiple agents representing software processes, each being executed by one of

multiple processor cores of a network processor in a network element. The **queue hardware can return a pass/fail value to the agent in response to the read, depending on whether there is a reservation available**, which may be specified via a reserved field (not shown) of state register 106. When an agent receives a pass value, it can perform a write to one of the data access registers. If it receives a fail value, it must attempt to obtain a reservation again. If the agent decides not to write an element on to the queue, the agent may be required to write the reservation register to free the credit for subsequent use. The **available reservation count is maintained by hardware**: it is decremented when a successful reservation read request is processed, and incremented when either a reservation register write is seen, or when an element is popped off of the queue.

Ex-1006, 4:51-5:6. Therefore, Gewirtz teaches a reservation register.¹

111. A POSITA would have been motivated to use Gewirtz's Queue State Register teachings because they provided a simple mechanism for checking the amount of free space in a queue. This would have improved efficiency for multiprocessor systems by preventing scenarios where processors write to queues

¹ The '596 patent confirms that these teachings fall within the scope of a reservation register because its specification describes the same concept as Gewirtz when discussing the reservation register: "the hardware automatically decrements the available free space by a preconfigured amount (e.g., 1) ... In the event space is reserved but later it is determined that space is not to be used (e.g., decided to not issue the command after space has been reserved), there may be a mechanism available to a processing element to return reserved space back to the pool (via another register action) making it available space again." Ex-1001, 3:43-4:51.

with insufficient space and then attempt to rewrite the same data in a second attempt. Additionally, Gewirtz teaches atomic hardware operations that allow a processor to check free space without having to enter a spin lock and allows updates to the register's amount of free space value to occur in a correct manner. Ex-1006, 1:46-64; 6:18-25; 6:66-7:6; *see also* Ex-1006, 1:30-33. Moreover, a POSITA would have been motivated to use Gewirtz's Queue State Register because it provides a simple hardware mechanism for making reservations by automatically decrementing the number of available entries on a successful reservation read. Ex-1006, 4:55-5:6.

112. Gewirtz teaches implementing this register in hardware, as a "HW-Based Register[]" (Ex-1006, Fig. 1) that is "updated by hardware logic that manages the queues." Ex-1006, 2:59-64. Accordingly, Gewirtz teaches Petitioner's proposed construction of "register" in District Court (*supra* §II.C) because hardware registers have, for many decades, been implemented using a set of flip-flops. A POSITA would have understood this because it was a ubiquitous approach taught in undergraduate logic design courses for decades, as evident from various textbooks from the 1990s and before. *See* Ex-1019, 000013; Ex-1020, 000012; Ex-1021, 000012; Ex-1022, 000008; Ex-1023, 000025; Ex-1024, 000026. Therefore, a POSITA would have understood Gewirtz's teaching of a "HW-Based Register" to be a set of flip-flops storing a set of related bits, i.e. the fields related

to the queue status, including the read pointer, write pointer, and # of Free Entries.

Ex-1006, 4:5-15. Gewirtz teaches that “[a]ll fields of state register 106 are maintained by **hardware logic**, for example, atomically.” Ex-1006, 4:13-15.

113. Ground 1 teaches that the reservation register is accessible by both the first and second processing elements. Gewirtz teaches making the reservation register accessible by multiple software agents running on multiple processor cores: “[w]hen an agent wants to write an element into the queue, it first reads the reservation register to see if an entry is available. An agent may be **one of multiple agents representing software processes**, each being executed by **one of multiple processor cores**” Ex-1006, 4:56-61. When combined with Gewirtz’s teachings, Dongare’s host and embedded processors are the processors in the system; therefore, the software running on the host and embedded processors would obviously, under Gewirtz’s teachings, have had access to the reservation register, so that they could access and utilize the reservation register according to Gewirtz’s teachings. *See* Ex-1006, 4:56-61. In other words, a POSITA would recognize the value in having this information be visible to multiple processors and processes so that they can synchronize their operations correctly. *See* Ex-1006, 4:56-61.

114. Additionally, the reservation register was accessible as a memory-mapped register to both the first and second processing elements. Gewirtz explains

that the Queue State Register “can be accessed via a status register of memory mapped registers 101 corresponding to the queue.”² Ex-1006, 3:10-31; 4:51-54. It would have been obvious to a POSITA for a memory-mapped register to be accessible by all processors in a multiprocessor system because conventional multiprocessor systems have included hardware-based memory-mapped registers for this purpose for decades. Undergraduate courses, including courses I teach personally, teach the use of memory-mapped registers, and students learn how to access those registers through a computer’s memory space. Therefore, a POSITA with basic education and work experience in computer architecture would have understood and found it obvious, from Gewirtz’s teaching, for the queue state register to be accessible to processors in the system—including Dongare’s host and embedded processors—because Gewirtz teaches it as a memory-mapped register. Ex-1006, 4:51-54. Since Ground 1 applies Gewirtz’s teachings to Dongare’s multiple processing engines architecture with a host and embedded processor, the conventional understanding of Gewirtz’s teaching would have been for its

² Gewirtz teaches a register under Patent Owner’s proposed construction of “register” in District Court as “a memory location within a microprocessor used to store values and external memory addresses” (Ex-1029, 000002) as 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.

memory-mapped queue status register to be accessible to both processors. *See* Ex-1006, 4:51-54.

115. **Motivation to Combine.** Dongare provides motivation to combine its teachings with known techniques for managing queues. Ex-1005, 10:50-11:30. Gewirtz provides those details with queue-management techniques for tracking queue space and flow control. Ex-1006, 1:12-35; 2:46-3:7; 4:16-50. Therefore, a POSITA would have been motivated to apply Gewirtz to Dongare. This would have been 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 flow control as Gewirtz teaches. By 2013, registers, queue management, and flow control techniques were well-known, and their combination with Dongare would have been predictable and straightforward to a POSITA.

Supra §I.

116. Dongare’s architecture creates a need for synchronization and flow control with its queues. Dongare uses queues to schedule and divide tasks among multiple processing engines. Ex-1005, 6:28-42; 7:34-39. Dongare teaches that different commands take more or less time to process, and its processing engines may have varying processing rates due to varying capabilities. Ex-1005, 6:30-35 (“However, depending on the nature of the command being processed, ... a command provided earlier to one engine may take more time to process than

another command assigned to another engine. Thus, processed commands would be ready to write out at different time points.”); Abstract (“Commands from single/multiple queues and multi-flows are processed by multi-engines with different processing times and/or out of order.”); *see also* 6:6-9 (“... each engine may differ in functionality and/or architecture, but capable of processing any command assigned to it.”). Since commands are completed at different rates, 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. Thus, Dongare and Gewirtz shared a similar problem of queue management and flow control.

117. Gewirtz teaches a solution to this common problem of variable-rate data flows. Ex-1006, 2:46-3:7 (describing high performance system with dynamic queues and queue state register); 3:32-42 (describing system that enables “potentially hundreds of independent software clients”); 8:7-17 (describing a priority scheme for the queues). In particular, Gewirtz teaches queue-management solutions—including a Queue State Register—optimized for assisting with flow control of queues. Ex-1006, 2:59-62 (describing its Queue State Register being used for “flow control purposes.”); 6:66-7:20 (describing advantages of Gewirtz’s

teachings, including “assist[ing] with system flow control.”). Therefore, a POSITA would have been motivated to apply Gewirtz’s teachings to Dongare to solve this common problem.

118. Gewirtz’s teachings provide several known benefits. For example, Gewirtz provides atomic hardware operations for queue management that avoid the need for software spin locks. Ex-1006, 1:46-64; 6:18-25; 6:66-7:6; *see also* Ex-1006, 1:30-33. Based on these atomic operations, Gewirtz explains that “the pointer is obtained and updated atomically through hardware so that the software process represented by the first agent is not required to obtain an exclusive access control of the first queue from the Operating System.” Ex-1006, 1:60-64. Gewirtz’s queue management techniques also addresses flow control through the use of a queue status or state register:

In one embodiment, **state register 106** includes an optional built-in credit counter represented by the **number of free entries 113** which can be accessed via a status register of memory mapped registers 101 corresponding to the queue. The credit counter originally contains the **number of entries** in the queue. When an agent wants to write an element into the queue, it first **reads the reservation register to see if an entry is available**. An agent may be one of multiple agents representing software processes, each being executed by one of multiple processor cores of a network processor in a network element.

Ex-1006, 4:51-61.

119. A POSITA would naturally have looked to Gewirtz because register implementations were extremely common in computer architecture. The prior art provides teaching, suggestion, and motivation to use registers to track queue space, including for multiprocessor architectures. Ex-1013, 1:11-37; Ex-1016, 000001-2; Ex-1006, 3:4-7. Therefore, a POSITA would have been motivated to use Gewirtz's Queue Status Register to manage queues in Dongare's architecture. As I discussed above, *supra* §I.4, Gewirtz implements a hardware interface for queue access to enforce atomic operations. Ex-1006, 3:4-7 ("All maintenance of the queue state is done automatically in hardware, and will be atomically visible, along with the new queue entry (or the new head of the queue for a pop), to the next access."). Additionally, a POSITA would recognize the value of using Gewirtz's register teachings because it prevents queue overflow. By providing the amount of free space in the Queue Status Register, Gewirtz prevents a host from attempting to write more data into the queue than space permits, which would waste bandwidth by forcing the host to attempt to rewrite the same data. Gewirtz's techniques enable Dongare's host and embedded processors to first verify that space is available before permitting a host to issue a write. Ex-1006, 4:51-67. When Dongare's host processor is aware of the state of the input queue, it can be certain whether the write to the input queue occurred successfully or was blocked by a lack of room in the queue. To the extent that Patent Owner argues that Gewirtz is

limited to memory-based FIFOs instead of hardware FIFOs, it is important to note that the operations Gewirtz describes (updating read and write pointers) are the primary way that hardware FIFOs work. In other words, the teachings of Gewirtz (e.g., providing a set of atomically updated, globally visible control registers) apply equally well to both software and hardware FIFOs.

120. Both references provide motivation to combine. Dongare teaches, suggests, and motivates combinations with other prior art that can improve throughput and efficiency for multi-processor systems:

[i]n a multi-engine processor or a multi-threading system, tasks from same program can be executed concomitantly on different processors to **achieve maximum throughput**. However, the completed task has to be provided in the order it was posted to the multi-processor system. The method and apparatus of the present disclosure provide for **reordering completed tasks efficiently**.

Ex-1005, 4:49-55. Gewirtz provides these teachings, explaining that “[e]mbodiments of the invention relate generally to the field of multiprocessing; and more particularly, to programmable queues structures for multiprocessors.”

Ex-1006, 1:6-8. Gewirtz provides several advantages, including increased throughput and efficiency. As Gewirtz explains:

[o]ne of the **advantages of the programmable queue** as shown in FIGS. 1-2 is that, once defined, system agents can perform normal read and write commands that are processed atomically as pushes and pops

from that queue. This reduces the latency seen when reading from the queue and, even more importantly, eliminates the need for a locking access to gain exclusive ownership of the memory holding all of the queue data. This avoids contention over the lock. Memory contention over shared memory space is always detrimental to multi-processor system performance. In addition to the atomicity and locking benefits, the queues are extremely flexible, and can include a variety of optional features. They can be defined to have a large range of depths and element sizes. They can have built-in credit counters (e.g., **represented by the number of free entries of a queue state register**) to assist with **system flow control**.

Ex-1006, 6:66-7:14. A POSITA would have been motivated by this stated advantage to combine the queue management techniques of Gewirtz with Dongare. Additionally, as explained *supra* §I.4, it was known in the art that hardware solutions such as those taught by Gewirtz offer improved efficiency and performance.

121. The references provides explicit motivation to modify their teachings and combine them with other known techniques: “[i]t will be evident that various modifications may be made thereto without departing from the broader spirit and scope of the invention as set forth in the following claims.” Ex-1006, 12:1-4; *see also* Ex-1005, 15:48-52 (same).

122. **Reasonable Expectation of Success.** A POSITA would have had a reasonable expectation of success in combining Dongare and Gewirtz because the

combination uses conventional and well-known techniques to yield a predictable result. Gewirtz is directed at queue management and flow control improvements. 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’s teachings provide solutions for. Ex-1006, 1:12-35; 2:46-3:7; 4:16-50. By the time of the alleged invention, Gewirtz’s flow control and queue management techniques were routine and well-known in the art; it would have been a straightforward application of these known techniques to improve Dongare’s architecture, including its queue management and flow control. Given the similarities and teachings of the references, and that these technologies—including registers, queues, queue management, and flow control—were well-known at the time of the alleged invention, a POSITA would have easily combined the teachings of Dongare and Gewirtz. For example, a POSITA would have easily been able to add Gewirtz’s memory-mapped register teachings to the memory space of Dongare’s processors. As explained *supra*, essentially all modern processors uses memory mapping to make control registers visible, and so adding a set of control registers (e.g., the control registers of Gewirtz to the processor address spaces of Dongare) would have been simple and straightforward to accomplish. 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. A POSITA would also have expected success in the combination because it is the use of a known technique (Gewirtz's queue management and flow control) to improve Dongare's known architecture.

123. Implementing the combination would have been well within the skills of a POSITA. It would not have required specialized skills beyond what a POSITA would have had at the time of the alleged invention because the use of registers, queue-management, and flow-control teachings, such as those taught by Gewirtz, would have been included in undergraduate education and would have been common in multiprocessor and computer architecture work. These are basic building blocks of digital design that have been known, used, and taught in schools for decades. A POSITA with this background would have understood and recognized that memory-mapping the registers, as described in Gewirtz, was commonly used and would have been an easily-implemented addition to Dongare because these types of registers were routinely memory-mapped to simplify access for software running on processors. Ex-1006, 2:64-66. Dongare teaches an architecture with multiple processors, which would have been expected to use memory-mapped registers, which were used by most processors by the 1990s. Therefore a POSITA would have been motivated to combine Dongare with Gewirtz's memory-mapped register to track available queue space, with a reasonable expectation of success in doing so.

124. Furthermore, a POSITA would have been motivated and would have had a reasonable expectation of success in combining the references, as explained above, because the combination would not have changed the principle of operation for any of the teachings of the references relied upon in the combination. For example, Dongare's teachings will still be used for the RAID ASIC architecture taught by Dongare. It would still have multiple processors, multiple processing engines, input queues, and output queues as Dongare teaches. Ex-1005, 14:30-55. The combination uses Gewirtz's queue management and flow control teachings for the same purpose taught by Gewirtz: e.g., to track available queue space with a status reservation register and manage queues. Ex-1006, 4:51-5:6. Gewirtz's techniques apply equally to queues implemented in hardware or software with atomic access as described and implemented in Gewirtz. Additionally, Gewirtz's flow control techniques based on a queue status register that contains the number of available entries is applicable to both software and hardware implemented queues with atomic operations. The use of memory-mapped registers was (and, indeed, is) nearly ubiquitous in computers.

125. Based on the foregoing, Ground 1 teaches limitation [1e].

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

126. Ground 1 teaches limitation [1f], as explained below.

127. Ground 1 teaches computer storage storing instructions, which when executed by said at least one second processing element causes the execution of steps described in limitations [1g]-[1i]. Dongare teaches an implementation for its embodiments using “a **computer program product stored in a machine-readable medium** (also referred to as a computer-readable medium, a processor-readable medium, or a **computer usable medium having a computer-readable program code embodied therein**). ... The machine-readable medium can contain various sets of instructions, code sequences, configuration information, or other data, which, when executed, cause a processor to perform steps in a method according to an embodiment of the disclosure. ...” Ex-1005, 15:27-47. Dongare teaches that “the elements of the multi-engine processing system 100 may be implemented as a software routine ...” Ex-1005, 11:14-16. Likewise, Gewirtz teaches that its embodiments “relate to an apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general-purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a **computer program may be stored in a computer readable medium**. ...” Ex-1006, 11:40-53. As explained below, Dongare and Gewirtz teach the second processing element performing steps [1g]-[1i]; therefore, they teach instructions that are executed by the second processing element for performing those steps. *See* Ex-1006, 11:40-53; *supra*

§V.A.3[1e] (Motivation Subsection explaining how and why Dongare and Gewirtz would have been combined).

128. Additionally, it would have been obvious to a POSITA for Dongare’s embedded processor to access computer storage with instructions for the embedded processor to execute the algorithms described in Dongare and to maintain queues as described in Gewirtz. By definition, processors execute instructions to perform their functionality, and it would have been obvious to a POSITA for the embedded processor to access computer storage with the instructions that enable its functionality because conventional processor systems have included computer storage for this express purpose (storing computer programs) for decades.

129. Based on the foregoing, Ground 1 teaches limitation [1f].

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

130. Ground 1 teaches limitation [1g], as described below. The combination of Ground 1 was explained above. *Supra* §V.A.3[1e] (Motivation Subsection explaining how and why Dongare and Gewirtz would have been combined). Dongare teaches that the embedded processor manages the input queues. 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. Therefore, for Dongare's RAID application, the embedded processor manages the input queue; this would have been obvious because Dongare explains how the embedded processor performs related to input-queue handling, scheduling, and reordering, which correspond to queue management functions described for Figure 5. Ex-1005, 14:30-55. Moreover, Dongare teaches a software implementation for these preprocessing steps (Ex-1005, 10:50-11:30), and in the RAID ASIC, the embedded processor would be the natural and obvious processor for running such software (Ex-1005, Fig. 7).

131. Ground 1 teaches at least one second processing element (e.g., embedded processor) that accesses said reservation register and reads said stored value. Dongare teaches that the embedded processor manages the input queues. Ex-1005, 10:50-11:13; 11:20-30. One part of queue management is tracking the amount of free space in the queue. Gewirtz teaches a technique for tracking queue space in the queue status register, which as Gewirtz explains, contains the "available entries for reservation." Ex-1006, 4:5-7. Therefore, in the combination of Dongare and Gewirtz, the embedded processor manages the input queue by reading the queue status register, as taught by Gewirtz, to read the value for the number of available entries for reservation. *See* Ex-1006, 4:5-7; 4:56-61. As Gewirtz explains, the "reservation field can be used to specify the number of

entries available for writing into the queue.” Ex-1006, 4:12-13. Gewirtz further explains that a processor “first **reads the reservation register** to see if an entry is available.” Ex-1006, 4:56-58. Additionally, it would have been obvious and natural for Dongare’s embedded processor to perform a coordination role and read the reservation register because the embedded processor is on-chip (on the ASIC with the queues) and therefore had fast and efficient access to the reservation register without having to cross an external bus interface. In contrast, Dongare’s host processor accesses the queues and their corresponding registers via the PCIe interface, which is slower and can potentially run into bandwidth constraints. Thus, it would have been 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, 4:56-58.

132. Based on the foregoing, Ground 1 teaches limitation [1g].

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

133. Ground 1 teaches limitation [1h], as described below. The combination of Ground 1 was explained above. *Supra* §V.A.3[1e] (Motivation Subsection explaining how and why Dongare and Gewirtz would have been combined).

134. Ground 1 teaches at least one second processing element (e.g., embedded processor) that determines when said read value indicates available space in said input queue for said first processing element to issue a command. For example, Gewirtz teaches that the 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; *see also supra* §V.A.3[1e]. Gewirtz further explains that, before entries are written to the queue, a processor “first reads the reservation register to see if an entry is available.” Ex-1006, 4:56-58. Therefore, the processor reads the value in the reservation field to determine the available space in the queue. Ex-1006, 4:56-61 (describing a processor core reading the reservation register to see if an entry is available). For example, Gewirtz teaches a processor reading the reservation register which “return[s] a pass/fail value to the agent in response to the read, depending on whether there is a reservation available ...” Ex-1006, 4:61-64.

135. In the combination, as explained above, Dongare’s embedded processor manages the input queue by reading the queue status register to determine whether the reservation field indicates that space is available in the input queue, in accordance with Gewirtz’s teachings. Ex-1006, 4:12-13; 4:61-64; *supra* §V.A.3[1g]. As Dongare explains, the embedded processor manages the input queue (Ex-1005, 10:50-11:30), and a basic part of queue-management includes

tracking the amount of free space in the queue, so that it does not overflow. The embedded processor would naturally be the processor that prevents overflow of the input queue because the embedded processor is located on chip along with the input queues for precisely this type of management function. *See* Ex-1005, 10:50-11:30. Therefore, in the combination, the embedded processor most easily accesses the queue status register 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. Accordingly, it would have been obvious and most efficient for Dongare’s embedded processor to determine whether the queue status register’s reservation field (for the number of available entries) indicates available space in the input queue by checking the field and seeing if there is enough free space for the host processor to issue a command to the input queue.

136. 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.

137. Based on the foregoing, Ground 1 teaches limitation [1g].

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

138. Ground 1 teaches limitation [1i], as explained below. The combination of Ground 1 was explained above. *Supra* §V.A.3[1e] (Motivation

Subsection explaining how and why Dongare and Gewirtz would have been combined).

139. Ground 1 teaches at least one second processing element (e.g., embedded processor) that notifies said first processing element (e.g., host processor) to issue said command to said input queue. As explained above, the embedded processor checks the reservation register and notifies the host processor when there is space available for additional commands. *Supra* §V.A.3[1g]-[1h]. Dongare's host processor writes the commands ("e.g., read/write operations from Hard Disks 770") to the input queue. Ex-1005, 14:41-49.

140. Dongare's embedded processor handles scheduling for the RAID engines. The embedded processor executes a scheduling algorithm that processes the commands in the input queues and assigns the commands to the one of the processing engines. Ex-1005, 10:50-11:2; 14:30-31 ("FIG. 7 provides an illustration of an application of the apparatus of FIG. 5 in a multi-engine processing system.").

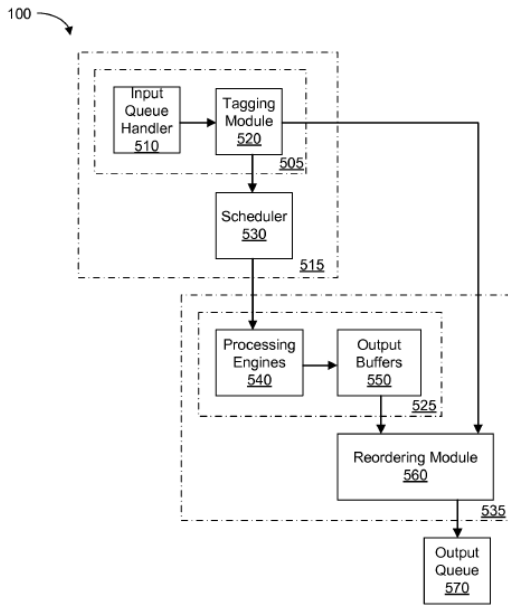


FIG. 5

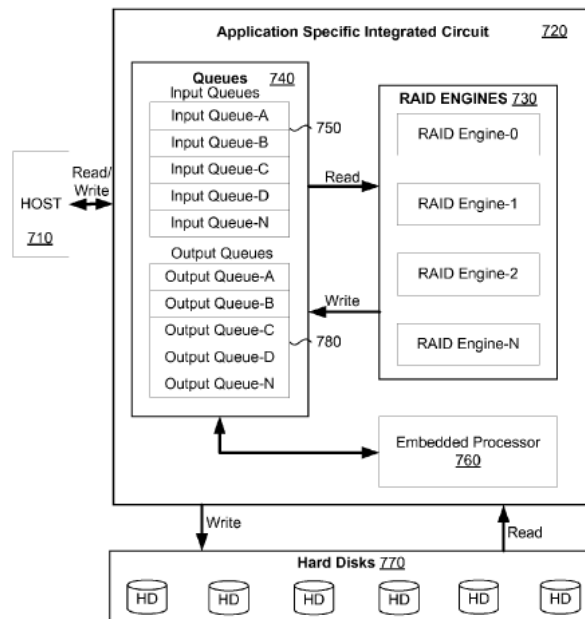


FIG. 7

Ex-1005, Figs. 5, 7. Dongare teaches that the “scheduler may be combined with the input queue handler 510 and the tagging module may be implemented as one element 515, which handles all the pre-processing steps.” Ex-1005, 11:20-23. The input queue handler is “configured to receive a plurality of input streams.” Ex-1005, 10:56-57.

141. Additionally, Gewirtz teaches queue configuration including “processor interrupt options [that] enable the queue hardware to generate processor interrupts to a programmable target whenever certain queue events are detected. For example, an interrupt can be generated when a **queue ... drains below a predetermined threshold ...**” Ex-1006, 3:63-4:2. This provides a mechanism for

interrupting a processor when queue space is available. When applied to Dongare, it would have been obvious for the programmable target to be Dongare's embedded processor because the embedded processor is a programmable target on the same chip as the queues. Ex-1005, 14:30-55; Fig. 7. Therefore, Dongare's embedded processor would have received interrupts faster and been able to respond and service the interrupt faster than an off-chip processor, for example if the interrupt were routed to an external host processor. *See* Ex-1005, 14:30-55; Fig. 7. By enabling interrupts to the embedded processor when the input command queue has been drained below the pre-determined threshold, the combination allows the embedded processor to fulfill its queue-management and scheduling role³ by reading the reservation register (Ex-1006, 5:3-6) and then notifying the host processor that there is space 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 (explaining sending a queue space available message (e.g., notification) after a full queue drains enough to accept additional data); 11:52-54 ("The Queue Space Available

³ Dongare teaches that the embedded processor performs input queue handling, which encompasses queue management and flow control. Ex-1005, 10:50-11:30. A POSITA would have understood this because the handling of a queue includes managing the capacity of the queue and the rates in which new items are added and removed. Ex-1005, 6:28-35; Ex-1006, 3:55-4:4; 2:59-3:7.

bus unit message is used to indicate to the source bus manager that queue space is available.”).

142. Dongare teaches mechanisms for the embedded processor to notify the host. As Dongare explains, the embedded processor communicates with the host processor over the PCIe bus (Ex-1005, 14:30-34), and is able to “signal the host ... by writing into the output queue” to “generate an interrupt to the Host.” Ex-1005, 14:52-55.

143. In summary, when an input queue is full⁴—for example when the queue is full and waiting on the RAID Engines to complete pending commands—the embedded processor notifies the host after the input queue drains below a predetermined threshold, as Gewirtz teaches (Ex-1006, 3:63-4:2), so that the host processor can issue its command to the input queue. *See* Ex-1006, 14:41-42. A POSITA would have been motivated to combine teachings from Dongare and Gewirtz in this manner because it would have allowed the host to issue a command to the input queue without having to repeatedly poll the Queue Status Register (or some other data structure) over the PCIe bus. Moreover, it allows the queues to be managed locally, on-chip by an embedded processor, which is able to make

⁴ Once the input queue becomes full, the host processor cannot write to the queue and must wait until the queue has space available. A POSITA would have understood this because it is a basic concept taught in undergraduate courses on logic design and data structures.

reservation decisions quickly in response to queue space becoming available, as compared to the off-chip host processor. Therefore, the combination of Dongare and Gewirtz takes advantage of Gewirtz's reservation register teachings to monitor queue status and leverages Dongare's embedded processor to notify the host processor of queue space availability.

144. Based on the foregoing, Ground 1 teaches limitation [1i].

[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.

145. Ground 1 teaches limitation [1j], as explained below. The combination was explained previously. *Supra* §V.A.3[1e] (Motivation Subsection explaining how and why Dongare and Gewirtz would have been combined).

146. Ground 1 teaches wherein said first processing element (e.g., host processor) receives notification from said at least one second processing element (e.g., embedded processor) regarding issuing the command, issues said command to said input queue. As explained above, *supra* §V.A.3[1i], when an input queue is full, the embedded processor notifies the host after the input queue drains below a predetermined threshold. *See* Ex-1006, 3:63-4:2. When the host receives this notification, it then issues its command to the input queue. Dongare teaches that the host processor “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.

147. Ground 1 also teaches wherein said first processing element receives a response corresponding to said command from said output queue. Dongare teaches that the “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 and manage RAID hard disks. Ex-1005, 14:35-37 (“RAID controller ASIC 720 has multiple RAID engines 730 to process data and commands to meet required throughput.”).

148. Dongare teaches that the embedded processor signals the host that a command has been processed by the RAID engines and the output is available in the output queue. Ex-1005, 14:49-55. Dongare explains how the embedded processor uses the output queue to signal the host of command completion, taking into account the possible out-of-order execution from multiple processing engines. Ex-1005, 13:18-25. The host processor then reads the output queue over the PCIe protocol bus to receive the response corresponding to the command it submitted.

Ex-1005, 14:31-39 (“... the Host computer 710 is connected to a RAID ASIC 720 via a PCIe protocol bus. ... The Host 710 and RAID controller 720 communicate to each other using a set of queues 740 on the RAID controller 720.”).

149. Based on the foregoing, Ground 1 teaches limitation [1j].

4. Dependent Claim 2

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

150. Ground 1 teaches claim 2, as explained below.

151. As I have explained above, the combination of Dongare and Gewirtz teaches claim 1 *supra* §V.A.3; *supra* §V.A.3[1e] (Motivation Subsection explaining how and why Dongare and Gewirtz would have been combined). Dongare teaches wherein said input queue is a first-in-first-out (FIFO) buffer. For example, Dongare teaches that input queues were known in the art to be FIFO buffers: “processed packets or commands are written into command **FIFO buffers** of a command queue Thus, command **FIFO buffer** holds ordered commands when read out on a **FIFO** basis.” Ex-1005, 2:61-67. Likewise, Gewirtz teaches queues as FIFO buffers: “it is frequently desirable to build queues, or **first-in-first-out (FIFO) buffers**.” Ex-1006, 1:12-16. Therefore, a POSITA would have found it obvious for the input queue to be a FIFO buffer because Dongare teaches it, and queues were generally known to be FIFO data structures. As Gewirtz explains, “[a] queue is a data structure with **FIFO properties** where agents push elements

onto the queue (writes) and pop elements off of the queue (reads).” Ex-1006, 1:12-16; 11:7-9. Therefore, a POSITA would have understood Dongare’s input queues to be FIFO buffers.

152. Based on the foregoing, Ground 1 teaches Claim 2.

5. Dependent Claim 5

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.

153. Ground 1 teaches claim 5, as explained below.

154. As I have explained above, the combination of Dongare and Gewirtz teaches claim 1 *supra* §V.A.3; *supra* §V.A.3[1e] (Motivation Subsection explaining how and why Dongare and Gewirtz would have been combined). Gewirtz teaches wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element: “The available reservation count is maintained by hardware: it is **decremented when a successful reservation read request is processed.**” Ex-1006, 5:3-6.

155. Gewirtz’s hardware logic is responsible for decrementing the number of free entries in the queue that is stored in the reservation register (e.g., Gewirtz’s Queue State Register).

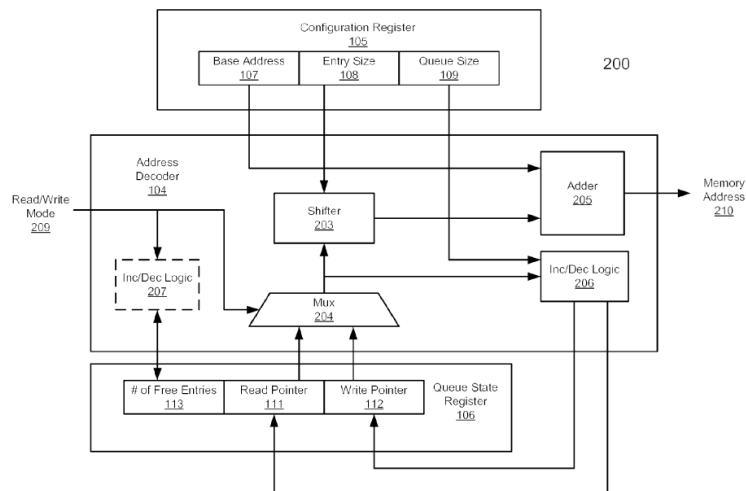


FIG. 2

Ex-1006, Fig. 2. Gewirtz describes that “the number of free entries 113 of the queue is updated via increment/**decrement** logic 207.” Ex-1006, 6:53-54; *see also* Claim 4 (“... wherein the step of atomically updating the pointer of the hardware-based queue state register further comprises atomically updating, via a second increment/**decrement** hardware logic, **a number of free entries of the queue** in the **hardware-based queue state register**.”).

156. Gewirtz further teaches that the stored value is decremented based on a successful read of the reservation register:

When an agent wants to write an element into the queue, it first **reads the reservation register to see if an entry is available**. An agent may be one of multiple agents representing software processes, each being executed by one of multiple processor cores of a network processor in a network element. The queue hardware can return a pass/fail value to the agent in response to the read, ... When an agent receives a pass

value, it can perform a write to one of the data access registers. ... The available reservation count is maintained by hardware: it is **decremented when a successful reservation read request is processed**, and incremented when either a reservation register write is seen, or when an element is popped off of the queue.

Ex-1006, 4:56-5:6.

157. Based on the foregoing, Ground 1 teaches Claim 5.

6. Dependent Claim 6

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.**

158. Ground 1 teaches claim 6, as explained below.

159. As I have explained above, the combination of Dongare and Gewirtz teaches claim 1 *supra* §V.A.3; *supra* §V.A.3[1e] (Motivation Subsection explaining how and why Dongare and Gewirtz would have been combined). Dongare teaches wherein at least one third processing element (e.g., one or more RAID Engines) executes the command and notifies the output queue to issue the response. For example, “**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 is a notification to the output queue to initiate a process to

issue a response (e.g., the fully complete and sequenced output) to the first processing element (i.e., the host):

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. A write to the outbound queue can generate an interrupt to the Host 710.

Ex-1005, 14:49-55. In other words, after the RAID engines finish executing the command and writing the results into the output queue, the embedded processor ensures that results are properly sequenced based on the command order as written by the host into the input queue. Ex-1005, 14:46-49. The host processor then receives the response from the output queue. Ex-1005, 14:49-55.

160. Based on the foregoing, Ground 1 teaches Claim 6.

7. Independent Claim 7

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

161. Ground 1 teaches limitation [7pre]. Limitation [7pre] is identical to limitation [1pre]. For the same reasons I have discussed for limitation [1pre], Ground 1 teaches limitation [7pre]. *Supra* §V.A.3[1pre].

[7a] an input queue implemented in hardware;

162. Ground 1 teaches limitation [7a]. Limitation [7a] is identical to limitation [1a]. For the same reasons I have discussed for limitation [1a], Ground 1 teaches limitation [7a]. *Supra* §V.A.3[1a].

[7b] an output queue implemented in hardware;

163. Ground 1 teaches limitation [7b]. Limitation [7b] is identical to limitation [1b]. For the same reasons I have discussed for limitation [1b], Ground 1 teaches limitation [7b]. *Supra* §V.A.3[1b].

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

164. Ground 1 teaches limitation [7c]. Limitation [7c] is identical to limitation [1c]. For the same reasons I have discussed for limitation [1c], Ground 1 teaches limitation [7c]. *Supra* §V.A.3[1c].

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

165. Ground 1 teaches limitation [7d], as explained below.

166. Ground 1 teaches at least one second processing element (e.g., embedded processor) in communication with said input queue. For example, Dongare's Figure 7 discloses a RAID controller architecture with input queues,

output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

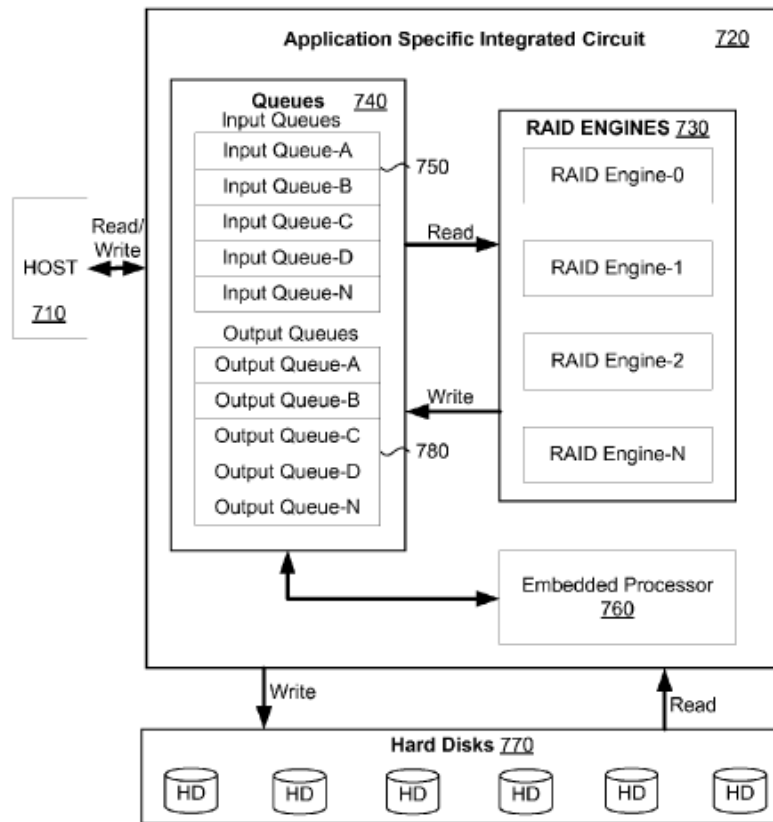


FIG. 7

Ex-1005, Fig. 7. Dongare's embedded processor is in communication with the input queues. Ex-1005, 14:42-44; Fig. 7. "[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. In other words, Dongare's embedded processor receives commands from the host processor via the input queue, schedules the

commands for the RAID engines, tags them, and loads (writes) the tagged commands back to the input queues on behalf of the host processor. Ex-1005, 7:40-64; 10:50-11:30; 14:39-55. Dongare's architecture is designed to "parallel process[]" in multiple processors to increase the processing efficiency or throughput of the system" while maintaining the original queue order. Ex-1005, 1:22-25; 9:5-9. The tagged commands are processed by the multiple processing engines, and the embedded processor uses the tags to re-sequence the output if the results come in a different order than the original command queue order. Ex-1005, 13:41-14:29.

167. Based on the foregoing, Ground 1 teaches limitation [7d].

[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;
--

168. Ground 1 teaches limitation [7e]. Limitation [7e] is identical to limitation [1e]. For the same reasons I have discussed for limitation [1e], Ground 1 teaches limitation [7e]. *Supra* §V.A.3[1e].

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

169. Ground 1 teaches limitation [7f]. Limitation [7f] is identical to limitation [1f]. For the same reasons I have discussed for limitation [1f], Ground 1 teaches limitation [7f]. *Supra* §V.A.3[1f].

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

170. Ground 1 teaches limitation [7g], as explained below.

171. Ground 1 teaches at least one second processing element (e.g., embedded processor) that receives a command from said first processing element (e.g., host processor). Specifically, Dongare's Figure 7 discloses a RAID controller architecture with input queues, output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

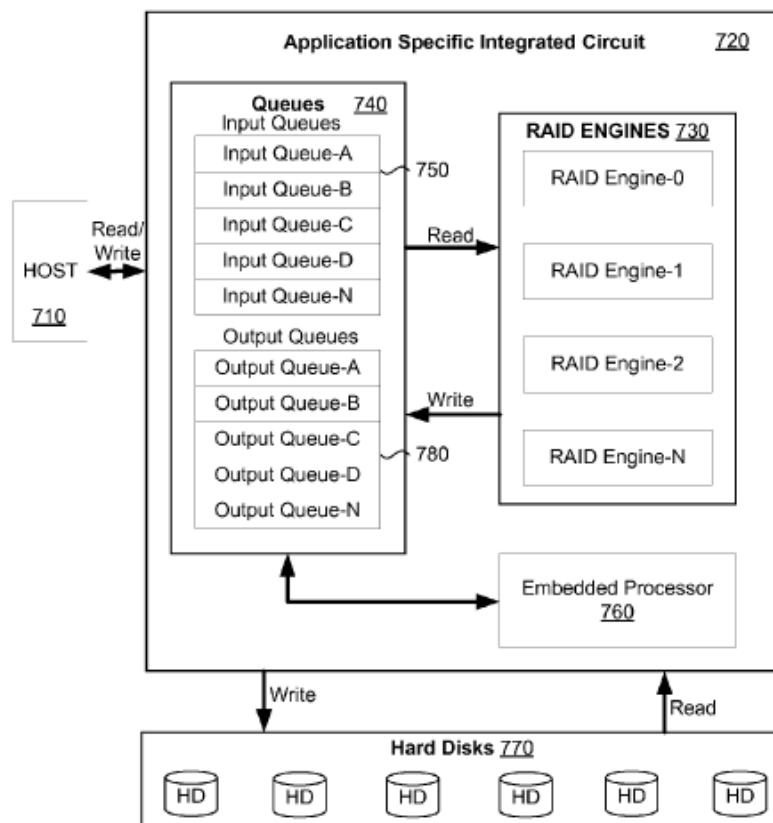


FIG. 7

Ex-1005, Fig. 7. Dongare's embedded processor 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), which may then perform subsequent steps as explained below (*infra* §V.A.7[7j]).

172. Based on the foregoing, Ground 1 teaches limitation [7g].

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

173. Ground 1 teaches limitation [7h]. Limitation [7h] is identical to limitation [1g]. For the same reasons I have discussed for limitation [1g], Ground 1 teaches limitation [7h]. *Supra* §V.A.3[1g]. In order to add the command to the input queue (*infra* §V.A.7[7j]), the embedded processor first accesses the reservation register and reads the number of free entries. *Supra* §V.A.3[1g].

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

174. Ground 1 teaches limitation [7i]. Limitation [7i] is nearly identical to limitation [1h], except that limitation [7i] recites "said command" while limitation [1h] recites "a command." This difference is immaterial for Ground 1 because I explained above how the purpose of checking the amount of space in the input queue is to determine whether there is space to issue the command from the host. *Supra* §V.A.3[1h]. For the same reasons I have discussed for limitation [1h], Ground 1 teaches limitation [7i]. *Supra* §V.A.3[1h].

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

175. Ground 1 teaches limitation [7j], as explained below.

176. Ground 1 teaches at least one second processing element (e.g., embedded processor) that issues said command to said input queue on behalf of said first processing element (e.g., host processor). Specifically, Dongare's Figure 7 discloses a RAID controller architecture with input queues, output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

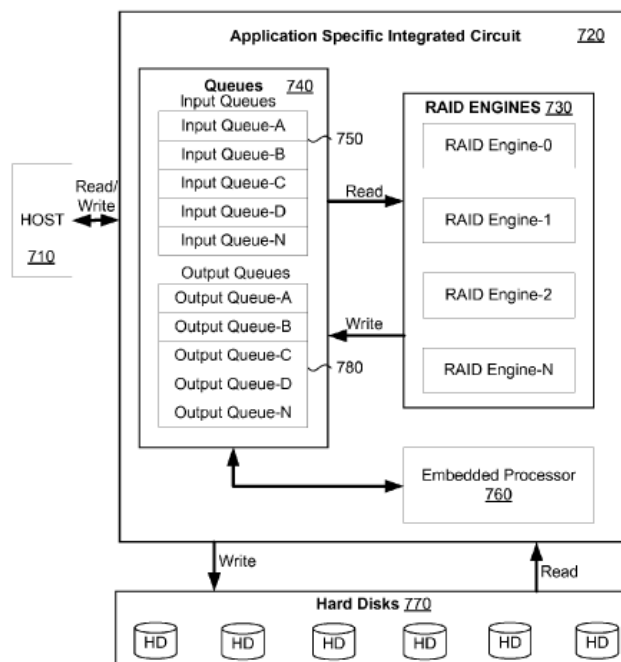


FIG. 7

Ex-1005, Fig. 7.

177. Dongare’s embedded processor receives a command from the host processor, tags it for scheduling purposes, and then writes the tagged command to the input queue, on behalf of the host processor, for execution by one or more RAID engines. Ex-1005, 14:41-44. Specifically, Dongare discloses that the “Host 710 writes commands for the RAID controller into the input queues 750. **The input queues 750 are read by the embedded processor 760 and commands are created for RAID engines 730 to process data.**” Ex-1005, 14:41-44. The embedded processor then loads the tagged command to the input queue. Ex-1005, 14:41-44. For longer commands, the embedded processor can divide the command into portions, tag the portions, and then issue the command—including all of its portions—to the input queue. Ex-1005, 7:40-64; Fig. 3. In other words, the host processor 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. Therefore, the embedded processor issues the command on behalf of the host processor to the input queue (and through it, to the RAID engines).

178. Based on the foregoing, Ground 1 teaches limitation [7j].

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

179. Ground 1 teaches limitation [7k]. Limitation [7k] is included in limitation [1j]. Limitation [1j] includes additional limitations that are not present

in limitation [7k], but limitation [7k] recites the final clause of [1j]: “receives a response corresponding to said command from said output queue.” For the same reasons I have discussed for limitation [1j], Ground 1 teaches limitation [7k].

Supra §V.A.3[1j].

8. Dependent Claim 8

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

180. Ground 1 teaches Claim 8. Claim 8 is nearly identical to Claim 2, except that Claim 8 recites “system of claim 7” while Claim 2 recites “system of claim 1.” As I have explained above, the combination of Dongare and Gewirtz teaches claim 7 *supra* §V.A.7. For the same reasons I have discussed for Claim 2, Ground 1 teaches Claim 8. *Supra* §V.A.4.

9. Dependent Claim 11

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.

181. Ground 1 teaches Claim 11. Claim 11 is nearly identical to Claim 5, except that Claim 11 recites “system of claim 7” while Claim 5 recites “system of claim 1.” As I have explained above, the combination of Dongare and Gewirtz teaches claim 7 *supra* §V.A.7. For the same reasons I have discussed for Claim 5, Ground 1 teaches Claim 11. *Supra* §V.A.5.

10. Dependent Claim 12

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.
--

182. Ground 1 teaches Claim 12. Claim 12 is nearly identical to Claim 6, except that Claim 12 recites “system of claim 7” while Claim 6 recites “system of claim 1.” As I have explained above, the combination of Dongare and Gewirtz teaches claim 7 *supra* §V.A.7. For the same reasons I have discussed for Claim 6, Ground 1 teaches Claim 12. *Supra* §V.A.5.

11. Independent Claim 13

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

183. Ground 1 teaches limitation [13pre]. Limitation [13pre] is identical to limitation [1pre]. For the same reasons I have discussed for limitation [1pre], Ground 1 teaches limitation [13pre]. *Supra* §V.A.3[1pre].

[13a] an input queue implemented in hardware;
--

184. Ground 1 teaches limitation [13a]. Limitation [13a] is identical to limitation [1a]. For the same reasons I have discussed for limitation [1a], Ground 1 teaches limitation [13a]. *Supra* §V.A.3[1a].

[13b] an output queue implemented in hardware;

185. Ground 1 teaches limitation [13b]. Limitation [13b] is identical to limitation [1b]. For the same reasons I have discussed for limitation [1b], Ground 1 teaches limitation [13b]. *Supra* §V.A.3[1b].

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

186. Ground 1 teaches limitation [13c]. Limitation [13c] is identical to limitation [1c]. For the same reasons I have discussed for limitation [1c], Ground 1 teaches limitation [13c]. *Supra* §V.A.3[1c].

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

187. Ground 1 teaches limitation [13d], as explained below.

188. Ground 1 teaches at least one second processing element (e.g., embedded processor) in communication with said input queue and said output queue. Specifically, Dongare's Figure 7 discloses a RAID controller architecture with input queues, output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

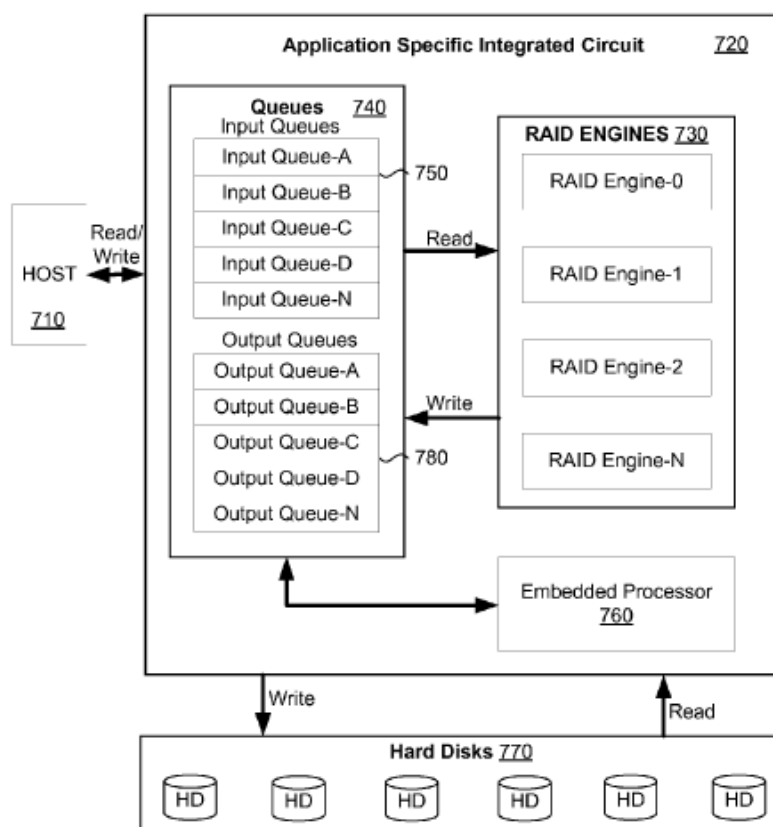


FIG. 7

Ex-1005, Fig. 7. Dongare’s embedded processor is in communication with the input queue. *Supra* §V.A.7[7d]. Specifically, Dongare discloses that “[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. In other words, Dongare’s embedded processor receives commands from the host processor via the input queue, schedules the commands for the RAID engines, tags them, and loads

(writes) the tagged commands back to the input queues on behalf of the host processor. Ex-1005, 7:40-64; 10:50-11:30; 14:39-55.

189. Dongare’s embedded processor is also in communication with the output queue. Specifically, Dongare discloses that “[t]he 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. In other words, Dongare’s embedded processor is responsible for reordering output from the RAID engines as necessary, and when the command has been completely processed and is ready in the output queue, signals the host processor of completion. Therefore, the embedded processor is in communication with the output queue to monitor and write to it. Ex-1005, 14:49-54.

190. Based on the foregoing, Ground 1 teaches limitation [13d].

[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;

191. Ground 1 teaches limitation [13e]. Limitation [13e] is identical to limitation [1e]. For the same reasons I have discussed for limitation [1e], Ground 1 teaches limitation [13e]. *Supra* §V.A.3[1e].

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

192. Ground 1 teaches limitation [13f]. Limitation [13f] is identical to limitation [1f]. For the same reasons I have discussed for limitation [1f], Ground 1 teaches limitation [13f]. *Supra* §V.A.3[1f].

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

193. Ground 1 teaches limitation [13g]. Limitation [13g] is identical to limitation [7g]. For the same reasons I have discussed for limitation [7g], Ground 1 teaches limitation [13g]. *Supra* §V.A.7[7g].

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

194. Ground 1 teaches limitation [13h]. Limitation [13h] is identical to limitation [1g]. For the same reasons I have discussed for limitation [1g], Ground 1 teaches limitation [13h]. *Supra* §V.A.3[1g].

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

195. Ground 1 teaches limitation [13i]. Limitation [13i] is nearly identical to limitation [1h], except that limitation [13i] recites “said command” while limitation [1h] recites “a command.” For the same reasons I have discussed for limitation [1h], Ground 1 teaches limitation [13i]. *Supra* §V.A.3[1h].

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

196. Ground 1 teaches limitation [13j]. Limitation [13j] is identical to limitation [7j]. For the same reasons I have discussed for limitation [7j], Ground 1 teaches limitation [13j]. *Supra* §V.A.7[7j].

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

197. Ground 1 teaches limitation [13k], as explained below.

198. Ground 1 teaches at least one second processing element (e.g., Dongare's embedded processor) receives a response corresponding to said command from said output queue. Specifically, Dongare's Figure 7 discloses a RAID controller architecture with input queues, output queues, a host processor, an embedded processor, and multiple processing engines (e.g., RAID engines).

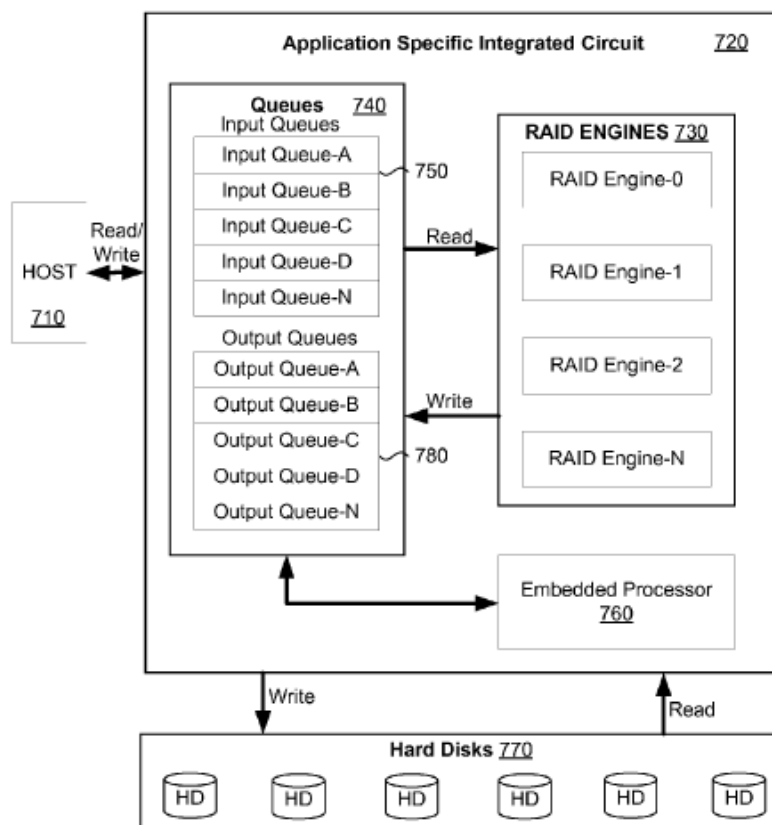


FIG. 7

Ex-1005, Fig. 7. Dongare discloses that “[t]he 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. In other words, Dongare’s embedded processor receives a response from the output queue that corresponds to the command issued by the Host, after it was processed

by the RAID engines; the embedded processor reorders the output as necessary for synchronization due to potential out-of-order completion, and then signals the host processor.

199. Based on the foregoing, Ground 1 teaches limitation [13k].

12. Dependent Claim 14

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

200. Ground 1 teaches Claim 14. Claim 14 is nearly identical to Claim 2, except that Claim 14 recites “system of claim 13” while Claim 2 recites “system of claim 1.” As I have explained above, the combination of Dongare and Gewirtz teaches claim 13 *supra* §V.A.11. For the same reasons I have discussed for Claim 2, Ground 1 teaches Claim 14. *Supra* §V.A.4.

13. Dependent Claim 17

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.
--

201. Ground 1 teaches Claim 17. Claim 17 is nearly identical to Claim 7, except that Claim 17 recites “system of claim 13” while Claim 5 recites “system of claim 1.” As I have explained above, the combination of Dongare and Gewirtz teaches claim 13 *supra* §V.A.11. For the same reasons I have discussed for Claim 5, Ground 1 teaches Claim 17. *Supra* §V.A.5.

14. Dependent Claim 18

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.

202. Ground 1 teaches Claim 18. Claim 18 is nearly identical to Claim 6, except that Claim 18 recites “system of claim 13” while Claim 6 recites “system of claim 1.” As I have explained above, the combination of Dongare and Gewirtz teaches claim 13 *supra* §V.A.11. For the same reasons I have discussed for Claim 6, Ground 1 teaches Claim 18. *Supra* §V.A.6.

B. Ground 2: Claims 3-4, 9-10, and 15-16 are Rendered Obvious in view of Dongare, Gewirtz, and Matsunami

203. The combination of Dongare and Gewirtz teaches the independent claims. Matsunami provides additional storage-area networking (SAN)/network-attached storage (NAS) teachings for certain dependent claims. The motivation to combine and reasonable expectation of success are explained below. *Infra* §V.B.2 (Motivation Subsection explaining how and why Matsunami would have been combined with Dongare and Gewirtz).

1. Matsunami (Ex-1007)

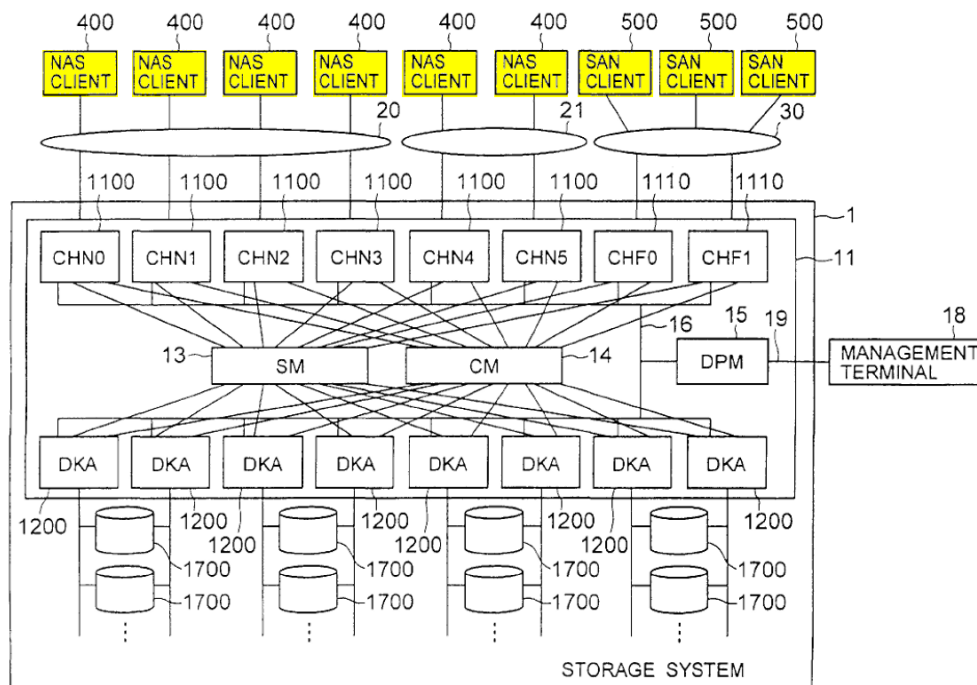
204. Matsunami is U.S. Patent No. 6,810,462 to Matsunami, et al., entitled “Storage System and Method using Interface Control Devices of Different Types.” Matsunami was filed on August 19, 2002, claiming priority to Japanese Patent Application 2002-125172, filed on April 26, 2002. Matsunami issued on October

26, 2004. I understand that Matsunami is prior art to the '596 patent under at least AIA §102(a)(1)-(2).

205. I have reviewed the Matsunami reference. I understand that the Matsunami reference was not cited nor considered during the prosecution of the '596 patent based primarily on the fact that Matsunami is not cited on the face of the '596 patent and was not discussed in the prosecution history of the '596 patent.

206. Matsunami teaches using RAID⁵ for disk storage in a combined storage area network (SAN) / network attached storage (NAS) controller.

FIG. 1



⁵ RAID is an acronym for redundant arrays of independent disks. Ex-1007, 7:1-17. RAID is used in storage systems that typically require high-levels of reliability, as the failure of an independent disk will not lead to a loss of data.

Ex-1007, Fig. 1 (annotated). Note the clients at the top of the figure, labelled NAS and SAN; this shows that Matsunami expressly teaches NAS and SAN access.

207. Matsunami teaches a controller for a storage system that can interface with both SAN and NAS clients. Ex-1007, 2:13-15 (“When it is desired to construct a system including a SAN and a NAS ... a NAS and a storage system having functions of the SAN are required.”). Matsunami also teaches that the storage system uses a plurality of hard disks that implement “redundant arrays of independent disks (RAID)” for storage. Ex-1007, 7:13-17.

208. Matsunami is analogous art in the same field as the ’596 patent (computer architecture, including the use of queues and multiple processing elements, *supra* §III) because it describes a storage controller system/architecture based on multiple processing elements. Ex-1007, 3:52-65.

2. Dependent Claims 3-4

- 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.**

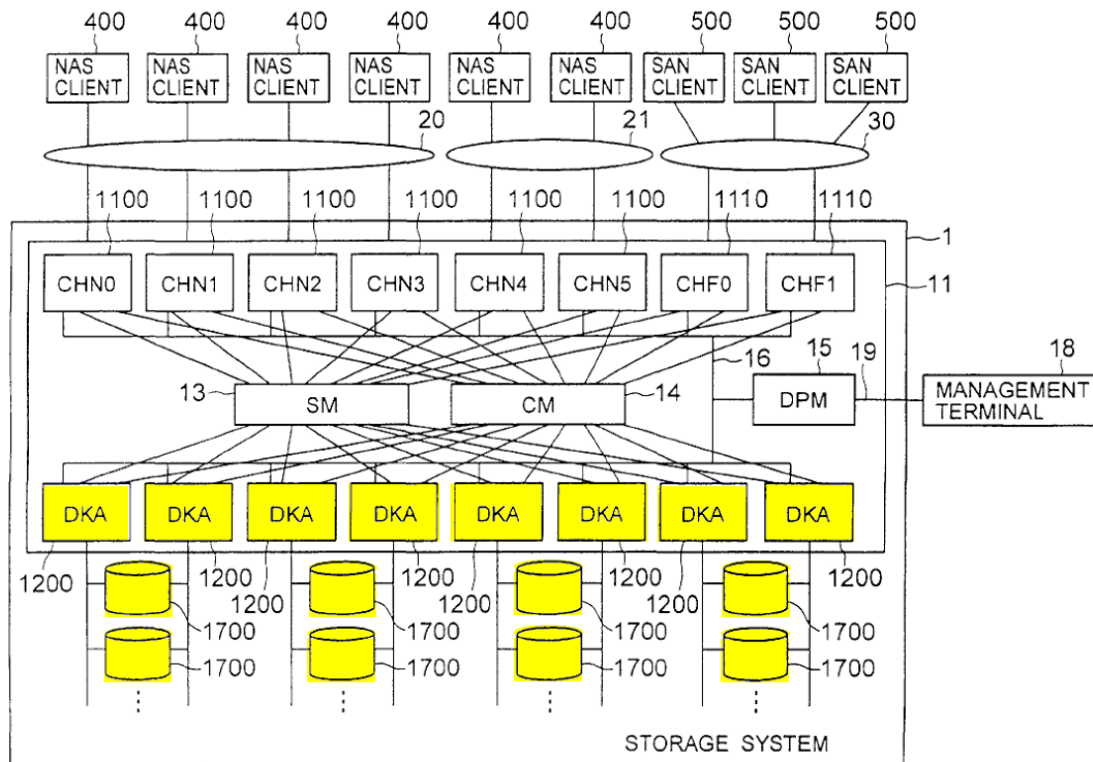
209. As I have explained above, Ground 1’s combination of Dongare and Gewirtz teaches claim 1 *supra* §V.A.3. Ground 2 further applies Matsunami’s teachings of a SAN/NAS controller. Ex-1007, Abstract (storage system with

“NAS and SAN functions”); 1:18-23 (SAN); 1:30-34 (NAS); 2:1-4 (SAN/NAS complementary); 2:13-27 (combined SAN/NAS).

210. Matsunami explains that “storage system having the NAS functions and the storage system having the SAN functions, the system user must manage the storage capacity for each storage system.” Ex-1007, 2:20-23. Matsunami uses RAID to implement high reliability storage for SAN and NAS functions. Ex-1007, 7:1-17. The Dongare-Gewirtz combination provides RAID teachings. Ex-1005, 14:30-55; Fig. 7. Therefore, it would have been obvious to use Ground 1’s RAID teachings as part of a SAN/NAS controller, which Matsunami teaches. In other words, Matsunami’s teachings were readily implemented using the Dongare-Gewirtz combination as part of the SAN/NAS controller.

211. For example, Matsunami teaches a SAN/NAS controller with a Disk Pool Manager (“DPM”) with a host processor and multiple disk adapters (“DKA”), each one 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).

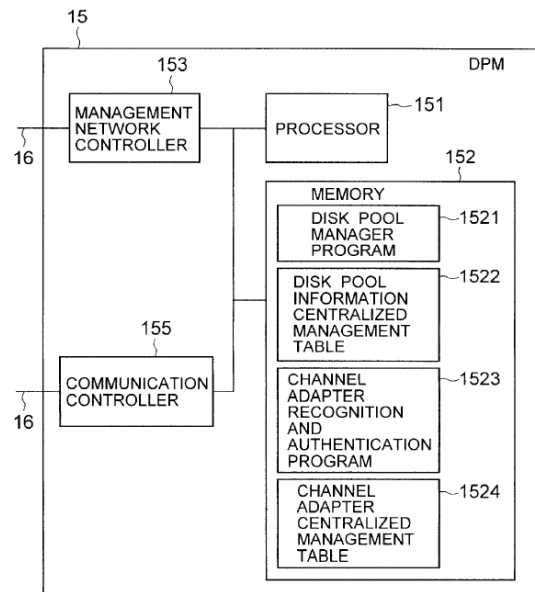
FIG. 1



Ex-1007, Fig. 1 (annotated).

212. It would have been obvious to apply Ground 1's teachings, with teachings for Dongare's host processor applied to the DPM and Dongare's RAID ASIC applied to the DKAs. *First*, Matsunami's DPM contains a processor (Ex-1007, 6:29-31) and is illustrated below:

FIG. 7



Ex-1007, Fig. 7. Specifically, Matsunami discloses that:

[t]he processor 151 controls the overall operation of the disk pool manager 15. ... The disk pool manager program 1521 is executed when the storage system 1 configures and manages the disk pool 6 using the disks 1700.

Ex-1007, 6:32-47. *Second*, Matsunami explains that its “disk adapter 1200 is connected to disks 1700. The disk adaptor 1200 controls data transfer between disks connected thereto and associated clients.” Ex-1007, 3:66-4:1. Matsunami’s disks are “a set of storage areas of a plurality of disks 1700 defined as one virtual storage area.” Ex-1007, 6:66-67. These disks are organized as “a plurality of disks 1700, concretely, RAID group (RG) 17000. The RG 17000 is redundant arrays of independent disks (RAID) including a plurality of disks 1700.” Ex-1007, 7:13-17.

In summary, Matsunami teaches a SAN/NAS controller that uses RAID for storage, 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.

213. In view of Matsunami's teachings, a POSITA would have found it obvious to use the RAID ASIC as taught by the Dongare-Gewirtz combination for Matsunami's disk adapters and Matsunami's DPM as Dongare's host processor, because Dongare's embodiment of Figure 7 is directed at storage systems that use RAID. Ex-1005, 14:30-37 ("FIG. 7 provides an illustration of an application of the apparatus of FIG. 5 in a multi-engine processing system. In applications such as RAID controllers, the Host computer 710 is connected to a RAID ASIC 720 via a PCIe protocol bus. RAID controller ASIC 720 has multiple RAID engines 730 to process data and commands to meet required throughput.").

214. **Motivation to Combine.** The motivation to combine Dongare with Gewirtz was explained above. *Supra* §V.A.3[1e] (Motivation Subsection). A POSITA would have been further motivated to apply Matsunami's teachings regarding SAN/NAS controllers as explained below.

215. Matsunami teaches a SAN/NAS controller with RAID used for storage. Ex-1007, Abstract; 1:12-35; 7:13-17. Dongare teaches a RAID storage controller. Ex-1005, 14:30-35 (discussing a RAID controller); Fig. 7. In particular, Dongare explains that its architecture can be used "[i]n applications

such as RAID controllers.” Ex-1005, 14:31-34. Therefore, a POSITA would have been motivated to combine Matsunami’s SAN/NAS teachings and Dongare’s RAID architecture. This would have been a simple combination of prior art elements according to known methods to yield predictable results (a SAN/NAS storage controller with RAID).

216. Matsunami itself teaches, suggests, and provides a motivation to use a RAID controller for its disks:

The disk pool 6 is a set of storage areas of a plurality of disks 1700 defined as one virtual storage area.

A plurality of disk pools 6 may be set to the storage system 1. Specifically, according to the difference in the characteristic of the disk devices, for example, performance thereof (rotating speed of 7200 rpm or 15000 rpm) or redundancy (RAID1 or RAID5) of the disk group, a disk pool may be set for each of a group of disks having the same characteristic.

...

FIG. 8 shows a specific example of a correspondence between one disk pool 6 and a plurality of disks 1700, concretely, RAID group (RG) 17000. The RG 17000 is redundant arrays of independent disks (RAID) including a plurality of disks 1700.

Ex-1007, 6:66-7:17.

217. A POSITA would have been further motivated to combine Dongare with Matsunami due to the benefits of Dongare’s teachings. RAID was known in

the art for providing data redundancy using an array of disks, where the same data was written to more than one disk. *See* Ex-1007, 6:66-7:17. If one disk failed, data would not be lost because it was recovered from the other disks in the array. In this way, RAID prevents data loss. Additionally, Dongare teaches a multi-engine RAID implementation with increased processing efficiency. Ex-1005, 14:30-55; Fig. 7.

218. **Reasonable Expectation of Success.** A POSITA would have had a reasonable expectation of success in combining Matsunami with the RAID controller taught by the combination of Dongare and Gewirtz because the combination uses conventional and well-known techniques to yield a predictable result. As discussed above, the Dongare-Gewirtz combination teaches a RAID implementation, while Matsunami teaches a SAN/NAS controller with RAID. Given Matsunami's suggestion of including RAID, a POSITA would have expected success in combining these teachings. Ex-1007, 6:66-7:17.

219. Furthermore, by the time of alleged invention, RAID had been well-known and widely used for storage controllers for decades. It was commonplace in servers, workstations, and data centers. Therefore, the application to Matsunami would not have required specialized skills beyond what a POSITA would have had at the time of the alleged invention.

220. A POSITA would have been motivated and would have had a reasonable expectation of success in combining the references because the combination would not have changed the principle of operation for any of the teachings of the references relied upon in the combination. Dongare's teachings would still be used for RAID. Matsunami's teachings would still be used for SAN/NAS. Gewirtz's teachings were still used for flow control as explained above for Ground 1. The application of Dongare's RAID teachings would not have fundamentally changed the character of Matsunami's teachings or the application domain of storage control systems, and particularly to SAN and NAS based storage control systems.

221. Based on the foregoing, Ground 2 teaches Claims 3-4.

3. Dependent Claims 9-10

- 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.**

222. Ground 2 teaches Claims 9-10. Claims 9-10 are nearly identical to Claims 3-4, except that Claims 9-10 recite "system of claim 7" while Claims 3-4 recite "system of claim 1." As I have explained above, the combination of Dongare and Gewirtz teaches claim 7 *supra* §V.A.7. For the same reasons I have discussed for Claims 3-4, Ground 2 teaches Claims 9-10. *Supra* §V.B.2.

4. Dependent Claims 15-16

- 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.**

223. Ground 2 teaches Claims 15-16. Claims 15-16 are encompassed by Claims 3-4. Claims 15-16 recite “system of claim 13” while Claims 3-4 recite “system of claim 1.” As I have explained above, the combination of Dongare and Gewirtz teaches claim 13 *supra* §V.A.11. For the same reasons I have discussed for Claims 3-4, Ground 2 teaches Claims 15-16. *Supra* §V.B.2.

C. Ground 3: Claims 1-2, 6-8, 12-14, and 18 are Rendered Obvious by Khawand and Rosenthal

224. It is my opinion that a POSITA would have found the Challenged Claims obvious based on Ground 3, i.e., the teachings of Khawand in combination with Rosenthal. Khawand teaches a multiprocessor architecture that improves efficiency using task delegation, queues, and flow control. Ex-1008, 1:8-10; 1:35-48; 3:65-4:5. Rosenthal provides implementation details regarding queues and flow control for buffers in a multiprocessor system through the use of a flow control register that monitors the amount of space available in the queues. Ex-1009, 2:15-33; 16:33-49; 17:4-18.

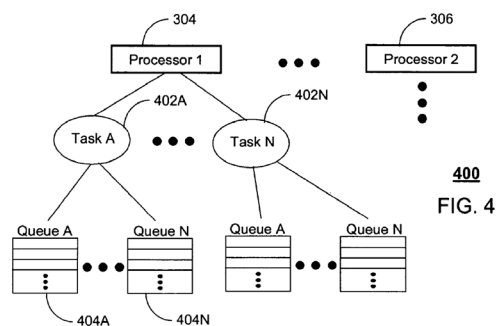
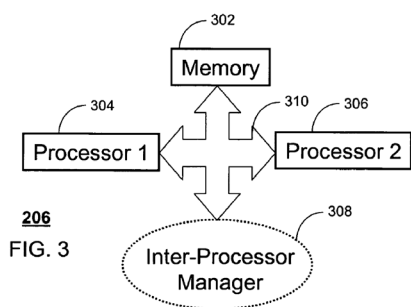
225. The motivation to combine and reasonable expectation of success are explained below. *Infra* §V.C.3[1e] (Motivation to Combine subsection).

1. Khawand (Ex-1008)

226. Khawand is U.S. Patent No. 7,398,528 to Khawand, et al., entitled “Method and System for Efficient Multiprocessor Processing in a Mobile Wireless Communication Device.” Khawand was filed on November 13, 2004, and issued on July 8, 2008. I understand that Khawand is prior art to the ’596 patent under at least AIA §102(a)(1)-(2).

227. I have reviewed the Khawand reference. I understand that the Khawand reference was not cited nor considered during the prosecution of the ’596 patent based primarily on the fact that Khawand is not cited on the face of the ’596 patent and was not discussed in the prosecution history of the ’596 patent.

228. Khawand teaches a multiprocessor architecture where service applications are delegated between the processors, which interact with an inter-processor manager to implement flow control for queues between the processors..
Ex-1008, 3:65-4:27.



Ex-1008, Figs. 3 & 4; Ex-1008, 3:18-22.

229. Khawand delegates service applications from one processor to another when a processor runs out of processing capacity:

We begin with step 602, where the first and second processors 304-306 of the mobile device 106 are programmed to **delegate a service application** from the first processor 304 **to a queue 404 of the second processor 306**. The rationale for this delegation can be, for instance, the first processor 304 running out of processing capacity for a particular task 402 or sub-task (represented by a particular queue 404).

Ex-1008, 3:65-4:5. Khawand describes the queues associated with the processors that are used as buffers to delegate service applications from one processor to another:

FIG. 4 is a block diagram of a software architecture 400 applied to each processor 304-306 according to an embodiment of the present invention. The software architecture 400 comprises at least one task. In the present example, however, a number of tasks 402A-402N are shown in FIG. 4. Each task 402 comprises at least one queue 404 for processing at least one service application. In the present example, a number of queues 404A-404N are shown. The second processor 306 can have any number and combination of tasks 402 and corresponding service queues 404.

Ex-1008, 3:52-61.

230. Khawand explains that the software processes delegate service applications over multiple processors when a processor runs out of capacity

(Ex-1008, 3:64-4:5) and uses queues with flow control to avoid overflowing any specific queue. Specifically Khawand teaches:

... the flow control criteria can include a capacity threshold for each queue 404. ... In one embodiment, the **service application is processed** in step 608 by the second processor 306 if at decision block 606 **it is determined that the capacity threshold of the queue 404 is not exceeded** (i.e., the flow control criteria has been satisfied).

Ex-1008, 4:15-22.

231. Khawand is analogous art in the same field as the '596 patent (computer architecture, including the use of queues and multiple processing elements, *supra* §III), because it teaches a computer architecture with multiple processors and queues to manage task delegation among the processors. Ex-1008, 1:32-48.

2. Rosenthal (Ex-1009)

232. Rosenthal is U.S. Patent No. 5,805,930 to Rosenthal, et al., entitled “System for FIFO Informing the Availability of Stages to Store Commands Which Include Data and Virtual Address Sent Directly from Application Programs.” Rosenthal was filed on March 28, 1997, as a continuation of Application Serial No. 441,044, filed on May 15, 1995. Rosenthal issued on September 8, 1998. I understand that Rosenthal is prior art to the '596 patent under at least AIA §102(a)(1)-(2).

233. I have reviewed the Rosenthal reference. I understand that the Rosenthal reference was cited during the prosecution of the '596 patent based primarily on the fact that Rosenthal is cited on the face of the '596 patent. However, Rosenthal was not discussed in the prosecution history of the '596 patent, nor was Rosenthal ever the basis of a rejection in the prosecution history of the '596 patent.

234. Rosenthal “provid[es] 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. Rosenthal implements flow control via “a **flow control register** which must be read by the central processing unit running the application before writing data to an input/output device.” Ex-1009, 2:21-28. “The register stores a **value which indicates the amount of space available in the FIFO buffer** to which data may be written.” Ex-1009, 2:26-28.

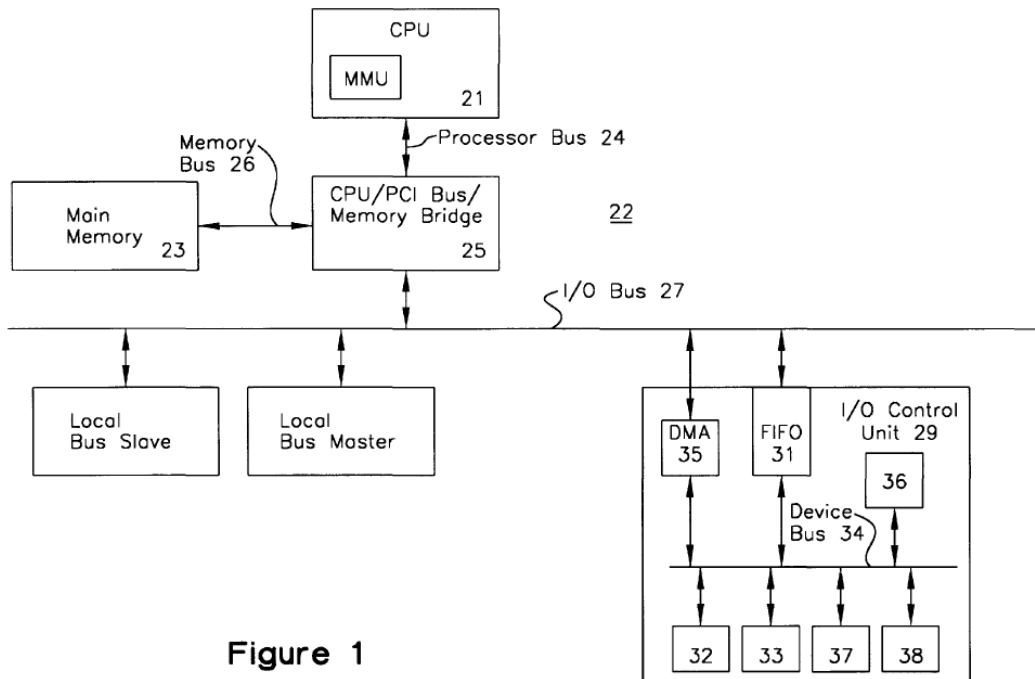


Figure 1

Ex-1009, Fig. 1.

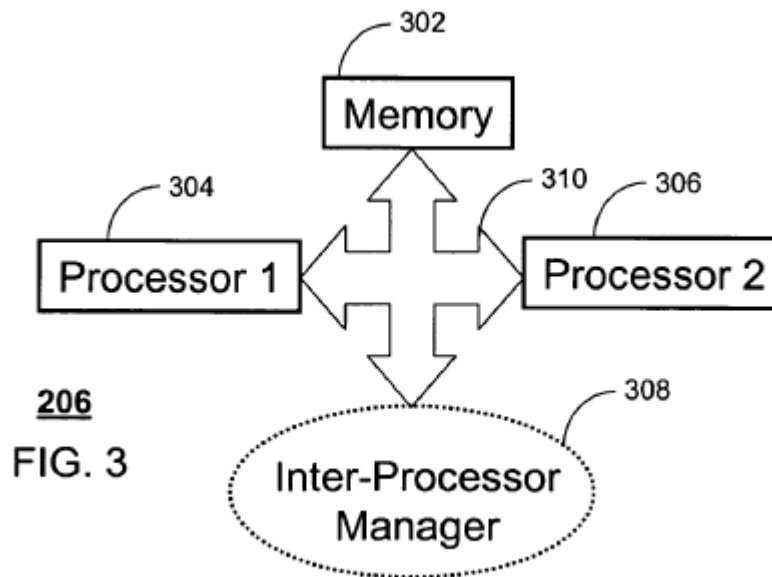
235. Rosenthal is analogous art in the same field as the '596 patent (computer architecture, including the use of queues and multiple processing elements, *supra* §III) because it teaches queue management and flow control for microprocessor systems. Ex-1009, 2:11-17.

3. Independent Claim 1

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

236. To the extent that the preamble is limiting, Ground 3 teaches limitation [1pre], as explained below.

237. Khawand teaches a system for dividing and synchronizing a processing service applications across a plurality of processing elements. For example, Khawand teaches that system designers are “employing **multi-processor** architectures such as, for example, an ARM (Advanced Risk [sic] Machines) combined, or integrated, with a similar processor intercommunicating with conventional inter-processor protocols.” Ex-1008, 1:20-24. Khawand introduces an inter-processor manager for flow control in multiprocessor systems. Ex-1008, 4:6-9 (“In step 604, the queue 404 of the second processor 306 is evaluated according to flow control criteria. In a first embodiment, the evaluation step can take place at the inter-processor manager 308.”).



Ex-1008, Fig. 3. Khawand teaches distributing service applications and sub-tasks among the various processors for enhanced performance. Ex-1008, 3:62-4:5

(“FIG. 6 is a flowchart depicting a method 600 for substantially improving the processing efficiency of service applications operating in the mobile device 106 according to an embodiment of the present invention. ... The rationale for this delegation can be, for instance, the first processor 304 running out of processing capacity for a particular task 402 or sub-task (represented by a particular queue 404).”). Khawand further teaches “service applications” including “software algorithms or portions thereof **subdivided into tasks...**” Ex-1008, 3:4-17.

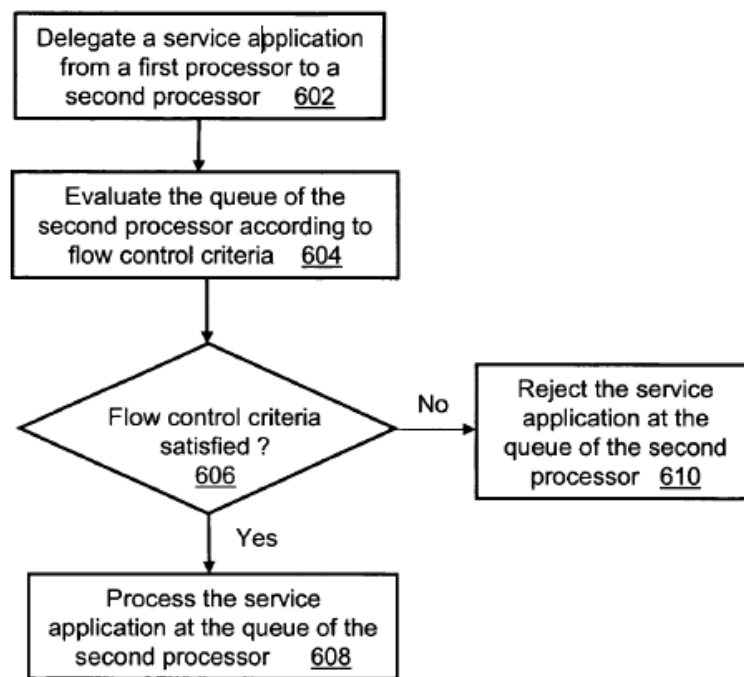
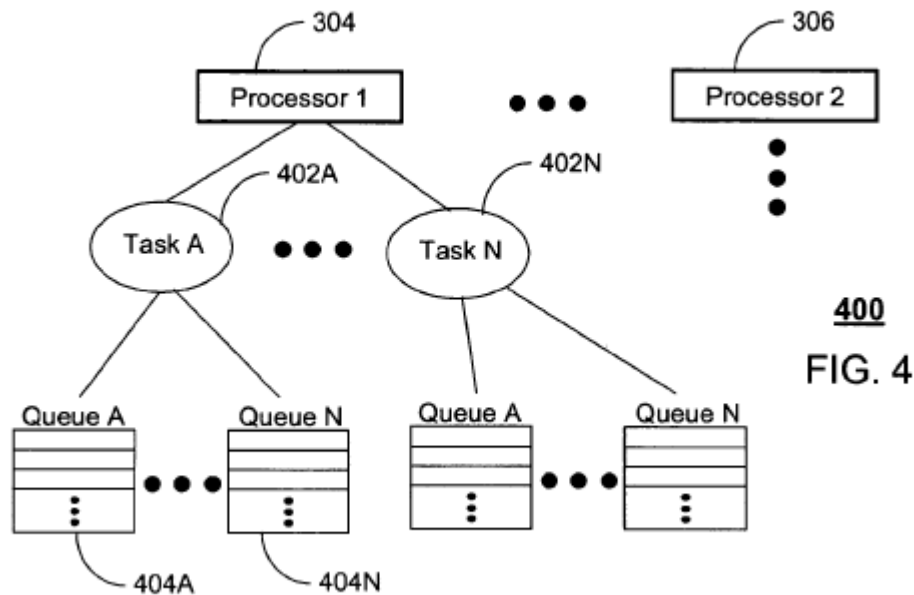


FIG. 6 600

Ex-1008, Fig. 6. Khawand further explains that the queues associated with service applications for each processor as illustrated in Figure 4, are subject to flow control:



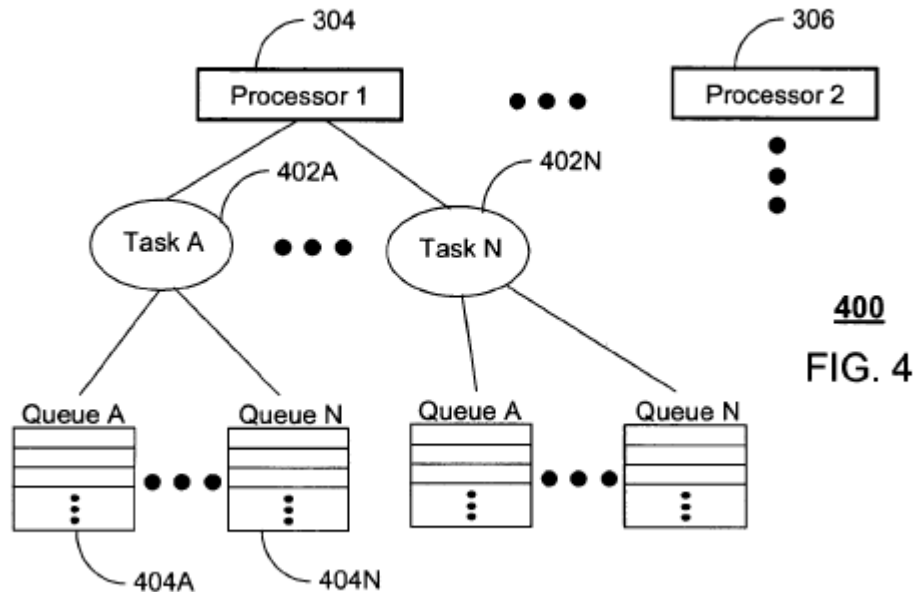
Ex-1008, Fig. 4. By using flow control, Khawand explains that “[i]n step 610, the second processor 306 submits a suspend command to the inter-processor manager 308, **which in turn synchronizes the first processor 304 according to the suspend command.** The suspend command indicates a corresponding one of a group of processing elements comprising the second processor 306, one or more tasks 402, and one or more associated queues 404 is no longer available to process service applications.” Ex-1008, 4:60-67. Thus, Khawand teaches distribution of service applications and sub-tasks using queues among the processors for increased

performance, with synchronization of the delegated service applications using the queues. Ex-1008, 3:62-4:5; 4:60-67 (suspend command); 5:18-29 (resume command). The system further comprises the elements explained below. *Infra* §§V.C.3[1a]-[1j].

[1a] an input queue implemented in hardware;

238. Ground 3 teaches limitation [1a], as described below.

239. Ground 3 teaches an input queue implemented in hardware (e.g., Processor 2's input queue). For example, Khawand teaches hardware queues for each processor to process various service applications:



Ex-1008, Fig. 4. Khawand describes that “[e]ach task 402 comprises at least one queue 404 for processing at least one service application. In the present example, a

number of queues 404A-404N are shown.” Ex-1008, 3:56-59. Khawand teaches implementing its queues “in **hardware**, software, or a combination of hardware and software.” Ex-1008, 6:14-17 (emphasis added); *see also* Ex-1008, 6:29-31 (“It is well known [in] (sic) the art, however, that software and hardware implementations may be designed to be equivalent structures generating the same results.”).

240. Additionally, Rosenthal teaches using “hardware FIFO” queues. Ex-1009, 4:60-64; Fig. 1. Rosenthal explains that “[t]he **FIFO unit 31** stores the data along with the addresses for **each of the commands** being transferred...” Ex-1009, 8:28-31; Fig. 3.

241. To the extent Patent Owner argues that Khawand somehow does not teach an input queue implemented in hardware, it would have been obvious to a POSITA to use a “hardware FIFO” as the input queue, as Rosenthal teaches. Khawand provides motivation to do so by teaching and suggesting that its components can be implemented in hardware. Ex-1008, 6:14-34. I explain the motivations for the combination below. *Infra* §V.C.3[1e] (Motivation subsection). The input queue is implemented in physical circuits, without software intervention, because both Khawand and Rosenthal teach hardware implementations. Ex-1008, 6:14-17; Ex-1009, 4:60-64.

242. INTENTIONALLY LEFT BLANK.

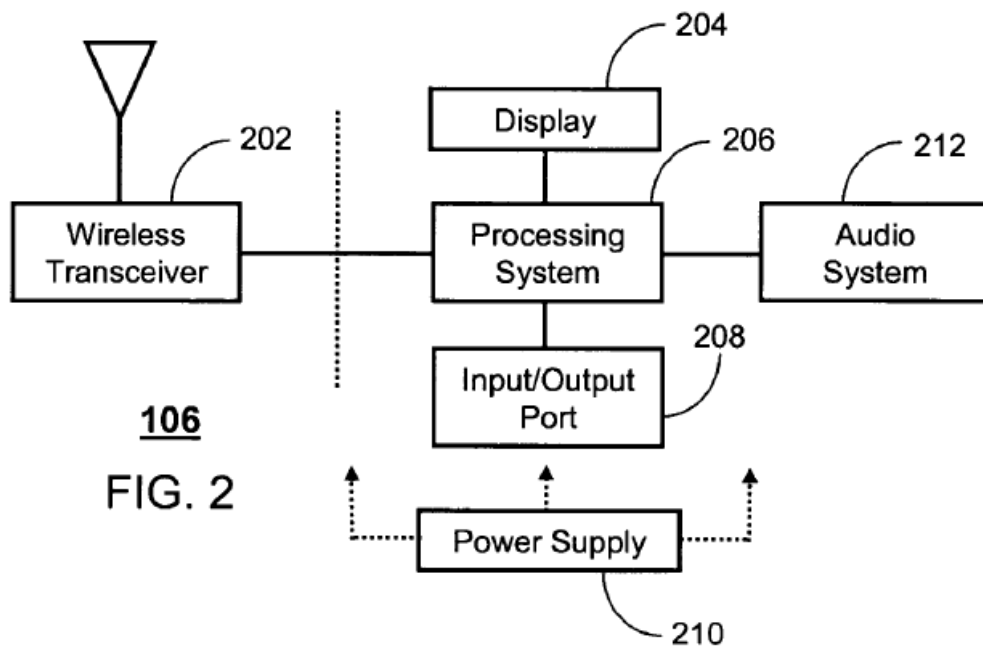
243. Based on the foregoing, Ground 3 teaches limitation [1a].

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

244. Ground 3 teaches limitation [1b], as explained below.

245. Ground 3 teaches an output queue implemented in hardware.

Khawand teaches service application delegation between processors and therefore teaches, or at least suggests, a mechanism for providing return data output from those service applications, e.g., for returning results from service application delegated from Processor 1 to Processor 2. *See* Ex-1008, Abstract; 1:53-61; 1:66-2:9; 3:65-4:5; Fig. 6. Indeed, Khawand teaches various examples of “service applications” where output is expected, including “multi-media processing,” “MPEG 4 video decoder,” “input/output drivers, display drivers, etc.” that include service for various units such as the display, audio system, or input/output port. Ex-1008, 3:4-17. Khawand applies its teachings to mobile devices, which contain 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-1008, Fig. 2.

246. Output queues were widely known and used for returning results. During prosecution of the EU counterpart, 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. I agree with the Examiner. Output queues have been used since the early days of computers. They have been commonly used to buffer return data between processors or devices in multiprocessor systems. A POSITA would have known this from basic logic design and operating system courses in undergraduate education. Accordingly, it would have been obvious to include output queues for buffering purposes with Khawand for transferring return data, which 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. At the very least, it would have been obvious to use output queues to move data from one functional unit to another because that is how data has been moved between computer components for decades. Additionally, Khawand's functionality as a mobile phone also suggests the existence of input and output queues to handle wireless communications, including for example queues for cellular modem and WiFi interfaces. These communications interfaces require buffering for both transmitting, receiving, acknowledging packets, and sending negative acknowledgments. 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.

247. The motivation to apply hardware FIFOs is further explained below.

Infra §V.C.3[1e] (Motivation subsection).

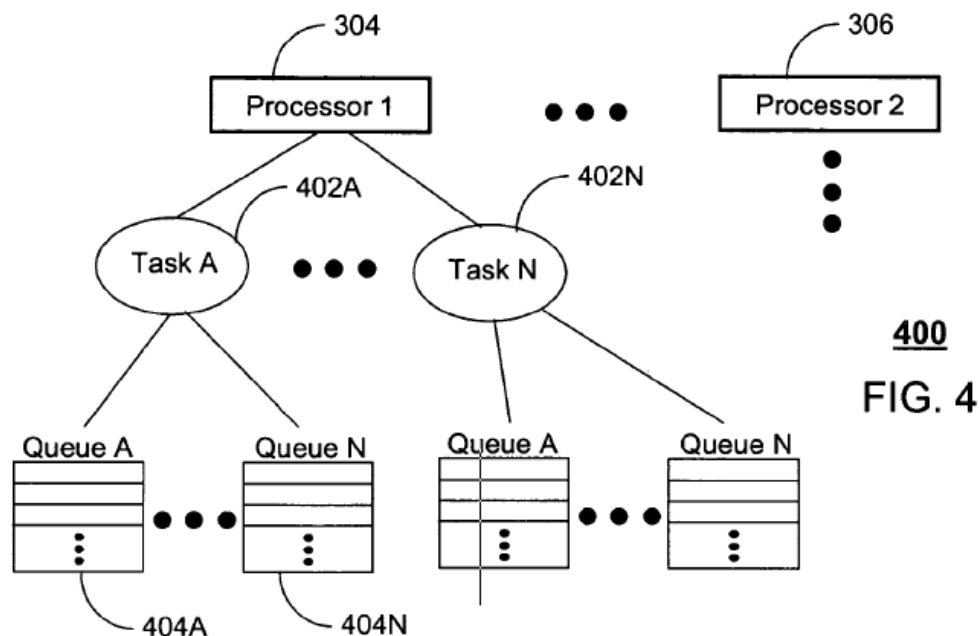
248. INTENTIONALLY LEFT BLANK.

249. Based on the foregoing, Ground 3 teaches limitation [1b].

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

250. Ground 3 teaches limitation [1c], as described below.

251. Ground 3 teaches a first processing element (e.g., Processor 1) having access to said input queue (e.g., Processor 2's task queue) and said output queue. Processor 1 is a first "processing element[.]" Ex-1008, 3:37-39. Specifically, Khawand's Figure 4 discloses a multiprocessor system, with each processor responsible for certain tasks, and each of those tasks having queues corresponding to service applications for that task.



Ex-1008, Fig. 4. Khawand explains that:

FIG. 4 is a block diagram of a software architecture 400 applied to each processor 304-306 according to an embodiment of the present invention. The software architecture 400 comprises at least one task. In the present example, however, a number of tasks 402A-402N are shown in FIG. 4. Each task 402 comprises at least one queue 404 for

processing at least one service application. In the present example, a number of queues 404A-404N are shown. The second processor 306 can have any number and combination of tasks 402 and corresponding service queues 404.

Ex-1008, 3:52-61.

252. Khawand also explains that processor 1 can delegate service applications to processor 2 by writing service applications into processor 2's queue. Khawand discloses that "the first and second processors 304-306 of the mobile device 106 are programmed to delegate a service application from the first processor 304 to a queue 404 of the second processor 306. The rationale for this delegation can be, for instance, the first processor 304 running out of processing capacity for a particular task 402 or sub-task (represented by a particular queue 404)." Ex-1008, 3:65-4:5.

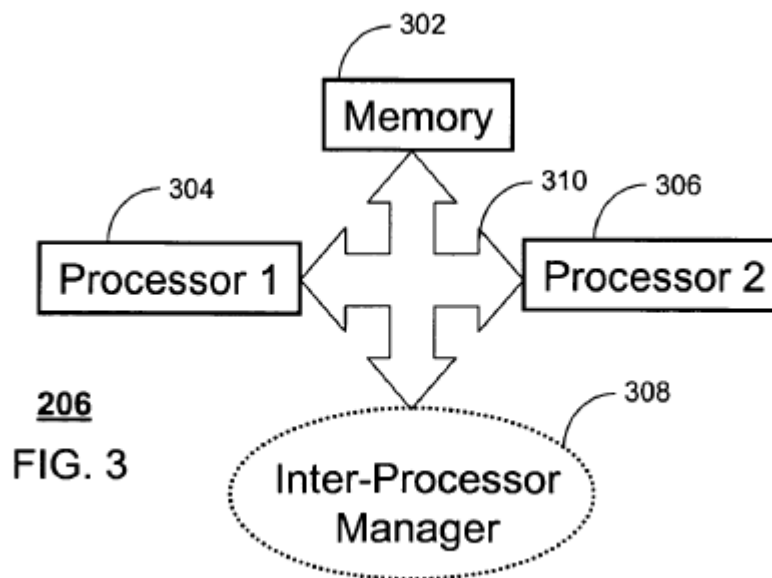
253. As described above, Khawand teaches or at least suggests an output queue implemented in hardware for returning results from service applications that Processor 1 delegates to Processor 2. *Supra* §V.C.3[1b]. Since the purpose of the output queue is to return data back to Processor 1, the processor would obviously have had access to the output queue. There is also a symmetry here—Processor 1 writes to a queue to send the service application, and so Processor 2 writes to a queue to respond. This symmetry would be clear to a POSITA. Khawand's processors would have had access to the output queues to place results or

intermediary data for downstream processors to access. A POSITA would have recognized this and found it obvious because output queues have been used for this purpose for decades.

254. Based on the foregoing, Ground 3 teaches limitation [1c].

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

255. Ground 3 teaches limitation [1d], as explained below. Ground 3 teaches at least one second processing element (e.g., Inter-Processor Manager) in communication with said first processing element (e.g., Processor 1). Khawand's Figure 3 teaches a multiprocessor system architecture where the multiple processors, and inter-processor manager, and memory are connected and in communication with each other.



Ex-1008, Fig. 3. Khawand teaches that “[t]he processing elements 302-308 of the processing system 206 are interconnected by a conventional communication bus 310.” Ex-1008, 3:37-39; *see also* Ex-1008, 3:44-47 (“Alternatively, a portion or whole of the communication bus 310 can represent a logical connection such as software components sharing data structures.”).

256. Khawand’s inter-processor manager 308 is a processing element. Khawand refers to it as one of “[t]he processing elements 302-308.” Ex-1008, 3:37-39. Khawand teaches that the inter-processor manager:

can comprise a conventional application specific integrated circuit (ASIC), a **microprocessor**, a state machine, or other processing means that is capable of operating according to the method 600 of FIG. 6. Alternatively, the processing function of the inter-processor manager 308 can take the form of hardware, software or combinations thereof operating in whole or in part from the first or second processors 304, or independent hardware.

Ex-1008, 3:29-36.

257. Based on the foregoing, Ground 3 teaches limitation [1d].

[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;
--

258. Khawand teaches using flow control for its queues. Ex-1008, Abstract; 4:6-27; 5:18-29; Fig. 6. Rosenthal teaches flow control details.

Ground 3 combines Khawand and Rosenthal; the motivation is explained in a subsection below.

259. Ground 3 teaches limitation [1e]. Ground 3 teaches a reservation register implemented in hardware (e.g., Rosenthal's flow control register) storing a value indicative of available space in said input queue (e.g., number of available spaces). For example, Rosenthal discloses that "a **flow control register 45** is included for each FIFO buffer 39 of the input/output control unit 29. The register 45 **stores an indication of the number of available spaces for commands** in the associated FIFO buffer 39." Ex-1009, 16:33-37. Rosenthal's flow control register is implemented in hardware because it is a physical register that stores values, accessible by the CPU. Ex-1009, 16:33-37; 16:44-49 ("central processing unit read[s] the value in the flow control register 45 associated with the FIFO buffer 39 being utilized...").⁶

260. Rosenthal teaches that the number of available spaces for commands is held in the flow control register:

[t]he use of this flow control register 45 allows a **requirement to be placed on an application program attempting an access of the input/output control unit 29 that it first determine whether space is**

⁶ Rosenthal discloses an option of using a bit-field (Ex-1009, 16:50-57). However, this is described as an optional implementation that "may be" used. It is not relied on for these Grounds.

available in the FIFO unit 31 before it writes any command. Before writing any data, the **application program must determine the amount of “free” space to which it may write.** This it does by having the central processing unit **read the value in the flow control register 45 associated with the FIFO buffer 39 being utilized.** Once this value has been read, the application program may send up to that amount of data before it need test again to determine whether free space remains to which it may send more data.

Ex-1009, 16:37-49. In other words, an application running on a processor must first read the flow control register 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. That is, Rosenthal not only teaches a register that stores a value indicative of available space in the queue, it also teaches that the writing process requires checking the status before an attempts to write to the queue.

261. Ground 3 teaches that said reservation register is accessible by both said first processing element and said at least one second processing element. In the multiprocessor architecture taught by the combination of Khawand and Rosenthal, Rosenthal teaches the flow control register being located in an I/O control unit that is accessible by any unit that is attached to the I/O bus:

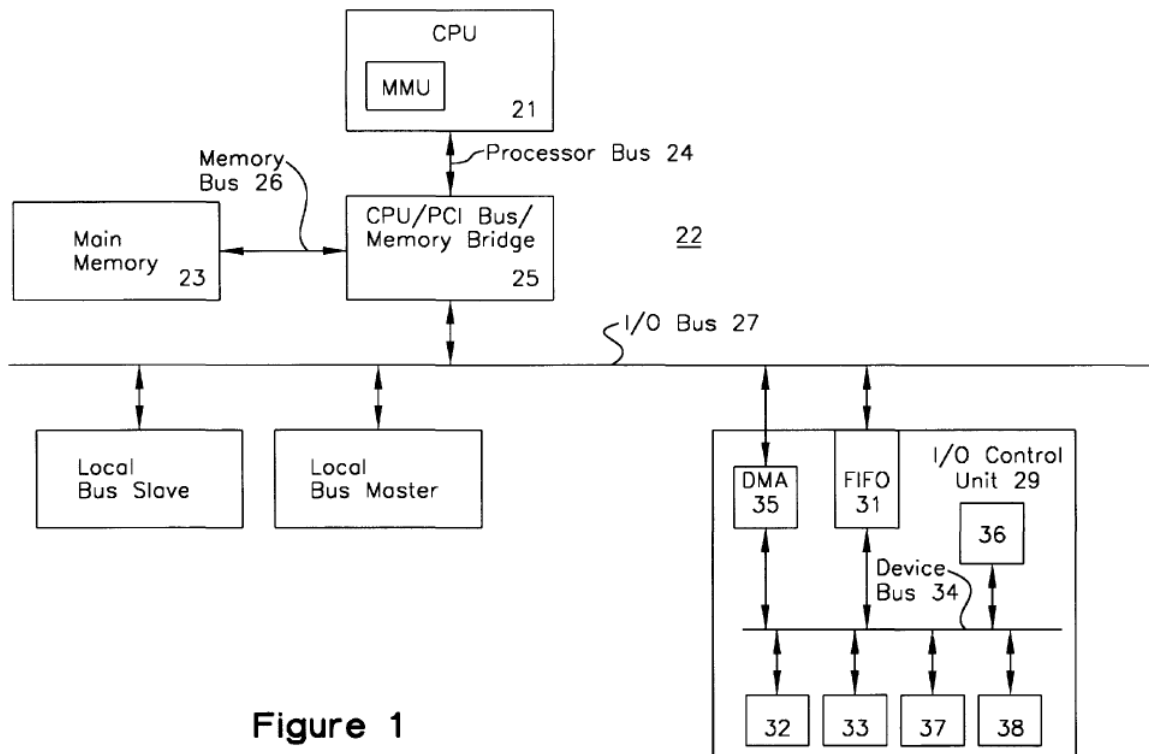


Figure 1

Ex-1009, Fig. 1. Specifically, Rosenthal discloses that:

Typically, various input/output devices are arranged on the input/output bus 27 as bus masters and bus slaves. In prior art systems, these local bus masters and slaves are those components (such as a graphics output device for connecting an output display monitor or a hard disk controller unit) which require the most rapid input/output operations for system success. If such **local bus masters and slaves are connected to the input/output bus 27, they are utilized with the present architecture for the purpose of running legacy programs and input/output functions** not implemented by the input/output control unit 29.

Ex-1009, 4:49-59. In other words, Rosenthal discloses that other devices can access the I/O control unit and use the flow control register to reserve space in the FIFO unit.

262. In the context of Khawand, the devices that are connected to the “communication bus” are the “processing elements 302-308” which includes Processor 1, Processor 2, and the inter-processor manager. Ex-1008, 3:37-39. Therefore, each of these processing elements 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.

263. Moreover, each of these processing elements is a processor—processor 1 and 2 are “conventional processing elements such as, for example, an ARM processor” (Ex-1008, 3:27-28), and the inter-processor manager can also be “a microprocessor” (Ex-1008, 3:29-32). These three processors would have obviously had access to the flow control register in the Khawand-Rosenthal combination because the purpose of the register is to provide flow control for devices that write to the associated queue. As Rosenthal explains, the flow control register is read by 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.⁷ Ex-1009, 16:33-49. Here, Rosenthal teaches the flow control register as a memory-mapped register, visible to and accessible to other Processors, because it is accessed by software applications running on a CPU. 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. At the very least, it would have been obvious for a memory-mapped register to be accessible by all processors in a multiprocessor system, because conventional multiprocessor systems have included hardware-based memory-mapped registers for this purpose since at least the 1990s. Accordingly, it would have been obvious for the flow control register to be similarly accessible by processors in Khawand's multiprocessor system for flow control purposes. Indeed, this is Rosenthal's express purpose: "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.

⁷ Rosenthal teaches a register under Patent Owner's proposed construction of "register" as "a memory location within a microprocessor used to store values and external memory addresses" (Ex-1029, 000002) as Rosenthal's flow control register is accessible to the microprocessor via a memory address and stores a value, i.e., the number of available spaces for commands in the associated FIFO buffer.

264. Ground 3 also teaches both parties' district court constructions.

Patent Owner's construction was explained in the previous paragraph. Rosenthal also teaches the 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). Registers have, for many decades, been implemented using a set of flip-flops. A POSITA would have understood this because it was a ubiquitous approach taught in undergraduate logic design courses for decades, as evident from various textbooks from the 1990s and before. *See* Ex-1019, 000013; Ex-1020, 000012; Ex-1021, 000012; Ex-1022, 000008; Ex-1023, 000025; Ex-1024, 000026. Accordingly, a POSITA would have understood Rosenthal's teaching of a flow control register to be 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.

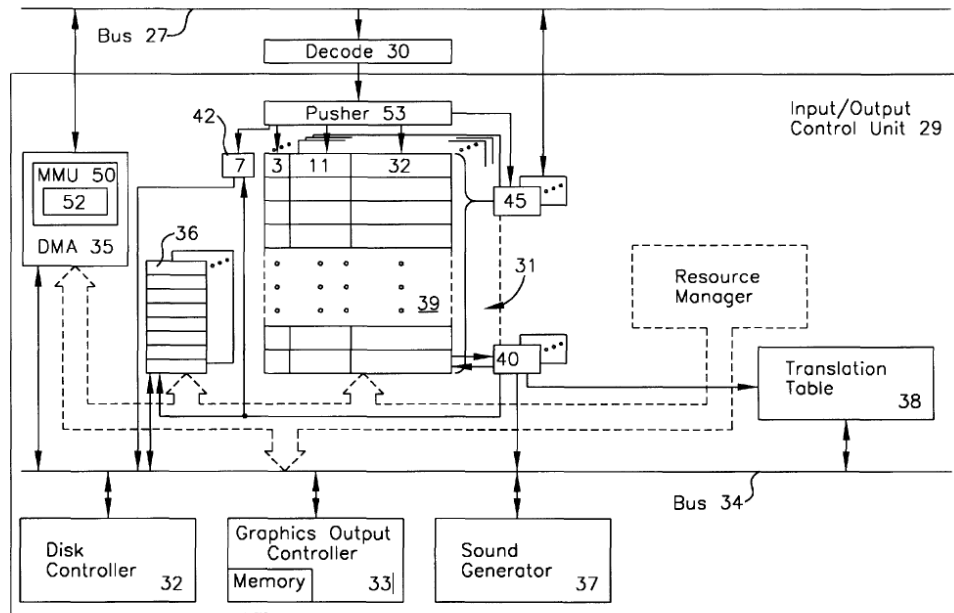


Figure 3

265. Motivation to Combine Khawand and Rosenthal. Khawand teaches a multiprocessor architecture that delegates service applications among processors using task queues. Ex-1008, 4:15-27. Khawand uses “flow control” to determine if queues have capacity for additional service applications. Ex-1008, 4:15-27; Abstract. For example, Khawand teaches the inter-processor manager evaluates “flow control criteria” including “capacity” for each queue. Ex-1008, 4:6-27. Rosenthal provides implementation details for using flow control to manage queue capacity, including teachings for hardware FIFOs and flow control based on a “flow control register” that indicates the amount of space remaining in the queue. Ex-1009, Abstract. Therefore, a POSITA would have been motivated

to apply Rosenthal's teachings to implement queues and flow control in Khawand's architecture.

266. Rosenthal provides motivation for the combination, explaining 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.” Ex-1009, 1:66-2:7. Rosenthal provides solution for this known issue: “[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. In particular, Rosenthal teaches a “flow control register” that contains the number of available spaces for command, which can be read before new data is added to the FIFO to prevent overflow. Ex-1009, 16:33-37. Therefore, a POSITA would have been motivated to combine Khawand and Rosenthal, as it would have been a straightforward combination of known prior art elements according to known methods to yield a predictable result—queue flow control with a register for indicating the amount of space in the queue, as described by Rosenthal.

267. By providing the amount of free space in a flow control register, Rosenthal provides a simple mechanism that prevents processors from attempting to write more data into a queue than space permits. Rosenthal's flow control

teachings allow 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). A POSITA would have been motivated by these benefits to apply Rosenthal's teachings to Khawand.

268. The references provides explicit motivation to modify their teachings and combine them with other known techniques: "all equivalent modifications and additions to the description above are intended to be inclusive of the scope [of] the following claims." Ex-1008, 6:31-34; Ex-1009, 23:64-24:1 ("Although the present invention has been described in terms of a preferred embodiment, it will be appreciated that various modifications and alterations might be made by those skilled in the art without departing from the spirit and scope of the invention.").

269. Additionally, a POSITA would have been motivated to combine Rosenthal's hardware FIFO buffers as input queues with Khawand. First, Khawand provides motivation by teaching a hardware implementation. Ex-1008, 6:14-34. Second, Rosenthal's hardware FIFO buffers provided an advantage of faster access time for processors, while enabling atomic queue operations. Third, using a mix of fast hardware queues would have enabled the processors to exploit temporal locality in determining which queues for each task should be in hardware, while still allowing flexibility to have additional overflow queues in memory, much like how cache memory systems work. Khawand provides motivation for

this hybrid approach. Ex-1008, 6:14-17. The combination have would been straightforward as it is simply combines known prior art elements according to known methods to yield a predictable result—hardware FIFO buffers operating as input or output queues.

270. **Reasonable Expectation of Success.** A POSITA would have had a reasonable expectation of success in combining Khawand and Rosenthal because the combination uses conventional and well-known techniques to yield a predictable result. Khawand already uses queues and teaches the use of flow control. Ex-1008, 3:52-61 (queues); 3:62-4:5 (flow control); Fig. 6. Rosenthal is directed at flow control with queues and registers. Ex-1009, Abstract; 2:15-17; 2:21-26; 16:33-57. By the time of the alleged invention, hardware FIFOs, flow control, registers, and queue management were routine and well-known in the art; it would have been a simple application of these known techniques from Rosenthal to Khawand because, as explained above, it would have been a straightforward application of known prior art elements according to known methods and would have yielded a predictable result.

271. In view of the conventional nature of these techniques, implementing the combination would not have required specialized skills beyond what a POSITA would have had at the time of the alleged invention. The use of registers, hardware FIFOs, queue-management, and flow-control teachings, such as those taught by

Rosenthal, would have been included in undergraduate education and would have been common in multiprocessor and computer architecture work. These are basic building blocks of digital design that have been known, used, and taught in schools for decades.

272. Furthermore, a POSITA would have been motivated and would have had a reasonable expectation of success in combining the references, as explained above, because the combination would not have changed the principle of operation for any of the teachings of the references relied upon in the combination. For example, Khawand's multiprocessor teachings would still have had multiple input queues for distributed task processing, as Khawand teaches. Rosenthal's hardware FIFOs, flow control mechanisms, and flow control register would still have been used to buffer data and manage queues, as Rosenthal teaches. Therefore a POSITA would have been motivated to combine Khawand and Rosenthal's teachings to use hardware FIFOs and track available queue space, with a reasonable expectation of success in doing so.

273. Based on the foregoing, Ground 3 teaches limitation [1e].

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

274. Ground 3 teaches limitation [1f], as explained below.

275. Ground 3 teaches computer storage storing instructions, which when executed by said at least one second processing element causes the execution of steps described in limitations [1g]-[1i]. 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. Therefore, Khawand teaches the Inter-Processor Manager executing software instructions to provide the functionality described by Khawand and further explained for limitations [1g]-[1j] below. This software was stored on computer storage such as memory. At the very least, it would have been obvious to store the software for Khawand’s processors (including the inter-processor manager) on computer storage because the software would need to be stored somewhere. That is how computers have operated since their inception.

276. Based on the foregoing, Ground 3 teaches limitation [1f].

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

277. Ground 3 teaches limitation [1g], as described below.

278. Ground 3 teaches a least one second processing element (e.g., Inter-Processor Manager) that accesses said reservation register and reads said stored value. Rosenthal teaches that processors access the flow control register to read 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.

279. The motivation to combine Khawand and Rosenthal was explained above. *Supra* §V.C.3[1e] (Motivation to Combine subsection). For the Khawand-Rosenthal combination, Khawand teaches that the inter-processor manager is responsible for evaluating the flow control criteria of the second processor’s queue before the first processor can delegate a service application to the second processor:

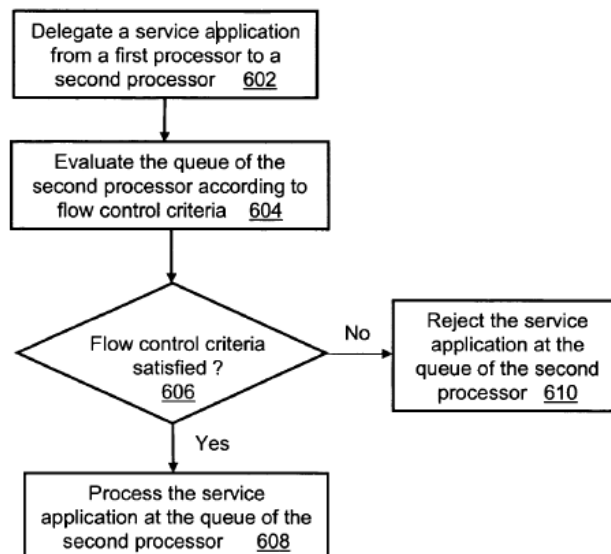


FIG. 6 600

Ex-1008, Fig. 6. Specifically, Khawand discloses that “[i]n step 604, the queue 404 of the second processor 306 is **evaluated according to flow control criteria**.

In a first embodiment, the **evaluation step can take place at the inter-processor manager 308.**” Ex-1008, 4:6-9. Khawand’s flow control criteria is based on whether the queue has sufficient capacity to hold the service application. Ex-1008, 4:15-27 (“... the flow control criteria can include a capacity threshold for each queue 404. ... the service application is processed if ... it is determined that the capacity threshold of the queue 404 is not exceeded (i.e., the flow control criteria has been satisfied).”). Khawand also discloses that the “inter-processor manager 308 can request an update from the queue 404 of each processor ...” Ex-1008, 5:50-54.

280. Rosenthal provides a mechanism for the inter-processor manager to evaluate flow control criteria, including queue capacity, as Khawand teaches. *Supra* §V.C.3[1e]. Therefore, in the Khawand-Rosenthal combination, the inter-processor manager accesses the flow control register, which stores the number of available entries in the input queue (Ex-1008, 4:6-27; Ex-1009, 16:35-37), to make a determination if there is enough space available in the queue. That is purpose for its flow control register:

The register 45 stores an indication of the number of available spaces for commands in the associated FIFO buffer 39. The use of this flow control register 45 allows a **requirement to be placed on an application program attempting an access of the input/output control unit 29 that it first determine whether space is available in**

the FIFO unit 31 before it writes any command. Before writing any data, the application program must determine the amount of “free” space to which it may write. This it does by having the **central processing unit read the value in the flow control register 45 associated with the FIFO buffer 39 being utilized.**

Ex-1009, 16:35-46.

281. Based on the foregoing, Ground 3 teaches limitation [1g].

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

282. Ground 3 teaches limitation [1h], as described below.

283. Ground 3 teaches at least one second processing element (e.g., Inter-Processor Manager) that determines when said read value indicates available space in said input queue (e.g., Processor 2’s input queue) for said first processing element (e.g., Processor 1) to issue a command. The inter-processor manager accesses the flow control register to read 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 explains, that is the purpose of the flow control register. Ex-1009, 16:35-49.

284. Khawand teaches evaluating flow control on basis of whether there is sufficient space in processor 2’s input queue:

Returning to step 604, the **flow control criteria** can include a **capacity threshold for each queue 404**. The capacity threshold indicates the capacity of the queue 404 for processing service applications. In one embodiment, the service application is processed in step 608 by the second processor 306 if at decision block 606 **it is determined that the capacity threshold of the queue 404 is not exceeded (i.e., the flow control criteria has been satisfied)**. On the other hand, the service application is rejected by the second processor 306 in step 610 upon determining at decision block 606 that the capacity threshold of the queue 404 has been exceeded. This embodiment reflects a fixed capacity threshold for managing the capacity of the queues 404.

Ex-1008, 4:15-27. This flow control evaluation is performed by the inter-processor manager. Ex-1008, 4:7-9 (“... the evaluation step can take place at the inter-processor manager.”). Therefore, in the Khawand-Rosenthal combination, the inter-processor manager determines when the value read from the flow control register indicates that 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 a service application to Processor 2. Ex-1008, 3:62-4:5.

285. The motivation to combine Khawand and Rosenthal was explained above. *Supra* §V.C.3[1e] (Motivation to Combine subsection). Based on the foregoing, Ground 3 teaches limitation [1h].

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

286. Ground 3 teaches limitation [1i], as explained below.

287. Ground 3 teaches at least one second processing element (e.g., inter-processor manager) that notifies said first processing element (e.g., Processor 1) to issue said command to said input queue (e.g., Processor 2's input queue). For example, Khawand teaches that when an input queue becomes full, a suspend command is sent from the processor to the inter-processor manager. Ex-1008, 4:60-63. The suspend command indicates that the associated queue is not available to process service applications. Ex-1008, 4:63-67.

288. When the input queue becomes available again for service applications, the inter-processor manager notifies the first processor of the availability through a resume command:

Reestablishing reception of service applications at the corresponding one of the group of processing elements mentioned above occurs upon satisfying the flow control criteria as described above. When this happens, the second processor 306 **submits a resume command to the inter-processor manager 308** indicating the corresponding one of the group of processing elements is available to process service applications, **which in turn synchronizes the first processor 304 according to the resume command**. In an alternative embodiment where the inter-processor manager 308 is not used, the suspend and

resume commands are transmitted from the second processor 306 to the first processor 304.

Ex-1008, 5:18-29; *see also* Ex-1008, 5:1-17 (“... suspend command corresponds to ... all tasks 402 and associated queues 404 would no longer be available to process service applications ...”). As Khawand explains, this algorithm “provides a stable environment for **managing queues 404 using suspend and resume commands[.]**” Ex-1008, 5:30-40. Khawand discloses this processing taking place in the inter-processor manager. Ex-1008, 5:40-41. Khawand teaches that the inter-processor manager sends queue capacity information to the first processing element, so that the first processor can issue the command. Ex-1008, 5:50-63. Therefore, Khawand teaches, or at least suggests, the inter-processor manager notifying 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.

289. Based on the foregoing, Ground 3 teaches limitation [1i].

[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.

290. Ground 3 teaches limitation [1j], as explained below.

291. Ground 3 teaches wherein said first processing element (e.g., Processor 1) receives notification from said at least one second processing element

(e.g., Inter-Processor Manager) regarding issuing the command, issues said command to said input queue (e.g., Processor 2's input queue). As Khawand explains, "the first and second processors 304-306 of the mobile device 106 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.

292. As explained above, *supra* §V.C.3[1i], when the input queue is full, the inter-processor manager sends a suspend command to the first processor, followed by a resume command when space becomes available in the input queue. When the first processing element (e.g., Processor 1) receives this notification (Ex-1008, 4:60-67; 5:18-41), it then issues its command to the input queue of Processor 2 to delegate a service application. Ex-1008, 3:65-4:22.

293. Ground 3 also teaches wherein said first processing element (e.g., Processor 1) receives a response corresponding to said command from said output queue. As Khawand explains, Khawand "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). Since the purpose of delegation is for another processor to process a service application, it would have been obvious to include an output queue to return processed data from that service application. *Supra* §V.C.3[1b]. In other words, a POSITA would have recognized the need to return results from processed service applications, and output queues had been used for that purpose

since at least the 1990s. *Supra* §V.C.3[1b]. Therefore, it would have been obvious to a POSITA for Processor 1 to receive a response corresponding to its delegated command from the output queue.

294. The motivation to combine Khawand and Rosenthal was explained above. *Supra* §V.C.3[1e] (Motivation to Combine subsection). Based on the foregoing, Ground 3 teaches limitation [1j].

4. Dependent Claim 2

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

295. Ground 3 teaches claim 2, as explained below.

296. As I have explained above, the combination of Khawand and Rosenthal teaches claim 1, *supra* §V.C.3. The motivation to combine Khawand and Rosenthal was explained above. *Supra* §V.C.3[1e] (Motivation to Combine subsection).

297. Rosenthal teaches wherein said input queue is a first-in-first-out (FIFO) buffer. Rosenthal discloses that FIFOs (first-in first-out) are used as write buffers. Ex-1009, 1:44-50. Rosenthal teaches using FIFO write buffers in the CPU, bridges, or I/O devices. Ex-1009, 1:50-53.

The architecture of the system 22 has been designed so that it eliminates almost all read operations of input/output devices by the central processing unit. In order to accomplish this, the input/output control

unit 29 includes a **first-in first-out (FIFO) unit 31** for storing write operations directed to the input/output control unit. The **FIFO unit 31 queues incoming write operations**. Unlike FIFO units in prior art systems, it stores both addresses and data. This allows the write operations to the input/output control unit 29 to occur asynchronously so that both the central processing unit 21 and the input/output control unit 29 may be functioning independently of one another and neither need wait for operations of the other.

Ex-1009, 3:66-4:11; *see also* Ex-1009, 4:61-64 (“The control unit 29 includes a hardware FIFO unit 31 for receiving incoming commands addressed to the input/output devices on a device bus 34.”).

298. Based on the foregoing, Ground 3 teaches Claim 2.

5. Dependent Claim 6

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.

299. Ground 3 teaches claim 6, as explained below.

300. As I have explained above, the combination of Khawand and Rosenthal teaches claim 1, *supra* §V.C.3. The motivation to combine Khawand and Rosenthal was explained above. *Supra* §V.C.3[1e] (Motivation to Combine subsection).

301. 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 discloses that “the **service application is processed in step 608 by the second processor 306** if at decision block 606 it is determined that the capacity threshold of the queue 404 is not exceeded (i.e., the flow control criteria has been satisfied).” Ex-1008, 4:18-22. In other words, when there is space in the second processor’s input queue, the service application is delegated to the second processor, which processes the command from its input queue.

302. On completion of the service application, the second processor obviously provides its results back to the first processor because the first processor delegated the service application to begin with. Ex-1008, 3:65-4:22. As explained above, it would have been obvious to use the output queue for return results.

Supra §§V.C.3[1b], [1j]. Accordingly, it would have been obvious for the second processor to provide the processed data/results from the delegated command to the output queue, and then to notify the output queue to provide the response back to the first processor. As Khawand explains, Processors 1 and 2 are connected by a bus; therefore, 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. Ex-1008, 3:37-51; Fig. 3. Additionally, it would have been obvious to a POSITA for the output queue to notify the first processor that a result was ready to be read, e.g., through a simple ready or queue not empty flag, because the first processor would naturally have read the output queue,

consistent with how a POSITA would have understood communication over a bus.

Ex-1008, 3:37-51; Fig. 3.

303. Based on the foregoing, Ground 3 teaches Claim 6.

6. Independent Claim 7

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

304. Ground 3 teaches limitation [7pre]. Limitation [7pre] is identical to limitation [1pre]. For the same reasons I have discussed for limitation [1pre], Ground 3 teaches limitation [7pre]. *Supra* §V.C.3V.C.3[1pre].

[7a] an input queue implemented in hardware;

305. Ground 3 teaches limitation [7a]. Limitation [7a] is identical to limitation [1a]. For the same reasons I have discussed for limitation [1a], Ground 3 teaches limitation [7a]. *Supra* §V.C.3V.C.3[1a].

[7b] an output queue implemented in hardware;
--

306. Ground 3 teaches limitation [7b]. Limitation [7b] is identical to limitation [1b]. For the same reasons I have discussed for limitation [1b], Ground 3 teaches limitation [7b]. *Supra* §V.C.3V.C.3[1b].

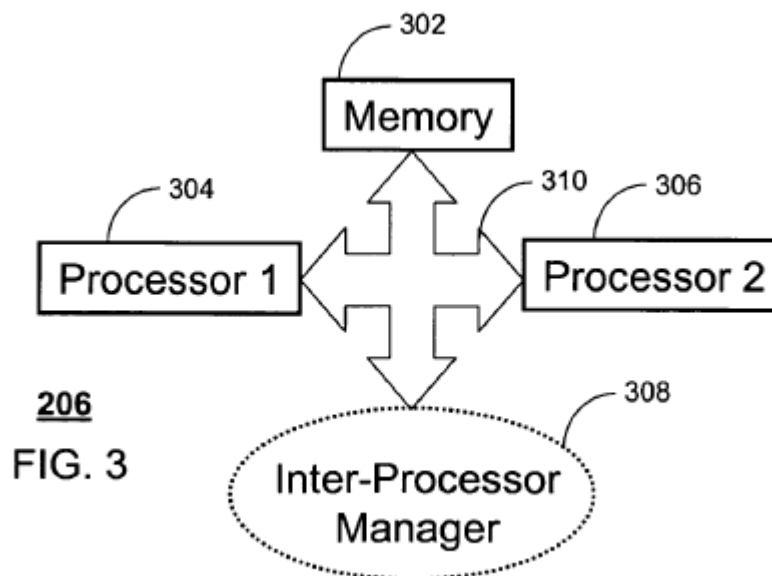
[7c] a first processing element having access to said input queue and said output queue;

307. Ground 3 teaches limitation [7c]. Limitation [7c] is identical to limitation [1c]. For the same reasons I have discussed for limitation [1c], Ground 3 teaches limitation [7c]. *Supra* §V.C.3V.C.3[1c].

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

308. Ground 3 teaches limitation [7d], as explained below.

309. Ground 3 teaches at least one second processing element (e.g., Khawand's inter-processor manager) in communication with said input queue (e.g., Processor 2's input queue). Khawand's Figure 3 discloses a multiprocessor system with multiple processors, an inter-processor manager, and memory.



Ex-1008, Fig. 3. Khawand explains that “[t]he processing elements 302-308 of the processing system 206 are interconnected by a conventional communication bus 310.” Ex-1008, 3:37-47. “The construction of the bus 310 can be physical or logical. ... Alternatively, a portion or whole of the communication bus 310 can represent a logical connection such as software components sharing data structures.” Ex-1008, 3:37-47.

310. Khawand’s processing elements, including the Inter-Processor Manager, use the communication bus to delegate service applications through the input queue for a given processor. Ex-1008, 3:65-4:22 (delegation); 3:37-51 (communication bus). As explained below, the Inter-Processor Manager delegates service applications to the second processor by writing to its input queue. *Infra* §V.C.6[7j]. Therefore, the Inter-Processor Manager is in communication with the input queue.

311. The input queue, implemented in hardware, was explained above. *Supra* §V.C.3[1a], §V.C.3[1e] (Motivation to Combine subsection).

312. Based on the foregoing, Ground 3 teaches limitation [7d].

[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;
-------------	--

313. Ground 3 teaches limitation [7e]. Limitation [7e] is identical to limitation [1e]. For the same reasons I have discussed for limitation [1e], Ground 3 teaches limitation [7e]. *Supra* §V.C.3[1e].

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

314. Ground 3 teaches limitation [7f]. Limitation [7f] is identical to limitation [1f]. For the same reasons I have discussed for limitation [1f], Ground 3 teaches limitation [7f]. *Supra* §V.C.3[1f].

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

315. Ground 3 teaches limitation [7g], as explained below.

316. Ground 3 teaches at least one second processing element (e.g., inter-processor manager) receives a command from said first processing element (e.g., Processor 1). Khawand teaches the 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 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. In step 606, if the flow control criteria is satisfied,

the inter-processor manager submits the command it received from the first processor (step 602 delegation) to the input queue of the second processor.

Ex-1008, Fig. 6 (step 602); 4:6-27. In order for the inter-processor manager to submit the command from the first processor as Khawand teaches, it must first receive the command from the first processor. Therefore, Khawand teaches or at least suggests that the inter-processor manager receives a command from the first processing element, when the steps of Figure 6 are centralized on the inter-processor manager.

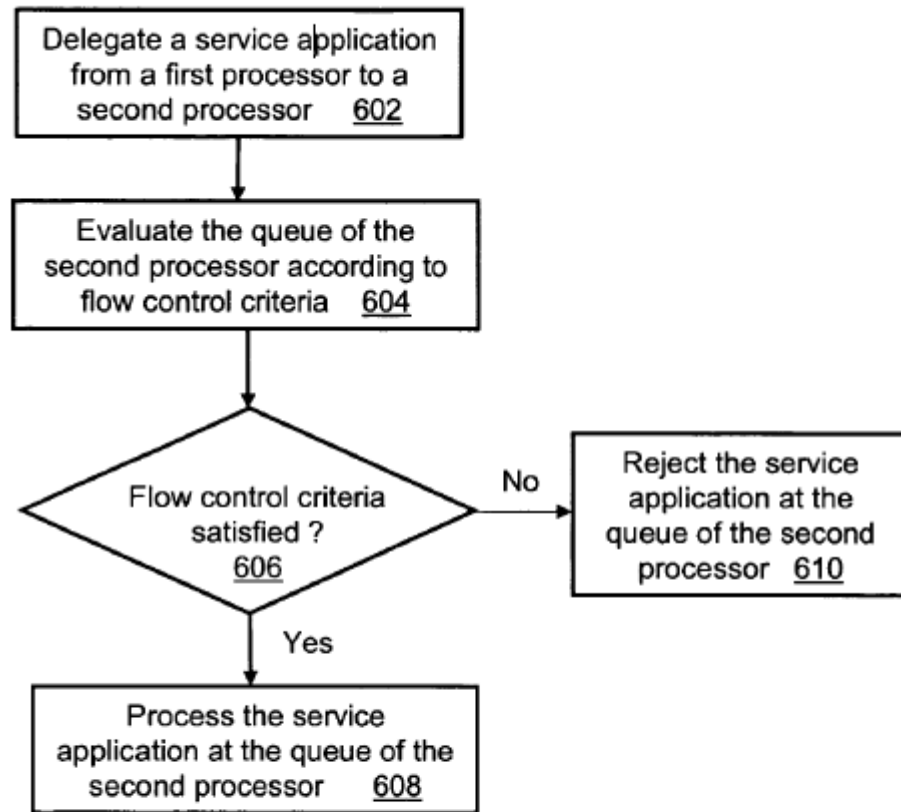


FIG. 6 600

Ex-1008, Fig. 6.

317. Based on the foregoing, Ground 3 teaches limitation [7g].

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

318. Ground 3 teaches limitation [7h]. Limitation [7h] is identical to limitation [1g]. For the same reasons I have discussed for limitation [1g], Ground 3 teaches limitation [7h]. *Supra* §V.C.3[1g].

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

319. Ground 3 teaches limitation [7i]. Limitation [7i] is nearly identical to limitation [1h], except that limitation [7i] recites “said command” while limitation [1h] recites “a command.” For the same reasons I have discussed for limitation [1h], Ground 3 teaches limitation [7i]. *Supra* §V.C.3[1h].

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

320. Ground 3 teaches limitation [7j], as explained below.

321. Ground 3 teaches at least one second processing element (e.g., inter-processor manager) that issues said command to said input queue (e.g., Processor 2’s input queue) on behalf of said first processing element (e.g., Processor 1). Khawand teaches the 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 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. In step 606, if the flow control criteria is satisfied, the inter-processor manager submits the command it received from the first processor (step 602 delegation) to the input queue of the second processor. Ex-1008, Fig. 6; 4:6-27. Therefore, when the steps of Figure 6 are centralized on the inter-processor manager, the inter-processor manager issues the command to

the input queue of the second processor on behalf of the first processor. Ex-1008, 3:65-4:2.

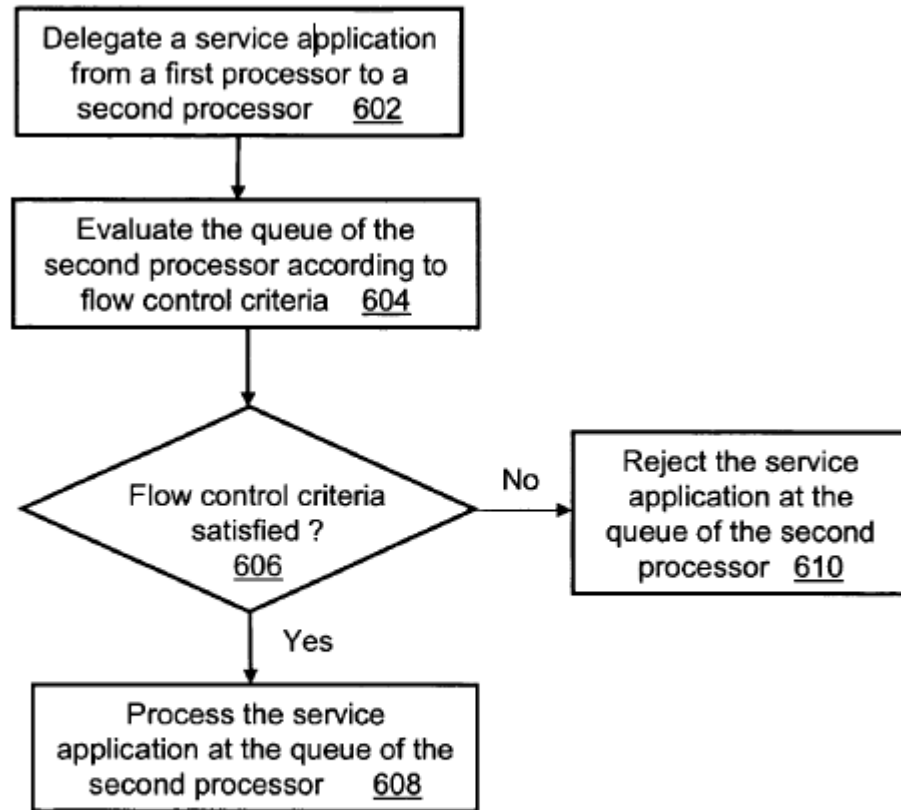


FIG. 6 600

Ex-1008, Fig. 6.

322. Based on the foregoing, Ground 3 teaches limitation [7j].

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

323. Ground 3 teaches limitation [7k]. Limitation [7k] is nearly identical to limitation [1j], except that limitation [7k] recites only the final clause of “receives a response corresponding to said command from said output queue” while limitation [1j] includes additional limitations on the first processing element that are not present in limitation [7k]. For the same reasons I have discussed for limitation [1j], Ground 3 teaches limitation [7k]. *Supra* §V.C.3[1j].

7. Dependent Claim 8

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

324. Ground 3 teaches Claim 8. Claim 8 is nearly identical to Claim 2, except that Claim 8 recites “system of claim 7” while Claim 2 recites “system of claim 1.” As I have explained above, the combination of Khawand and Rosenthal teaches claim 7 *supra* §V.C.6. For the same reasons I have discussed for Claim 2, Ground 3 teaches Claim 8. *Supra* §V.C.4.

8. Dependent Claim 12

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.

325. Ground 3 teaches Claim 12. Claim 12 is nearly identical to Claim 6, except that Claim 12 recites “system of claim 7” while Claim 6 recites “system of

claim 1.” As I have explained above, the combination of Khawand and Rosenthal teaches claim 7 *supra* §V.C.6. For the same reasons I have discussed for Claim 6, Ground 3 teaches Claim 12. *Supra* §V.C.5.

9. Independent Claim 13

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

326. Ground 3 teaches limitation [13pre]. Limitation [13pre] is identical to limitation [1pre]. For the same reasons I have discussed for limitation [1pre], Ground 3 teaches limitation [13pre]. *Supra* §V.C.3[1pre].

[13a] an input queue implemented in hardware;

327. Ground 3 teaches limitation [13a]. Limitation [13a] is identical to limitation [1a]. For the same reasons I have discussed for limitation [1a], Ground 3 teaches limitation [13a]. *Supra* §V.C.3[1a].

[13b] an output queue implemented in hardware;

328. Ground 3 teaches limitation [13b]. Limitation [13b] is identical to limitation [1b]. For the same reasons I have discussed for limitation [1b], Ground 3 teaches limitation [13b]. *Supra* §V.C.3[1b].

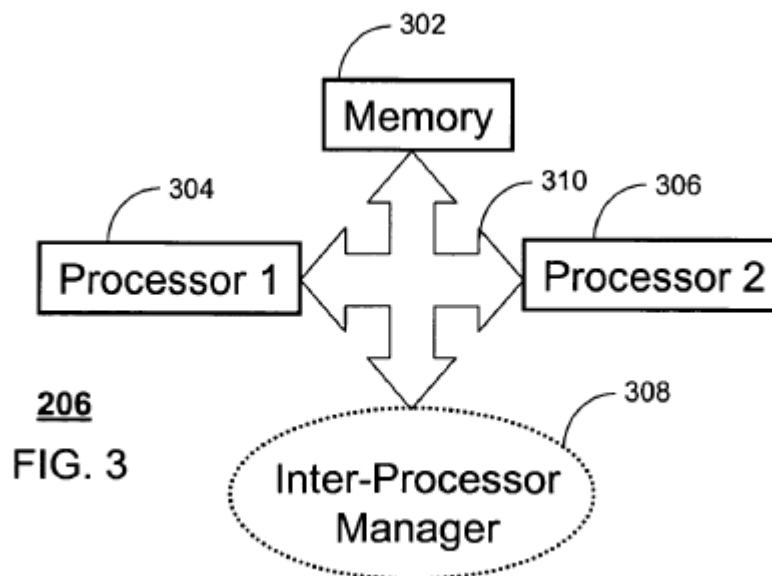
[13c] a first processing element having access to said input queue and said output queue;

329. Ground 3 teaches limitation [13c]. Limitation [13c] is identical to limitation [1c]. For the same reasons I have discussed for limitation [1c], Ground 3 teaches limitation [13c]. *Supra* §V.C.3[1c].

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

330. Ground 3 teaches limitation [13d], as explained below.

331. Ground 3 teaches at least one second processing element (e.g., Khawand's inter-processor manager) in communication with said input queue and said output queue. Khawand's Figure 3 discloses a multiprocessor system with multiple processors, an inter-processor manager, and memory.



Ex-1008, Fig. 3. “The processing elements 302-308 of the processing system 206 are interconnected by a conventional communication bus 310. The construction of the bus 310 can be physical or logical. ... Alternatively, a portion or whole of the communication bus 310 can represent a logical connection such as software components sharing data structures.” Ex-1008, 3:37-47.

332. Khawand’s processing elements, including the Inter-Processor Manager, use the communication bus to delegate service applications through the input queue for a given processor, e.g., the input queue for Processor 2. Ex-1008, 3:65-4:22 (delegation); 3:37-51 (communication bus). As explained above, the Inter-Processor Manager delegates service applications to the second processor by writing to its input queue. *Supra* §V.C.6[7j]. The Inter-Processor Manager communicates with the input queue using a communication bus. Ex-1008, 3:65-4:22 (delegation), 3:37-51 (communication bus), Fig. 3. Therefore, the Inter-Processor Manager is in communication with the input queue. The input queue, implemented in hardware, was explained above. *Supra* §V.C.3[1a], §V.C.3[1e] (Motivation to Combine subsection).

333. Khawand’s inter-processor manager is also in communication with the output queue. As discussed above, Khawand teaches or at least suggests an output queue. V.C.3*Supra* §V.C.9V.C.3[13b], §V.C.3[1b], §V.C.3[1j]. Khawand teaches the inter-processor manager having 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 (“This processing can also take place in the inter-processor manager 308 monitoring the plurality of processors.”). Therefore, in view of its centralized role in service application delegation, the inter-processor manager would obviously have been in communication with whatever output queues are used to return processed data from those delegated service applications, including the output queue for Processor 2. This is further explained below. *Infra* §V.C.9[13k].

334. Based on the foregoing, Ground 3 teaches limitation [13d].

[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;

335. Ground 3 teaches limitation [13e]. Limitation [13e] is identical to limitation [1e]. For the same reasons I have discussed for limitation [1e], Ground 3 teaches limitation [13e]. *Supra* §V.C.3[1e].

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

336. Ground 3 teaches limitation [13f]. Limitation [13f] is identical to limitation [1f]. For the same reasons I have discussed for limitation [1f], Ground 3 teaches limitation [13f]. *Supra* §V.C.3[1f].

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

337. Ground 3 teaches limitation [13g]. Limitation [13g] is identical to limitation [7g]. For the same reasons I have discussed for limitation [7g], Ground 3 teaches limitation [13g]. *Supra* §V.C.6[7g].

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

338. Ground 3 teaches limitation [13h]. Limitation [13h] is identical to limitation [1g]. For the same reasons I have discussed for limitation [1g], Ground 3 teaches limitation [13h]. *Supra* §V.C.3[1g].

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

339. Ground 3 teaches limitation [13i]. Limitation [13i] is nearly identical to limitation [1h], except that limitation [13i] recites “said command” while limitation [1h] recites “a command.” For the same reasons I have discussed for limitation [1h], Ground 3 teaches limitation [13i]. *Supra* §V.C.3[1h].

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

340. Ground 3 teaches limitation [13j]. Limitation [13j] is identical to limitation [7j]. For the same reasons I have discussed for limitation [7j], Ground 3 teaches limitation [13j]. *Supra* §V.C.6[7j].

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

341. Ground 3 teaches limitation [13k], as explained below.

342. Ground 3 teaches at least one second processing element (e.g., Khawand's inter-processor manager) receives a response corresponding to said command from said output queue. As explained above, Khawand teaches delegation of service applications, and it would have been obvious to a POSITA to include an output queue to buffer return data from those delegated service applications. *Supra* §V.C.3[1b]. Therefore, it would have been obvious for the 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 ("This processing can also take place in the inter-processor manager 308 monitoring the plurality of processors."). Additionally, Khawand teaches that the inter-processor manager can directly request an update from each queue, which teaches or at least suggests that the inter-processor manager would also be able to access the corresponding output queues and receive a response corresponding to the issued command. Ex-1008, 5:50-54 ("the inter-processor manager 308 can request an update from the

queue 404 of each processor, and submit an aggregate update to the processors representing the capacity of each of the queues of the plurality of processors.”).

343. Based on the foregoing, Ground 3 teaches limitation [13k].

10. Dependent Claim 14

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

344. Ground 3 teaches Claim 14. Claim 14 is nearly identical to Claim 2, except that Claim 14 recites “system of claim 13” while Claim 2 recites “system of claim 1.” As I have explained above, the combination of Khawand and Rosenthal teaches claim 13, *supra* §V.C.9. For the same reasons I have discussed for Claim 2, Ground 3 teaches Claim 14. *Supra* §V.C.4.

11. Dependent Claim 18

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.

345. Ground 3 teaches Claim 18. Claim 18 is nearly identical to Claim 6, except that Claim 18 recites “system of claim 13” while Claim 6 recites “system of claim 1.” As I have explained above, the combination of Khawand and Rosenthal teaches claim 13, *supra* §V.C.9. For the same reasons I have discussed for Claim 6, Ground 3 teaches Claim 18. *Supra* §V.C.5.

D. Ground 4: Claims 1-2, 6-8, 12-14, and 18 are Rendered Obvious by Khawand, Rosenthal, and Ohkawa

346. To the extent Patent Owner argues that hardware output queues were somehow not obvious by 2013, Ohkawa teaches hardware output queues.

Ex-1010, 4:15-17. Ground 4 adds Ohkawa's output queue teachings to Ground 3 (the combination of Khawand and Rosenthal, explained above). It is my opinion that a POSITA would have found the Challenged Claims obvious based on Ground 4.

1. Ohkawa (Ex-1010)

347. Ohkawa is U.S. Patent No. 8,028,284 to Ohkawa, entitled "Data Processing System and Program for Transferring Data." Ohkawa was filed on October 31, 2006, claiming priority to Japanese Patent Application 2005-321572 filed on November 4, 2005. Ohkawa issued on September 27, 2011. I understand that Ohkawa is prior art to the '596 patent under at least AIA §102(a)(1)-(2).

348. I have reviewed the Ohkawa reference. I understand that the Ohkawa reference was not cited nor considered during the prosecution of the '596 patent based primarily on the fact that Ohkawa is not cited on the face of the '596 patent and was not discussed in the prosecution history of the '596 patent.

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

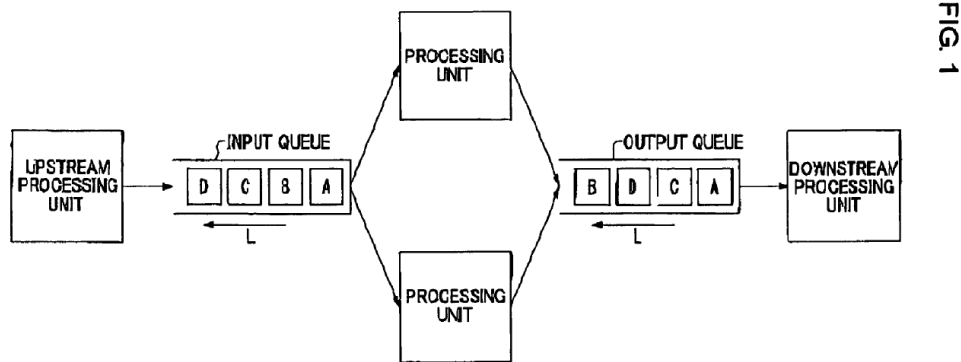


FIG. 1

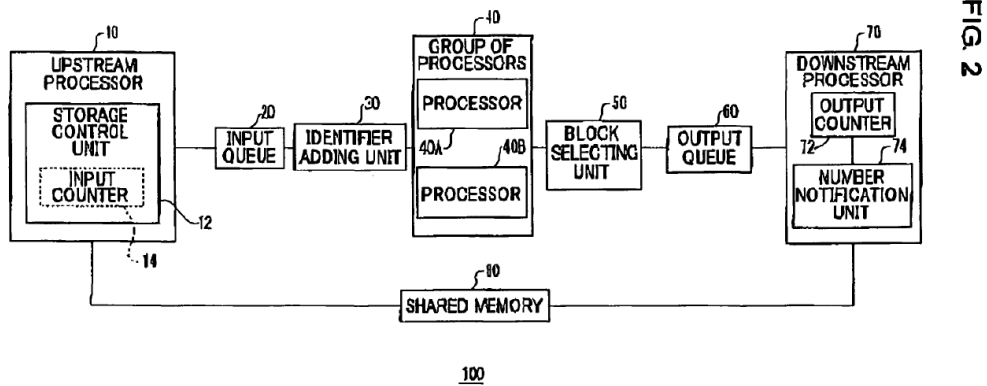


FIG. 2

Ex-1010, Figs 1 & 2. Specifically, Ohkawa teaches that:

The group of processors 40 includes a plurality of processors, or two processors (processors 40A and 40B) in this case. These processors read data from the input queue 20, process the same, and write the processed data to the output queue 60.

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, 5:7-13.

350. Ohkawa is analogous art in the same field as the '596 patent (computer architecture, including the use of queues and multiple processing elements, *supra* §III) because it teaches a multiprocessor architecture with coordinated processing and queue structures. Ex-1010, 1:30-40.

2. Independent Claim 1

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

351. Ground 4 teaches limitation [1pre]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1pre], Ground 4 teaches limitation [1pre]. *Supra* §V.C.3[1pre].

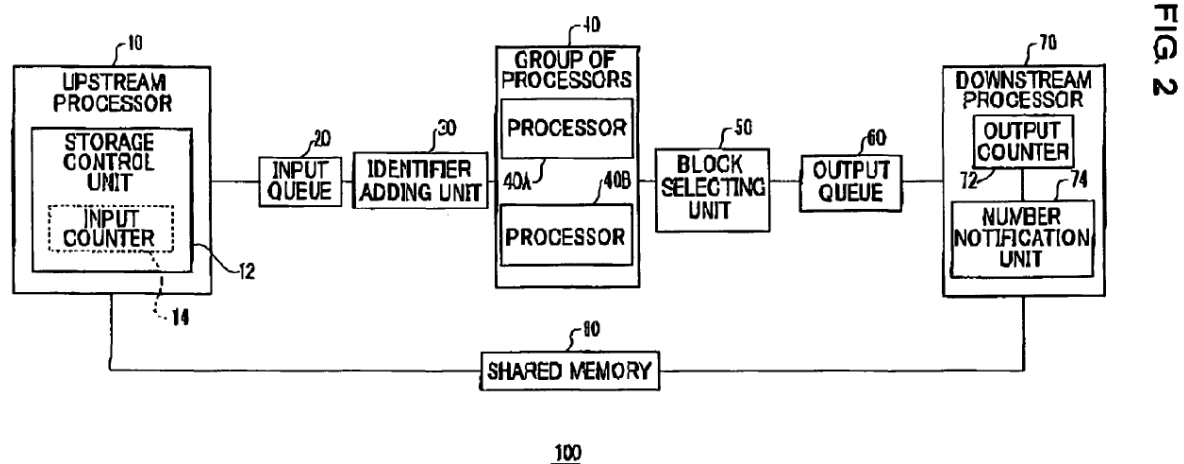
[1a] an input queue implemented in hardware;

352. Ground 4 teaches limitation [1a]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1a], Ground 4 teaches limitation [1a]. *Supra* §V.C.3[1a].

[1b] an output queue implemented in hardware;

353. Ground 4 teaches limitation [1b], as explained below.

354. Ohkawa teaches an output queue implemented in hardware. For example, Ohkawa teaches an “output queue” for buffering data passed between processors in a multiprocessor system:



Ex-1010, Fig. 2. Ohkawa describes the purpose of the output queue: “[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.

355. Ohkawa also teaches implementing the output queue in “hardware.” Ex-1010, 4:52-61 (“In FIG. 2 and the like, the individual elements shown as functional blocks for performing various types of processing can be constituted, in terms of **hardware**, by CPUs, memories, and other LSIs. In terms of software, they can be achieved by a program and the like that are loaded on a memory and have a data transfer function. It will thus be understood by those skilled in the art that these **functional blocks may be achieved by various forms, including**

hardware alone, software alone, and combinations of these, and therefore not be limited to any one of them.”). The output queue is implemented in physical circuits, without software intervention, because Ohkawa teaches a hardware implementation for its output queue. Ex-1010, 4:52-61.

356. **Motivation to Combine.** The motivation to combine Khawand with Rosenthal was explained above. *Supra* §V.C.3[1e] (Motivation to Combine subsection).

357. Given Khawand’s delegation of service applications from one processor to another, it was obvious to use an output queue for return data from completed service applications, as explained above. *Supra* §V.C.3[1b]. Therefore, Khawand’s architecture suggests the use of output queues to pass data between processors, and Ohkawa teaches output queues for this purpose. Ex-1010, 5:11-13; Fig. 2. In view of Ohkawa’s teachings, it would have been obvious to a POSITA, who would have been motivated to use an output queue for returning processed data from delegated service applications in Khawand’s multiprocessor architecture.

358. Moreover, Khawand teaches a multiprocessor architecture, and Ohkawa teaches output queues for multiprocessor systems. Ex-1010, Abstract; 1:37-40; 5:11-13. Ohkawa describes its field as “technology for transferring data to a **group of processors** or transferring data from the **group of processors** in a system where the individual processors included in the **group of processors**

perform coordinated processing.” Ex-1010, 1:6-11 (emphasis added). Ohkawa explains that:

[a] multiprocessor system, or a system that implements a plurality of processors, can perform processing in parallel or in a coordinated fashion to achieve speedup of the entire processing. The coordinated processing by a plurality of processors is applied to part of the processing of the system. Such a system requires a process of transferring data generated by, for example, a processor that lies upstream of a plurality of processors in charge of the coordinated processing (hereinafter, referred to as a group of processors), to this group of processors, and a process of transferring data from this group of processors to a processor that lies downstream of this group of processors. The processing efficiency of the system depends on how these transfers are devised.

Ex-1010, 1:13-26. Therefore, 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, as taught by Ohkawa. Moreover, this would have been the application of a known technique (output queues) to improve a similar device (Khawand’s multiprocessor architecture) in the same way.

359. Ohkawa teaches that output queues are “interposed between the plurality of processor units and the downstream processing unit.” Ex-1010, 4:15-17. Then, the “processing units process data transferred thereto, and transfer

the processed data to the output queue.” Ex-1010, 4:31-33. Ohkawa describes the purpose of the output queues:

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. 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].

360. Additionally, Khawand and Rosenthal provide motivation by teaching the use of 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).

361. Hardware output queues, such as those taught by Ohkawa, were widely used building blocks in computer architecture, and a POSITA would have been motivated to apply these ubiquitous hardware building blocks because they were known to provide efficient buffering for data transfer between processors. Indeed, using a mix of fast hardware queues would have enabled the processors to exploit temporal locality in determining which queues for each task should be in hardware, while still allowing software flexibility to have additional overflow

queues in memory, much like how cache memory systems work. Khawand and Ohkawa both suggest this type of hybrid approach. Ex-1008, 6:14-17; Ex-1010, 4:52-61.

362. **Reasonable Expectation of Success.** The explanation for combining Khawand and Rosenthal was explained above. *Supra* §V.C.3[1e] (Motivation to Combine subsection). A POSITA would have had a reasonable expectation of success in further applying Ohkawa's teachings because the combination used conventional and well-known techniques (output queues) to yield a predictable result (buffers between processors).

363. Additionally, Ohkawa is directed at transferring data in a multiprocessor system. Given the similarities with Khawand's multiprocessor architecture, and since output queues had been well-known for many decades, a POSITA would have expected success in combining the teachings of the three references. Implementing the combination would not have required specialized skills beyond what a POSITA would have had at the time of the alleged invention, since output queues were elementary building blocks taught in undergraduate logic design courses.

364. Furthermore, a POSITA would have been motivated and would have had a reasonable expectation of success in combining the references because the combination would not have changed the principle of operation for any of the

references. 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.

365. Based on the foregoing, Ground 4 teaches limitation [1b].

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

366. Ground 4 teaches limitation [1c], as explained below.

367. Ohkawa teaches an output queue. *Supra* §V.D.2[1b]. Khawand and Rosenthal teach 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 to Combine Subsection).

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

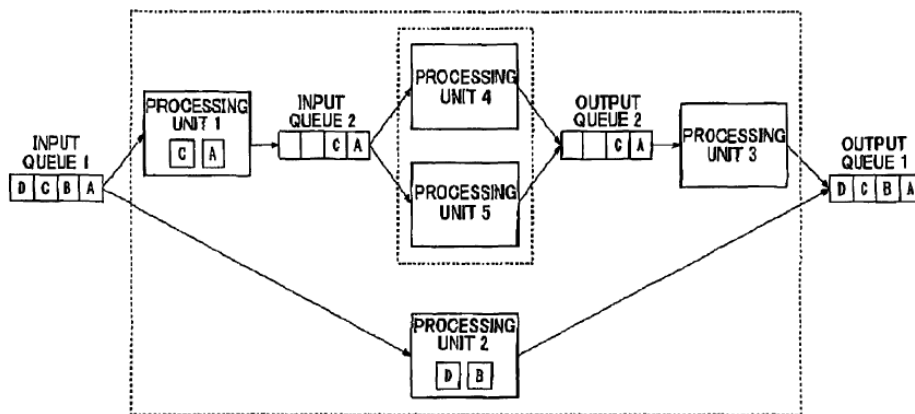


FIG 14

Ex-1010, Fig. 14. For example, in Figure 14, processing unit 1 produces an output that is placed into a queue. From processing unit 1's perspective the downstream queue is an output queue, however from processing unit 4's perspective the queue is an input queue. Ohkawa explains that its 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. Accordingly, it would have been obvious to use Ohkawa's queue structure where each of Khawand's processing elements accesses both an input queue and an output queue because, like Ohkawa, Khawand includes multiple interconnected processing engines that pass data among them. Ex-1008, Fig. 3.

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

369. Ground 4 teaches limitation [1d]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1d], Ground 4 teaches limitation [1d].

Supra §V.C.3[1d].

[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;

370. Ground 4 teaches limitation [1e]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1e], Ground 4 teaches limitation [1e].

Supra §V.C.3[1e].

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

371. Ground 4 teaches limitation [1f]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1f], Ground 4 teaches limitation [1f].

Supra §V.C.3[1f].

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

372. Ground 4 teaches limitation [1g]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1g], Ground 4 teaches limitation [1g].

Supra §V.C.3[1g].

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

373. Ground 4 teaches limitation [1h]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1h], Ground 4 teaches limitation [1h].

Supra §V.C.3[1h].

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

374. Ground 4 teaches limitation [1i]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1i], Ground 4 teaches limitation [1i].

Supra §V.C.3[1i].

[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.

375. Ground 4 teaches limitation [1j]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [1j], Ground 4 teaches limitation [1j]. *Supra* §V.C.3[1j].

376. Ground 4 includes additional teachings from Ohkawa regarding the output queue, as explained above. *Supra* §§V.D.2[1b]-[1c]. Khawand and Rosenthal teach the first processing element (e.g., Processor 1) receives a response corresponding to the delegated command from the output queue. *Supra* §V.C.3[1j]. The motivation to combine was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection).

377. Ohkawa further 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 would have been 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* Ex-1010, 5:11-13.

378. Based on the foregoing, Ground 4 teaches limitation [1j].

3. Dependent Claim 2

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

379. Ground 4 teaches claim 2. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 claim 2, Ground 4 teaches claim 2. *Supra* §V.C.4.

380. Ground 4 includes additional teachings from Ohkawa that explicitly teaches "data stored in the output queue is read in the order stored." Ex-1010, 4:41-43. The motivation to combine Khawand, Rosenthal, and Ohkawa was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection).

4. Dependent Claim 6

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.

381. Ground 4 teaches claim 6. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3, Ground 4 teaches claim 6. *Supra* §V.C.5 (claim 6).

382. Ground 4 includes additional teachings from Ohkawa regarding the output queue, as explained above. *Supra* §§V.D.2[1b]-[1c]. Khawand and Rosenthal teach at least one 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. The motivation to combine was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection).

383. 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 would have been 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.

5. Independent Claim 7

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

384. Ground 4 teaches limitation [7pre]. Limitation [7pre] is identical to limitation [1pre]. For the same reasons I have discussed for limitation [1pre], Ground 4 teaches limitation [7pre]. *Supra* §V.D.2[1pre].

[7a] an input queue implemented in hardware;

385. Ground 4 teaches limitation [7a]. Limitation [7a] is identical to limitation [1a]. For the same reasons I have discussed for limitation [1a], Ground 4 teaches limitation [7a]. *Supra* §V.D.2[1a].

[7b] an output queue implemented in hardware;

386. Ground 4 teaches limitation [7b]. Limitation [7b] is identical to limitation [1b]. For the same reasons I have discussed for limitation [1b], Ground 4 teaches limitation [7b]. *Supra* §V.D.2[1b].

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

387. Ground 4 teaches limitation [7c]. Limitation [7c] is identical to limitation [1c]. For the same reasons I have discussed for limitation [1c], Ground 4 teaches limitation [7c]. *Supra* §V.D.2[1c].

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

388. Ground 4 teaches limitation [7d]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [7d], Ground 4 teaches limitation [7d]. *Supra* §V.C.6[7d]. The motivation to combine Khawand, Rosenthal, and Ohkawa was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection).

[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;

389. Ground 4 teaches limitation [7e]. Limitation [7e] is identical to limitation [1e]. For the same reasons I have discussed for limitation [1e], Ground 4 teaches limitation [7e]. *Supra* §V.D.2[1e].

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

390. Ground 4 teaches limitation [7f]. Limitation [7f] is identical to limitation [1f]. For the same reasons I have discussed for limitation [1f], Ground 4 teaches limitation [7f]. *Supra* §V.D.2[1f].

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

391. Ground 4 teaches limitation [7g]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [7g], Ground 4 teaches limitation [7g]. *Supra* §V.C.6[7g]. The motivation to combine Khawand, Rosenthal, and Ohkawa was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection).

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

392. Ground 4 teaches limitation [7h]. Limitation [7h] is identical to limitation [1g]. For the same reasons I have discussed for limitation [1g], Ground 4 teaches limitation [7h]. *Supra* §V.D.2[1g].

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

393. Ground 4 teaches limitation [7i]. Limitation [7i] is nearly identical to limitation [1h], except that limitation [7i] recites “said command” while limitation [1h] recites “a command.” For the same reasons I have discussed for limitation [1h], Ground 4 teaches limitation [7i]. *Supra* §V.D.2[1h].

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

394. Ground 4 teaches limitation [7j]. Ground 4 relies on the disclosures of Ground 3’s combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [7j], Ground 4 teaches limitation [7j]. *Supra* §V.C.6[7j]. The motivation to combine Khawand, Rosenthal, and Ohkawa was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection).

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

395. Ground 4 teaches limitation [7k]. Limitation [7k] is nearly identical to limitation [1j], except that limitation [7k] recites only the final clause of

“receives a response corresponding to said command from said output queue” while limitation [1j] includes additional limitations on the first processing element that are not present in limitation [7k]. For the same reasons I have discussed for limitation [1j], Ground 4 teaches limitation [7k]. *Supra* §V.D.2[1j].

6. Dependent Claim 8

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

396. Ground 4 teaches Claim 8. Claim 8 is nearly identical to Claim 2, except that Claim 8 recites “system of claim 7” while Claim 2 recites “system of claim 1.” As I have explained above, the combination of Khawand, Rosenthal, and Ohkawa teaches claim 7 *supra* §V.D.5. For the same reasons I have discussed for Claim 2, Ground 4 teaches Claim 8. *Supra* §V.D.3.

7. Dependent Claim 12

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.
--

397. Ground 4 teaches Claim 12. Claim 12 is nearly identical to Claim 6, except that Claim 12 recites “system of claim 7” while Claim 6 recites “system of claim 1.” As I have explained above, the combination of Khawand, Rosenthal, and Ohkawa teaches claim 7 *supra* §V.D.5. For the same reasons I have discussed for Claim 6, Ground 4 teaches Claim 12. *Supra* §V.D.4.

8. Independent Claim 13

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

398. Ground 4 teaches limitation [13pre]. Limitation [13pre] is identical to limitation [1pre]. For the same reasons I have discussed for limitation [1pre], Ground 4 teaches limitation [13pre]. *Supra* §V.D.2[1pre].

[13a] an input queue implemented in hardware;

399. Ground 4 teaches limitation [13a]. Limitation [13a] is identical to limitation [1a]. For the same reasons I have discussed for limitation [1a], Ground 4 teaches limitation [13a]. *Supra* §V.D.2[1a].

[13b] an output queue implemented in hardware;

400. Ground 4 teaches limitation [13b]. Limitation [13b] is identical to limitation [1b]. For the same reasons I have discussed for limitation [1b], Ground 4 teaches limitation [13b]. *Supra* §V.D.2[1b].

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

401. Ground 4 teaches limitation [13c]. Limitation [13c] is identical to limitation [1c]. For the same reasons I have discussed for limitation [1c], Ground 4 teaches limitation [13c]. *Supra* §V.D.2[1c].

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

402. Ground 4 teaches limitation [13d]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [13d], Ground 4 teaches limitation [13d]. *Supra* §V.C.9[13d]. The motivation to combine Khawand, Rosenthal, and Ohkawa was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection).

403. Ground 4 includes additional teachings from Ohkawa that further teaches at least one second processing element in communication with said input queue and said output queue. For example, Ohkawa teaches a general multiprocessor architecture, where each processing unit is in communication with both an input queue (to the processor) and output queue (from the processor):

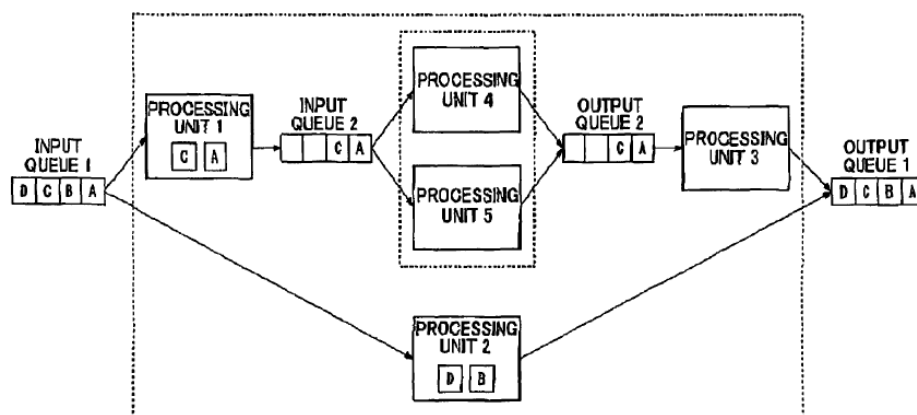


FIG 14

Ex-1010, Fig. 14. For example, in Figure 14, processing unit 4 produces an output that is placed into an output queue showing that processing unit 4 is in communication with an output queue. From processing units 4's perspective the downstream queue is an output queue, however from processing unit 3's perspective the queue is an input queue. Ohkawa explains that its queue system "may also be applied to a system where a plurality of processors perform coordinate processing using a hierarchical configuration." Ex-1010, 9:64-66. Each of Ohkawa's processing units communicates with both an input queue and an output queue. Accordingly, it would have been obvious to use Ohkawa's queue structure where each of Khawand's processing elements accesses both an input queue and an output queue because, like Ohkawa, Khawand includes multiple interconnected processing engines that pass data among them. Ex-1008, Fig. 3.

404. Based on the foregoing, Ground 4 teaches limitation [13d].

[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;

405. Ground 4 teaches limitation [13e]. Limitation [13e] is identical to limitation [1e]. For the same reasons I have discussed for limitation [1e], Ground 4 teaches limitation [13e]. *Supra* §V.D.2[1e].

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

406. Ground 4 teaches limitation [13f]. Limitation [13f] is identical to limitation [1f]. For the same reasons I have discussed for limitation [1f], Ground 4 teaches limitation [13f]. *Supra* §V.D.2[1f].

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

407. Ground 4 teaches limitation [13g]. Limitation [13g] is identical to limitation [7g]. For the same reasons I have discussed for limitation [7g], Ground 4 teaches limitation [13g]. *Supra* §V.D.5[7g].

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

408. Ground 4 teaches limitation [13h]. Limitation [13h] is identical to limitation [1g]. For the same reasons I have discussed for limitation [1g], Ground 4 teaches limitation [13h]. *Supra* §V.D.2[1g].

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

409. Ground 4 teaches limitation [13i]. Limitation [13i] is nearly identical to limitation [1h], except that limitation [13i] recites “said command” while limitation [1h] recites “a command.” For the same reasons I have discussed for limitation [1h], Ground 4 teaches limitation [13i]. *Supra* §V.D.2[1h].

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

410. Ground 4 teaches limitation [13j]. Limitation [13j] is identical to limitation [7j]. For the same reasons I have discussed for limitation [7j], Ground 4 teaches limitation [13j]. *Supra* §V.D.5[7j].

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

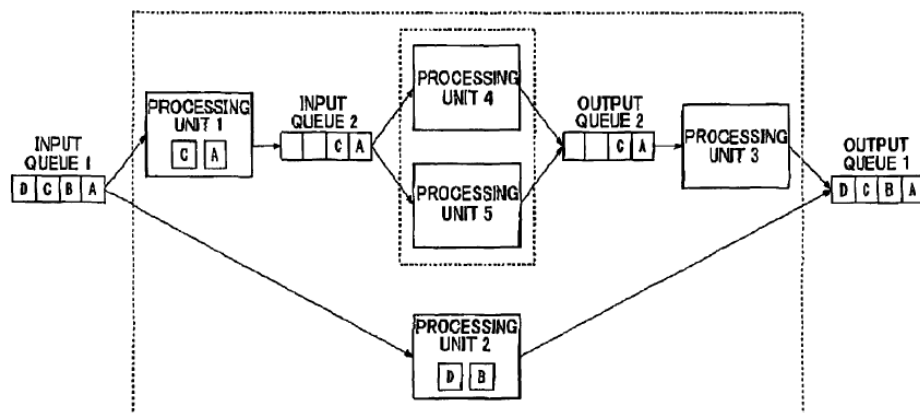
411. Ground 4 teaches limitation [13k]. Ground 4 relies on the disclosures of Ground 3's combination of Khawand and Rosenthal, and for the same reasons I have discussed for Ground 3 limitation [13k], Ground 4 teaches limitation [13k]. *Supra* §V.C.9[13k].

412. Ground 4 includes additional teachings from Ohkawa regarding the output queue, as explained above. *Supra* §§V.D.2[1b]-[1c]. Regarding claim 1, Khawand and Rosenthal teach the first processing element (e.g., Processor 1) receives a response corresponding to the delegated command from the output queue. *Supra* §V.C.3[1j]. The motivation to combine was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection).

413. 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 would have been 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].



414. Based on the foregoing, Ground 4 teaches limitation [13k].

9. Dependent Claim 14

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

415. Ground 4 teaches Claim 14. Claim 14 is nearly identical to Claim 2, except that Claim 14 recites “system of claim 13” while Claim 2 recites “system of claim 1.” As I have explained above, the combination of Khawand, Rosenthal, and Ohkawa teaches claim 13, *supra* §V.D.8. For the same reasons I have discussed for Claim 2, Ground 4 teaches Claim 14. *Supra* §V.D.3.

10. Dependent Claim 18

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.

416. Ground 4 teaches Claim 18. Claim 18 is nearly identical to Claim 6, except that Claim 18 recites “system of claim 13” while Claim 6 recites “system of claim 1.” As I have explained above, the combination of Khawand, Rosenthal, and Ohkawa teaches claim 13 *supra* §V.D.8. For the same reasons I have discussed for Claim 6, Ground 4 teaches Claim 18. *Supra* §V.D.4.

E. Ground 5: Claims 3-4, 9-10, and 15-16 are Rendered Obvious in view of Khawand, Rosenthal, Ohkawa, and Matsunami

417. Certain dependent claims recite SAN/NAS limitations. Matsunami teaches SAN and NAS. Ground 5 adds Matsunami’s teachings for these claims.

1. Dependent Claims 3-4

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.

418. Ground 4 teaches claim 1. *Supra* §V.D.2. 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. Therefore, Ground 5 teaches the multiprocessor system of Ground 4 as part of a SAN/NAS controller.

419. For example, Matsunami teaches processors on the SAN/NAS channel interface controllers may be implemented as 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

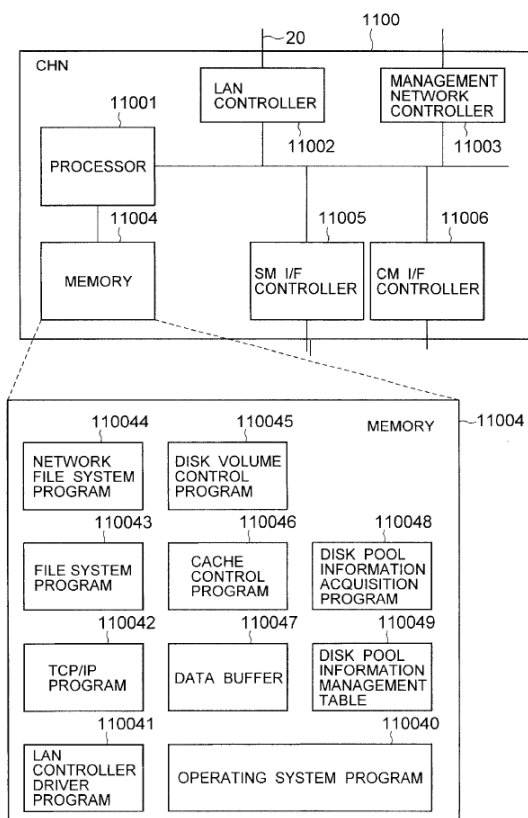
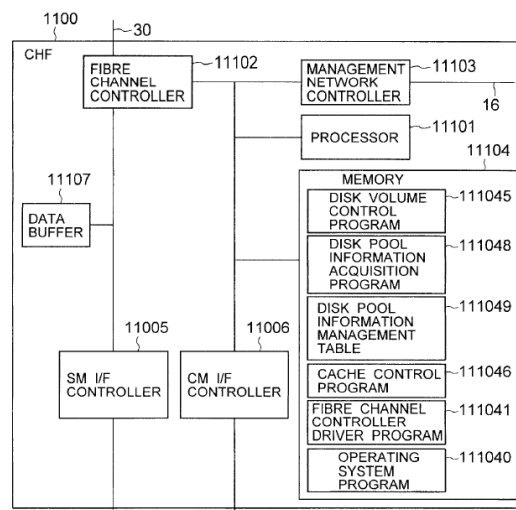


FIG. 5



Ex-1007, Figs. 4 & 5.

420. The combination presents a simple substitution of the processors in Matsunami's channel interface control cards with the multiprocessor system as taught by Ground 4. Therefore, the multiprocessor system is part of a SAN/NAS controller.

421. **Motivation to Combine.** The motivation to combine for Ground 4 was explained above. *Supra* §V.D.2[1b] (Motivation to Combine Subsection). Ground 4 teaches a multiprocessor system. *Supra* §V.D.2 (claim 1). Matsunami teaches a SAN/NAS controller and provides motivation to combine its teachings

with a multiprocessor system. Ex-1007, 5:35-45; 6:10-12. Therefore, a POSITA would have been motivated to combine Ground 4 with Matsunami. In particular, a POSITA would have been motivated to use the multiprocessor system taught by Ground 4 as the processor in Matsunami's channel interface control cards. Ex-1007, 5:35-45; 6:10-12. This would have been straightforward as it is simply combines 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.

422. Khawand provides teachings that are broadly applicable for general multiprocessor architectures. Khawand applies its teachings for "substantially improving the efficiency of a multiprocessor system." Ex-1008, 1:32-34. Khawand's teachings, for delegating service applications between processors, are explained at a computer architecture level, and are thus applicable to multiprocessor architectures generally. E.g., Ex-1008, Fig. 3. Accordingly, a POSITA would have been motivated to apply those teachings, including as enhanced by Rosenthal and Ohkawa, to the processors in Matsunami's channel interface control cards to improve efficiency, including for shared service applications, based on Khawand's teachings. Ex-1008, 1:25-34.

423. Additionally, Khawand identifies and solves issues in ARM-based multiprocessor architectures, as an example. Ex-1008, 1:20-24. By 2013, ARM-based architectures were becoming increasingly popular in servers and data centers. Ex-1031, 000001-4. SAN/NAS was commonly used for servers and in data centers. Ex-1007, 1:10-35.

424. A POSITA would also have been motivated to combine Matsunami with Ground 4 because the references provide teachings related to I/O operations and storage. For example, Khawand's multiprocessor system is described as handling service applications and software applications divided into tasks such as "input/output." Ex-1008, 3:13-17. Additionally, Khawand describes its use of solid-state flash memory for the multiprocessor system. Ex-1008, 3:11-26; 2:63-67. Rosenthal teaches I/O operations, including access to hard disk drives. Ex-1009, Fig. 3. Rosenthal explains that "[t]ypically, various input/output devices are arranged on the input/output bus 27 as bus masters and bus slaves. ... these local bus masters and slaves ... (such as ... hard disk controller unit) which require the most rapid input/output operations for system success." Ex-1009, 4:49-55; *see also* Ex-1009, 21:6-10 ("However, it will be understood that the resource manager should have software commands for performing some of the operations of a graphics output controller, of a sound controller, of a **disk controller**, and any other input/output device which may be connected to the device bus 34.").

Matsunami describes its field of technology as “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. A POSITA looking to improve the performance of Matsunami would naturally look to references like Khawand and Rosenthal, as both were also applicable to storage systems.

425. **Reasonable Expectation of Success.** A POSITA would have had a reasonable expectation of success in combining Matsunami with the multiprocessor system taught by the combination of Ground 4 because the combination uses conventional and well-known techniques to yield a predictable result. Matsunami already states that its interface control cards can be based on multiprocessor systems. Ex-1007, 5:35-45; 6:10-12. The combination simply applies a known multiprocessor system. Accordingly, the combination would not have changed the principle of operation for any of the teachings of the references relied upon in the combination. And moreover, the application of a multiprocessor system architecture would not have fundamentally changed the character of Matsunami’s teachings or application domain of storage control systems, and particularly to SAN and NAS based storage control systems.

426. Additionally, a POSITA would also have expected success because multiprocessor architectures and SAN/NAS were well known. 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. Implementing the combination would not have required specialized skills beyond what a POSITA would have had at the time of the alleged invention.

2. Dependent Claims 9-10

- 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.**

427. Ground 5 teaches Claims 9-10. Claims 9-10 are nearly identical to Claims 3-4, except that Claims 9-10 recite “system of claim 7” while Claims 3-4 recite “system of claim 1.” As I have explained above, the combination of Khawand, Rosenthal, and Ohkawa teaches claim 7, *supra* §V.D.5. For the same reasons I have discussed for Claims 3-4, Ground 5 teaches Claims 9-10. *Supra* §V.E.1.

3. Dependent Claims 15-16

- 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.**

428. Ground 5 teaches Claims 15-16. Claims 15-16 are encompassed by Claims 3-4. Claims 15-16 recite “system of claim 13” while Claims 3-4 recite “system of claim 1.” As I have explained above, the combination of Khawand,

Rosenthal, and Ohkawa teaches claim 13, *supra* §V.D.8. For the same reasons I have discussed for Claims 3-4, Ground 5 teaches Claims 15-16. *Supra* §V.E.1.

VI. BACKGROUND AND QUALIFICATIONS

429. This section contains a summary of my educational background, career history, publications, and other relevant qualifications. My full curriculum vitae is attached as Appendix 1 to this declaration.

430. A summary of my experience, qualifications, and current employment are included in my CV, which is attached as Appendix A. The following is a brief summary of my relevant qualifications and professional experience.

431. I obtained my Ph.D. in Electrical Engineering from the Georgia Institute of Technology (Georgia Tech) in 1991. I worked at a start-up company named DVT from 1992-1994.

432. I taught at Mississippi State University from 1994-1999. While there, I worked on a SuperMSPARC, a system that used low-latency FIFOs to provide inter-processor communications in hardware. I worked to debug the FIFO chips at one point. I also managed a multi-year research project that created a run-time environment to execute complex physical simulations on networked workstations. The workstations used inter-processor communications and a message-passing programming interface to allocate simulation work among multiple processors. All

of these systems entailed multi-threaded programming, maintenance of state information, and the ability to share information between message queues.

433. After that, I worked at a company that designed point-of-sale retail payment terminals from 1999-2000.

434. From 2000-2007, I worked at Scientific-Atlanta (“S-A”), which was later acquired by Cisco in 2005. During that time, I had several responsibilities. I managed an advanced technology group that included individuals with a range of education and years of experience. Our group addressed manufacturing, supply chain, best practices in the field, hardware, and software and made major improvements to then-existing set-top boxes. I was involved in all stages of set-top box development, such as layout, high-speed digital design, emissions considerations, gate array conversion, and designing with custom Application Specific Integrated Circuits (“ASICs”).

435. I designed and developed set-top boxes that implemented a range of technologies, including home networking and multi-room video distribution. For example, my work with multi-room DVR set-top box, won an Engineering and Technology Emmy® Award. Many of the products I worked were widely deployed throughout the cable TV industry because S-A’s customers were the major television service providers, such as Time Warner Cable and Cablevision.

436. Since 2007, I have been a professor of Electrical Engineering at University of South Alabama. As a professor I have taught courses such as EE 457/557 Embedded System Design, EE 469/569 Signal Integrity, EE 548 Cryptography, EE 264 Microprocessors, and EE 368 Microprocessor Lab. While at South Alabama, I managed a team that successfully designed and launched a fully functional satellite. The design of satellite entailed multiple communicating processors that used a CAN bus to exchange state information.

A. Compensation

437. For my efforts in connection with the preparation of this declaration I have been compensated at my standard rate for this type of consulting activity. My compensation is in no way contingent on the results of these or any other proceedings relating to the above-captioned patent.

B. Materials and Other Information Considered

438. I have considered information from various sources in forming my opinions. I have reviewed and considered each of the exhibits listed in the attached Appendix 2 (Materials Considered in the Preparation of This Declaration) in forming my opinions.

VII. UNDERSTANDING OF THE LAW

439. I am not an attorney. In forming my opinions in this Declaration, I applied the relevant legal principles provided to me by counsel, which are summarized in Appendix 4.

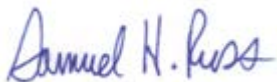
VIII. RESERVATION OF RIGHTS

440. My opinions are based upon the information that I have considered to date. I am unaware of any evidence of secondary considerations with respect to the '596 Patent that would render any of the challenged claims non-obvious. I reserve the right, however, to supplement my opinions in the future to respond to any arguments raised by the owner of the '596 Patent and to take into account new information that becomes available to me.

441. I declare that all statements made herein of my knowledge are true, and that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Executed on March 6, 2025.

By:



Samuel Russ

APPENDIX 1: CURRICULUM VITAE OF DR. SAMUEL RUSS

Curriculum Vitae of Dr. Samuel H. Russ

Personal

Mailing Address: 150 Student Services Dr., SH 4119, Mobile, AL 36688
Phone: (251) 461-1378
e-mail: sruss@southalabama.edu
Citizenship: U.S.

Education

Ph.D. in Electrical Engineering, Georgia Institute of Technology, 1991.
Thesis Title: *An Information-Theoretic Approach to Analysis of Computer Architectures and Compression of Instruction Memory Usage*
Thesis Advisor: Dr. Cecil O. Alford
Bachelor's of Electrical Engineering (with Highest Honor), Georgia Institute of Technology, 1986.

Employment

University of South Alabama

2007 – Present

Interim Chair, Department of Electrical and Computer Engineering (2015-2016)

Associate Professor, Department of Electrical and Computer Engineering (2011-present)

Assistant Professor, Department of Electrical and Computer Engineering

Developing new class in signal integrity, developing cross-disciplinary collaborative research efforts, updating labs and class materials, teaching both undergraduate and graduate classes.

Scientific-Atlanta (Later Cisco, Now Technicolor)

2000 - 2007

Manager, Advanced Technologies R&D, Explored major new technologies including home networking and multi-room video distribution; oversaw construction of prototypes and demonstrations; demonstrated technologies at trade shows and to stock analysts; managed group of approximately 10 designing set-top boxes for the cable television industry including all phases of high-volume electronic design including layout, high-speed digital design, emissions considerations, gate array conversion, and designing with custom ASICs; supervised cross-functional team to increase hard drive reliability; wrote numerous patent applications. Products deployed widely, including Time Warner Cable and Cablevision.

IVI Checkmate (Now Ingenico)

1999 – 2000

Matrix Manager, Electrical Engineering, Supervised staff of approximately 6 electrical engineers, oversaw assignments, allocated time across multiple projects, worked to coordinate project schedules with priorities, assisted in sales and other customer communications.

Staff Electrical Engineer, Oversaw major design overhaul of touch-pad payment system including embedded-systems design, power-system design, and system layout and schematic capture. System deployed widely at Wal-Mart, Home Depot, and others.

Mississippi State University

1994 - 1999

Thrust Leader, Computing Systems Research Thrust and Systems Integration Group, NSF Engineering Research Center -- Coordinate research plans and NSF reporting for 6 faculty and 3 full-time staff, assist in development of strategic plans and pursuit of funding opportunities, oversight of development of the ERC-wide integrated testbed. Managed multi-year research project in distributed computing that included 11 students and 5 faculty.

Assistant Professor, Department of Electrical & Computer Engineering -- Developed all-new graduate class in circuit board design, authored textbook chapter, developing new undergraduate lab, pioneered teaching classes over 2-way interactive video.

Dickerson Vision Technologies (Now Cognex)**1992 - 1994**

Director of Manufacturing--Oversaw the development and implementation of a manufacturing plan for the company.

Senior Applications Engineer--Developed firmware and user-interface software for the company's proprietary hardware, Assisted in development of machine vision system hardware, Assisted in sales duties.

Georgia Institute of Technology**1987 - 1991**

Graduate Research Assistant--Helped design five VLSI chips while developing two custom CPU's, Silicon-compiler-based computer chip design, including initial design, full functional simulation, size and timing optimization, and manufacturing test development.

Publications**Books**

1. Samuel H. Russ, *Signal Integrity: Applied Electromagnetics and Professional Practice*, Springer, 2016. (Textbook)
2. Samuel H. Russ, *Signal Integrity: Applied Electromagnetics and Professional Practice*, Second Edition, Springer, 2022. (Textbook)

Book Chapters

1. Samuel H. Russ, "High-Speed Digital Effects", Chapter 10 of John Kraus and Daniel Fleisch, *Electromagnetics with Applications, Fifth Edition*, McGraw-Hill, 1998. (Textbook)
2. Samuel Russ, Bret Webb, Jonathan Holifield and Justin Walker (2011). "PILS: Low-Cost Water-Level Monitoring, Environmental Monitoring", Dr. E.O. Ekundayo, Ph.D., (Ed.), pp. 137-156, ISBN: 978-953-307-724-6, InTech, (Peer-reviewed)

Journal and Magazine Articles

1. Samuel Russ, "The Threat of Malicious Circuit-Board Alteration: Attack Taxonomy and Examples," *Electronic Device Failure Analysis*, Vol. 23, Issue 3, August 2021, pp. 13-22.
2. Mink DM, McDonald J, Bagui S, Glisson WB, Shropshire J, Benton R, Russ S. Near-Real-Time IDS for the U.S. FAA's NextGen ADS-B. *Big Data and Cognitive Computing*. 2021; 5(2):27. <https://doi.org/10.3390/bdcc5020027>
3. S. H. Russ and J. Gatlin, "Ways to hack a printed circuit board: PCB production is an underappreciated vulnerability in the global supply chain," *IEEE Spectrum*, vol. 57, no. 9, pp. 38-43, Sept. 2020.
4. Samuel H. Russ, Cade C. Cashen, Thomas Thomas, and Roma Hanks, "Smart Grid Architecture for Unobtrusive Home Health Monitoring," *National Social Science Journal*, Vol. 44, No. 1, 2015, pp. 62-68.
5. Samuel H. Russ, Tho D. Ha, and Jonathan H. Shapiro, "Simulation-based Optimization of Wear Leveling for Solid-State Disk Digital Video Recording," *IEEE Transactions on Consumer Electronics*, Vol. 60, No. 3, Nov. 2014, pp. 363-367.
6. Samuel H. Russ and Salim Alsharif, "Packet Loss Behavior of HomePlug AV Traffic at Video Bit Rates", *IEEE Transactions on Consumer Electronics*, Vol. 57, No. 2, May 2011, pp. 823-826.
7. Samuel H. Russ and Ramesh Nallur, "Digital Video Recording onto Solid-State Disks: Feasibility and Lifetime Estimation", *IEEE Transactions on Consumer Electronics*, Vol. 57, No. 2, May 2011, pp. 558-563.
8. Samuel H. Russ and Sasan Haghani, "802.11g Packet-Loss Behavior at High Sustained Bit Rates in the Home", *IEEE Transactions on Consumer Electronics*, Vol. 55, No. 2, May 2009, pp. 788-791.
9. Samuel H. Russ, and Ramesh Nallur, "A Framework for Calculating Fundamental DVR Performance Limits", *IEEE Transactions on Consumer Electronics*, Vol. 55, No. 1, Feb. 2009, pp.

132-138.

10. Ioana Banicescu, Sheikh Ghafoor, Vijay Velusamy, Samuel H. Russ, Mark Bilderback, "Experiences from integrating algorithmic and systemic load balancing strategies", *Concurrency and Computation: Practice and Experience*, Volume 13, Number 2, pp. 121-139 (2001).
11. Roger L. King, Samuel H. Russ, Aric B. Lambert, Donna S. Reese, "An artificial immune system model for intelligent agents", *Future Generation Computer Systems*, Volume 17, Issue 4, pp. 335 - 343 (January 2001).
12. Dr. Samuel H. Russ, Jonathan Robinson, Matt Gleeson, Brad Meyers, and Chun-Heong Tan, "Using Hector to Run MPI Programs over Networked Workstations", *Concurrency - Practice and Experience*, Vol. 11, No. 4, pp. 189-204 (1999).
13. Russ, S.H. Reece, K. Robinson, J. Meyers, B. Rajan, D. Rajagopalan, L. Chun-Heong Tan, "Hector: an agent based architecture for dynamic resource management", *IEEE Concurrency*, Vol. 7, No. 2, pp. 47-55, Apr-Jun 1999.
14. Dr. Samuel H. Russ, Jonathan Robinson, Dr. Brian K. Flachs, and Bjorn Heckel, "The Hector Distributed Run-Time Environment", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 11, pp. 1102-1114 (1998).
15. S.H. Russ and C.O. Alford, "Coupled Noise and Its Effects on Modern Packaging Technology", *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, Volume 13, Number 4, December 1990, pp. 1074-1082.

U.S. Patents

1. U.S. Patent Number 6,748,080, "Apparatus for entitling remote client devices", Samuel Russ, Michael Gaul, Anthony Wasilewski, and Howard Pinder, June 8, 2004.
2. U.S. Patent Number 7,181,010, "Apparatus for entitling remote client devices", Samuel Russ, Michael Gaul, Anthony Wasilewski, and Howard Pinder, February 20, 2007.
3. U.S. Patent Number 7,360,233, "Broadcast carousel system access for remote home communication terminal", Samuel Russ, Michael Gaul, and Anthony Wasilewski, April 15, 2008.
4. U.S. Patent Number 7,360,235, "Systems and methods for operating a peripheral record/playback device in a networked multimedia system", David Davies, Samuel Russ, and Irvan Krantzler, April 15, 2008.
5. U.S. Patent Number 7,499,822, "Analog set-top calibration patterns in manufacturing", Leo Montreuil, Wayne B. Williams, Samuel H. Russ, and Robert Kriete, March 3, 2009
6. U.S. Patent Number 7,505,592, "Apparatus for entitling and transmitting service instances to remote client devices", Samuel H. Russ, Michael Gaul, Anthony Wasilewski, and Howard Pinder, March 17, 2009
7. U.S. Patent Number 7,516,470, "Locally-updated interactive program guide", Samuel H. Russ and Michael Gaul, April 7, 2009
8. U.S. Patent Number 7,826,711, "Digital Video Disc (DVD) Player or Digital Video Recorder (DVR) Playback with Memory", Samuel H. Russ, Nov. 2, 2010.
9. U.S. Patent Number 7,849,486, "Networked subscriber television distribution", Samuel H. Russ, David Lett, Jonathan Robinson, and Michael Gaul, Dec. 7, 2010.
10. U.S. Patent Number 7,860,250, "Apparatus for entitling and transmitting service instances to remote client devices", Samuel H. Russ, Michael Gaul, Anthony Wasilewski, and Howard Pinder, Dec. 28, 2010.
11. U.S. Patent Number 7,861,272, "Networked subscriber television distribution", Samuel H. Russ, David Lett, Jonathan Robinson, and Michael Gaul, Dec. 28, 2010.
12. U.S. Patent Number 7,870,584, "Interactive program guide with selectable updating", Samuel H. Russ and Michael Gaul, Jan. 11, 2011.
13. U.S. Patent Number 7,876,998, "DVD playback over multi-room by copying to HDD", William Wall and Samuel H. Russ, Jan. 25, 2011.

14. U.S. Patent Number 7,908,625, "Networked multimedia system", Neil C. Robertson, David B. Lett, Samuel H. Russ, Bohdan S. Prus, Michael A. Gaul, and Ajith N. Nair, March 15, 2011.
15. U.S. Patent Number 7,937,737, "Field qualification of disk drives in consumer electronics devices", Samuel H. Russ and Mark R. Murray, May 3, 2011.
16. U.S. Patent Number 7,961,780, "Generated set top calibration patterns in manufacturing", Leo Montreuil, Wayne B. Williams, Samuel H. Russ, and Robert Kriete, Jun. 14, 2011.
17. U.S. Patent Number 7,978,720, "Digital media device having media content transfer capability", Samuel H. Russ, Christopher L. Stallings, July 12, 2011.
18. U.S. Patent Number 8,046,806, "Multiroom point of deployment module", William E. Wall, Samuel H. Russ, Oct. 25, 2011.
19. U.S. Patent Number 8,127,236, "Proximity detection using wireless connectivity in a communications system", Paul J. Claussen, Ajith N. Nair, and Samuel H. Russ, Feb. 28, 2012.
20. U.S. Patent Number 8,181,205, "PVR Channel and PVR IPG Information", Samuel H. Russ, Michael A. Gaul, and Dariusz S. Kaminski, May 15, 2012.
21. U.S. Patent Number 8,208,796, "Systems and methods for prioritizing the storage location of media data", Bohdan Prus and Samuel H. Russ, June 26, 2012.
22. U.S. Patent Number 8,280,229, "DVD playback over multi-room by copying to HDD", William E. Wall and Samuel H. Russ, October 2, 2012.
23. U.S. Patent Number 8,295,338, "Generated set top calibration patterns in manufacturing", Leo Montreuil, Wayne B. Williams, Samuel H. Russ, and Robert A. Kriete, October 23, 2012.
24. U.S. Patent Number 8,364,015, "Stretch and zoom bar for displaying information", Samuel H. Russ and Gary Hibbard, January 29, 2013.
25. U.S. Patent Number 8,549,567, "Media content sharing over a home network", Samuel H. Russ, David B. Lett, Jonathan Robinson, Michael A. Gaul, October 1, 2013.
26. U.S. Patent Number 8,627,385, "Systems and methods for operating a peripheral record playback device in a networked multimedia system", David B. Davies, Samuel H. Russ, Irvan J. Krantzler, January 7, 2014.
27. U.S. Patent Number 8,966,550, "Home communication systems," David B. Davies, Samuel H. Russ, Irvan J. Krantzler, February 24, 2015.
28. U.S. Patent Number 9,762,970, "Access of stored video from peer devices in a local network," David B. Davies, Samuel H. Russ, and Irvan J. Krantzler, September 12, 2017.
29. U.S. Patent Number 10,412,439, "PVR Channel and PVR IPG Information", Samuel H. Russ, Michael A. Gaul, and Dariusz S. Kaminski, Sep. 10, 2019.

European Patents

1. European Patent Number EP1543680B1, "Locally-updated interactive program guide", Samuel Russ and Michael Gaul, Feb. 17, 2010.
2. European Patent Number EP1673937B1, "Proximity detection using wireless connectivity in a communications system", Paul Claussen, Ajith Nair, and Samuel Russ, April 21, 2010.
3. European Patent Number EP1604523B1, "Method for determining the encryption of a service content depending on the fidelity of the reformatting of said content", Samuel Russ, Michael Gaul, Anthony Wasilewski, and Howard Pinder, Nov. 21, 2012.
4. European Patent Number EP1552699B1, "Networked multimedia system", Neil Robertson, David Lett, Samuel Russ, Bohdan Prus, Michael Gaul, and Ajith Nair, Dec. 18, 2013.
5. European Patent Number EP1334617B1, "Networked Subscriber Television Distribution", Samuel Russ, David Lett, Jonathan Robinson, and Michael Gaul, April 1, 2015.
6. European Patent Number EP1920601B1, "Multiroom Point of Deployment Module," William Wall and Samuel Russ, May 4, 2016.
7. European Patent Number EP1510033B1, "Apparatus for Entitling Remote Client Devices," Samuel H. Russ, Michael Gaul, Anthony Wasilewski, and Howard Pinder, Aprl 26, 2017.
8. European Patent Number EP1540946B1, "Broadcast Carousel System Access for Remote Home

Communications Terminal via a Gateway Device,” Samuel Russ, Michael Gaul, and Anthony Wasilewski, April 18, 2018.

9. European Patent Number EP2008280B1, “Method and Device for Prioritizing the Storage Location of Media Data,” Bohdan Prus and Samuel Russ, September 26, 2018.

Conference Papers

1. S.H. Russ, M. Khan, S. Latif, “Surreptitiously Adding a WLAN Antenna to a Printed Circuit Board” 2022 IEEE Physical Assurance and Inspection of Electronics (PAINE), Huntsville, AL, USA, 2022.
2. J. Todd McDonald, Rebecca C. Clark, Lee M. Hively, and Samuel H. Russ. 2022. “Phase space power analysis for PC-based rootkit detection.” In *Proceedings of the 2022 ACM Southeast Conference (ACM SE '22)*. Association for Computing Machinery, New York, NY, USA, 82–90. <https://doi.org/10.1145/3476883.3520212>.
3. S. H. Russ, " Surreptitiously Adding a Microcontroller to a Printed Circuit Board," 2021 IEEE Physical Assurance and Inspection of Electronics (PAINE), Washington, DC, USA, 2021.
4. S.H. Russ, “Circuit Board Security Vulnerabilities and Counteractions”, IPC APEX EXPO 2021.
5. S. H. Russ, "Techniques to Thwart Surreptitiously Altered PCBs," 2020 IEEE Physical Assurance and Inspection of Electronics (PAINE), Washington, DC, USA, 2020, pp. 1-4.
6. S. H. Russ, E. Spencer, T. Scroggins, and S. Latif, “Power-Efficient Software Architecture for a CubeSat System”, *2020 SoutheastCon*, Raleigh, NC, USA, 2020, pp. 1-6.
7. A. Russ, S. H. Russ, E. Spencer and M. Frank, "Motor Controller and Reaction Wheel for CubeSat," *2019 SoutheastCon*, Huntsville, AL, USA, 2019, pp. 1-3.
8. F. Lorenzo, J. T. McDonald, T. R. Andel, W. B. Glisson and S. Russ, "Evaluating Side Channel Resiliency in iPhone 5c Unlock Scenarios," *2019 SoutheastCon*, Huntsville, AL, USA, 2019, pp. 1-7.
9. E. Spencer, D. Clark, S. Russ, *et al.*, "First results from a time domain impedance probe for measuring plasma properties in the ionosphere," 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, 2017, pp. 1284-1287.
10. E. Spencer, D. Clark, R. Gollapalli, S. Russ and B. Kerrigan, "A System On Chip design for fast time domain impedance spectroscopy," 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, 2017, pp. 124-127.
11. Ramya Manikyam, J. Todd McDonald, William R. Mahoney, Todd R. Andel, and Samuel H. Russ, “Comparing the effectiveness of commercial obfuscators against MATE attacks,” *Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering (SSPREW '16)*, 2016, Article 8. DOI: <https://doi.org/10.1145/3015135.3015143>
12. Samuel Russ, Tho Ha, and Jonathan Shapiro, “Simulation of Digital Video Recording to Maximize Solid-State Disk Lifetime,” *Proceedings of the 2014 International Conference on Consumer Electronics*, Jan. 10-13, 2014, pp. 562-563.
13. Thomas Thomas, Cade Cashen, and Samuel Russ, “Leveraging Smart Grid Technology for Home Health Care”, *Proceedings of the 2013 International Conference on Consumer Electronics*, Jan. 11-14, 2013, pp. 276-277.
14. Cade Cashen, Samuel Russ, and Thomas Thomas, “Using a Wireless LAN to Perform Motion Detection”, *Proceedings of the 2013 International Conference on Consumer Electronics*, Jan. 11-14, 2013, pp. 63-64.
15. Nicklaus Thomas, David Evans, and Samuel Russ, “A Markovian Algorithm for Creating Immersive Public-Speaking Audiences”, *Proceedings of the 2013 International Conference on Consumer Electronics*, Jan. 11-14, 2013, pp. 384-385.
16. Samuel H. Russ, “Podcasting to Improve Software Usage”, *Proceedings of the 2012 ASEE Southeastern Section Conference*, April 1-3 2012, Starkville, MS, Paper ID 605.
17. Samuel H. Russ, “Using Student ID Numbers to Create Customized Homework Problems”, *Proceedings of the 2012 ASEE Southeastern Section Conference*, April 1-3 2012, Starkville, MS,

Paper ID 606.

18. Samuel H. Russ and Ramesh Nallur, "Digital Video Recording Onto Solid-State Disks", *2011 Digest of Technical Papers, 2011 International Conference on Consumer Electronics*, pp. 495-496. (2nd place in best paper contest, 2011.)
19. Mohammad Mushtaq, Samuel H. Russ, and Mohammad S. Alam, "Variable FEC to Reduce ECC Overhead in Digital Video Transmission", *2011 Digest of Technical Papers, 2011 International Conference on Consumer Electronics*, pp. 215-216.
20. Samuel H. Russ and Salim Alsharif, "Behavior of HomePlug AV Traffic At High Sustained Bit Rates in the Home", *2011 Digest of Technical Papers, 2011 International Conference on Consumer Electronics*, pp. 682-683.
21. Samuel H. Russ, Viswakalyan Perepa, Silas Leavesly, and Bret Webb, "Novel Low-Cost Salinity Sensor for Embedded Environmental Monitoring", *Proceedings of IEEE Southeastcon 2010*, March 18-21, 2010, pp. 53-56.
22. Samuel H. Russ and Kung-Hsien Chen, "Thermally Conductive Circuit Boards", *Proceedings of the 59th Electronic Components and Technology Conference (ECTC 2009)*, May 2009, pp. 465-468.
23. Samuel H. Russ and Sasan Haghani, "Behavior of 802.11g Traffic at High Sustained Bit Rates in the Home", *2009 Digest of Technical Papers, 2009 International Conference on Consumer Electronics, Jan. 2009*, pp. 423-424.
24. Samuel H. Russ, and Ramesh Nallur, "A Framework for Calculating Fundamental DVR Performance Limits", *2009 Digest of Technical Papers, 2009 International Conference on Consumer Electronics, Jan. 2009*, pp. 51-52.
25. Samuel H. Russ, "Home Networking: Extending Cable's Reach", *The Complete Technical Paper Proceedings from NCTA*, 2001, pp. 160-165.
26. Samuel H. Russ, Rashid Jean-Baptiste, Tangirala Shailendra Krishna Kumar, and Marion G. Harmon, "Transparent Real-Time Monitoring in MPI", *IPPS/SPDP Workshops 1999*, in *Parallel and Distributed Processing*, Lecture Notes in Computer Science, vol. 1586, Springer-Verlag, 1999, pp. 1327-1334.
27. Roger L. King, Aric B. Lambert, Samuel H. Russ, and Donna S. Reese, "The Biological Basis of the Immune System as a Model for Intelligent Agents", *IPPS/SPDP Workshops 1999*, in *Parallel and Distributed Processing*, Lecture Notes in Computer Science, vol. 1586, Springer-Verlag, 1999, pp. 156-164.
28. Aric Lambert, Roger King, Sam Russ, and Donna Reese, "Intelligent Control Agents for Resource Management of Heterogeneous Parallel Computing", *Computational Intelligence for Modeling, Control, and Automation (CIMCA 98) in Concurrent Systems Series*, vol. 55, IOS Press, 1999, pp. 116-121.
29. Samuel H. Russ and Adam Gaither, "Developing An Integrated Environment for Computational Field Simulation", *Object-Oriented Methods for Inter-Operable Scientific and Engineering Computing: Proceedings of the 1998 SIAM Workshop*, pp. 50-57.
30. Samuel H. Russ, Ioana Banicescu, Sheikh Ghafoor, Bharathi Janapareddi, Jonathan Robinson, and Rong Lu, "Hectiling: An Integration of Fine and Coarse-Grained Load-Balancing Strategies", *Proceedings of the Seventh International Symposium on High Performance Distributed Computing*, August 1998, IEEE Computer Society, pp. 106-113.
31. Samuel H. Russ, "An Architecture for Rapid Distributed Fault Tolerance", 3rd International Workshop on Embedded High-Performance Computing, in J. Rolim, Editor, *Parallel and Distributed Processing*, Lecture Notes in Computer Science, vol. 1388, Springer-Verlag, 1998, pp. 925-930.
32. Dr. Samuel H. Russ, Brad Meyers, Jonathan Robinson, Matt Gleeson, Laxman Rajagopalan, Chun-Heong Tan, and Bjørn Heckel, "User-Transparent Run-Time Performance Optimization",

Proceedings of the 2nd International Workshop on Embedded HPC Systems and Applications, Geneva, April 1997.

33. J. Robinson, S.H. Russ, B. Flachs, and B. Heckel, "A Task Migration Implementation of the Message-Passing Interface", *Proceedings of the 5th High Performance Distributed Computing Conference (HPDC-5)*, August 1996, IEEE Computer Society, pp. 61-68.
34. S.H. Russ, B. Flachs, J. Robinson, and B. Heckel, "Hector: Automated Task Allocation for MPI", *Proceedings of the 10th International Parallel Processing Symposium*, April, 1996, IEEE Computer Society, pp. 344-348.
35. W.S. Tan, C.O. Alford, and S.H. Russ, "GT-EP: A Novel High-Performance Real-Time Architecture", *Proceedings of the 18th International Symposium on Computer Architecture*, May 1991, pp. 12-21.

Reports

1. Aric B. Lambert, Roger L. King, Samuel H. Russ, Donna S. Reese, "Adaptive Analysis for the Design of Hardware Agents Using the Artificial Immune System Model for Resource Management of Heterogeneous Systems", Miss. State Technical Report No. MSSU-COE-ERC-98-10, Aug. 1998.
2. Daniel H. Linder, Robert B. Reese, Jon Robinson, and Samuel H. Russ, "JavaCADD: A Java-based Server and GUI for Providing Distributed ECAD Services", Mississippi State University Technical Report No. MSSU-COE-ERC-98-07, May 1998.
3. Samuel H. Russ, "Using Hector in an Architecture for Rapid Distributed Fault Tolerance", Mississippi State Technical Report No. MSSU-EIRS-ERC-97-17, December 1997.
4. Samuel H. Russ, Jonathan Robinson, Matt Gleeson, and Jose Figueroa, "Dynamic Communication Mechanism Switching in Hector", Mississippi State Technical Report No. MSSU-EIRS-ERC-97-8, September 1997.
5. Samuel H. Russ, Brian Flachs, Jonathan Robinson, and Bjorn Heckel, "Hector: Automated Task Allocation for MPI", Mississippi State Technical Report No. MSSU-EIRS-ERC-95-6, September, 1995.

Funded Grants and Contracts

University of South Alabama

1. National Space Grant Foundation, Aug. 2018-May 2019, "Automation Infrastructure for Gateway Biological Laboratory," PI, \$28,062.
2. Nfina (Local Mobile Company), March 2018 – March 2019, "Redhawk", PI, \$119,881.
3. NASA, May 2016 – Present, "Spatially Resolved Measurements Of Plasma Density Irregularities In The Ionosphere F Region For Scintillation Studies", Co-I, \$100,000.
4. Marine Environmental Science Consortium (GRI), Jan. 2011-Dec. 2011, "Conductivity Based In-Situ Hydrocarbon Sensor", PI, \$9,338.
5. University of South Alabama Research Council (USARC), Feb. 2008 – Feb. 2009, "Instrumentation of Hurricane Forecast Models for Performance Optimization", \$5,000 for one year.

Mississippi State University

1. National Science Foundation, April 1998-March 1999, "An ERC-CREST Partnership in Distributed and Computational Systems", PI, \$100,000/year, renewable for 5 years.
2. Office of Naval Research, Nov. 1996-Nov. 1998, "Distributed Visualization and Virtual Environments for Oceanography and Meteorology", Co-PI, \$200,000.
3. U.S. Army Waterways Experiment Station, July 1996-July 1999, "Efficient Parallel Wave Modeling", PI, \$200,000, Funded under CHSSI (Common HPC Scalable Software Initiative) of the DoD. (\$50,000 actually funded.)

4. NSF Engineering Research Center for Computational Field Simulation, April 1995- May 1999, "Heterogeneous Computing Testbed (CA 8)", PI, By year: \$34,983/\$70,111/\$55,916. Funded through NSF Engineering Research Center.

Dickerson Vision Technology

1. Small Business Innovations Research Program of the NSF, Division of Industrial Interface, September 1993 - February 1994, Award No. III-9260914. (\$50,000 Phase I contract conducted at Dickerson Vision Technologies.)

Technology Transfer and Licensing

Hector Distributed Run-Time Environment licensed by MPI Software Technology, Inc., Starkville, MS in June 1998 (while at Mississippi State University)

Students Advised (Major Professor)

Level	At USA	At MSU
Ph.D.	--	0
Masters	3	9

Courses Created and Enhanced

University of South Alabama

EE 264 Microprocessor Systems and Interfacing – Replaced 68HC11 with Freescale 9S12 architecture and redesigned the entire course, Introduced use of 68HC11 simulator and accompanying podcast to enhance student's ability to visualize microprocessor operation

EE 268 Digital Logic Design Lab – Updated one lab experiment

EE 368 Microprocessor Systems Lab – Developed all-new sequence to tie EE 368 to senior projects with hands-on experiments like motors, photocells, and servomotors.

EE 401 Introduction to ECE Design – Completely overhauled the first-semester senior project course. Added lectures on professionalism, career development, and legal aspects of engineering. Structured the course so that reports would lead up to their senior-project reports.

EE 404 ECE Design – Completely overhauled the second-semester senior project course by requiring better projects and adding material on giving effective presentations. Obtained very enthusiastic response from the department's Industrial Advisory Board.

EE 446 – Obtained donation of boards from Freescale and redesigned all of the experiments to develop more interesting demonstrations and to switch from assembly language to C.

EE 457/557 – Developing series of podcasts to augment in-class lectures with materials related to embedded systems and professional development.

EE 469/569 Advanced Digital System Design – Developed all-new course in high-speed circuit board design from an electromagnetics perspective, and authored a textbook for the course. Added a lab experience in 2009.

EE 560 Advanced Computer Architecture – Selected new textbook for the class and developed lecture materials to match

EG 101 – Developed systematic curriculum shared by all sections including "checklists" for instructors, handouts, and lecture materials. Introduced motors project, which has become the most popular EG 101 project.

Mississippi State University

EE 8993 High Speed Digital Design -- Offered on video nationwide, it features pioneering treatment of high-speed circuit board design from an electromagnetics perspective. Material from the class has been included in one of the best-known electromagnetics textbooks, and it continues to be one of the most popular graduate classes in the department.

EE 4743 Computer Aided Digital Design -- Added breakthrough educational technology to permit remote access to logic synthesis capabilities. Offered on live two-way video starting Fall 1998.

Courses Taught

University of South Alabama

EE 220, Circuit Analysis I, 1 course.
EE 263, Digital Logic Design, 2 courses.
EE 264, Microprocessor Sys-Interfacing, 4 courses.
EE 268, Digital Logic Design Lab, 1 course.
EE 368, Microprocessor Sys Interf Lab, 14 courses.
EE 401, Intro Elec and CpE Design - W, 7 courses.
EE 404, Elec and Comp Eng Designs-W, 7 courses.
EE 440, HDL Logic Synthesis, 1 course.
EE 446, Embedded System Design Lab, 2 courses.
EE 457, Embedded System Design, 10 courses.
EE 469, Signal Integrity, 5 courses.
EE 534, VLSI Design Systems, 1 course.
EE 540, HDL Logic Synthesis, 1 course.
EE 548, Computer-Network Security, 5 courses.
EE 557, Adv. Embedded System Design, 8 courses.
EE 560, Adv Computer Architecture, 3 courses.
EE 569, Signal Integrity, 5 courses.
EG 101, Intro to Engineering & Design, 5 courses.
EG 231, Intro to Ethics and Economics, 2 courses.

Mississippi State University

Number	Description	Class Description
EE 2111	Electrical Lab I	Accompanies introductory digital class
EE 2713	Digital Devices	Introductory digital class
EE 4723	Microcomputers II	Microcontroller interfacing
EE 4743	Computer Aided Digital Design	VHDL, logic design, and synthesis
EE 8993	High-Speed Digital Design	Signal integrity and high-speed design

Honors and Awards

University of South Alabama College of Engineering "Teacher of the Year" 2015

Mobile Area Council on Engineering "Engineering Educator of the Year" 2015

Technology and Engineering Emmy® Award, 2012, for "Pioneering Development of Multi-Room DVR." Awarded to Cisco. I managed the groups that built the first prototypes, gave public demonstrations, wrote numerous patent applications, and developed the first mass-production set-top containing Multi-Room DVR.

2nd Place in "Best Paper – Storage Category" at ICCE 2011. The paper is "Digital Video Recording Onto Solid-State Disks" as noted in Conference publications above.

Attendee, "Top Prof" Banquet – University of South Alabama chapter of Mortar Board, 2009,2010

Tau Beta Pi Engineering Professor of the Year – University of South Alabama chapter of Tau Beta Pi, 2008. One of two highest awards in the College of Engineering for teaching, and the only award voted on

by honor students in the College.

Finalist, 53rd International Science and Engineering Fair, 1983, Albuquerque, NM.

Professional Activities

Expert Witness Engagements

Case	Jurisdiction and Case Number	Year(s)	Witness for	Expert Reports	Dep.	Trial	IPR* Rep.
WNE Holdings v. Rockwell Automat.	Southern District of Alabama	2011	Plaintiff	1	0	0	0
Personalized Media Corp. (PMC) v. Echostar	Eastern District of Texas	2010-2015	Plaintiff	2	3	0	0
Pegasus v. DirecTV et al.	Delaware	2011-13	Plaintiff	1	0	0	0
Apple v. Samsung	Int'l Trade Commission 337-TA-796	2012	Respondent	4	1	1	0
EON v. MobiTV et al.	Delaware	2013	Defendant	1	1	0	0
PMC v. Zynga	Eastern District of Texas	2013	Plaintiff	1	5	0	4
OpenTV v. Netflix	Delaware	2014	Defendant	0	0	0	0
Kaplan v. OnForce	N.D. California	2014	Defendant	0	0	0	1
Intellectual Ventures v. J.P. Morgan Chase	Southern District of New York	2013-2019	Defendant	2	2	0	0
PMC v. Amazon	Delaware	2014-2015	Plaintiff	1	9	0	5
Better Mouse Corp. v. Steelseries et al.	Eastern District of Texas	2015-2016	Plaintiff	2	0	1	0
SyncPoint v. Nintendo et al.	Eastern District of Texas	2015	Plaintiff	2	1	0	0
TQ Beta v. Dish	USPTO (IPR)	2016	Defendant	1	1	0	0
Script v. ADT et al.	Eastern District of Texas	2016	Plaintiff	2	2	0	0
Sony v. Pace	USPTO (IPR)	2016	Defendant	0	1	0	4
PMC v. Apple	EDTX and USPTO (IPR)	2016-Now	Patent Owner	2	1	1	2
PMC v. Samsung	Eastern District of Texas	2016	Patent Owner	2	0	0	0
Music Choice v. Stingray	Eastern District of Texas	2016-now	Plaintiff	2	2	0	4
Broadcom v. AMD	USPTO (IPR)	2016	Petitioner	0	0	0	1
Tech Pharmacy v. Alixa et al.	Eastern District of Texas	2017	Plaintiff	2	2	1	0
Rovi v. PMC	Eastern District of Texas, Arbitration	2017	Defendant	0	1	0	0

Case	Jurisdiction and Case Number	Year(s)	Witness for	Expert Reports	Dep.	Trial	IPR* Rep.
Implicit v. Trend Micro	Eastern District of Texas	2017	Defendant	2	1	0	0
Implicit v. Palo Alto	Eastern District of Texas	2018	Defendant	1	1	0	0
Invensas v. Samsung	Eastern District of Texas	2018	Plaintiff	2	1	0	0
Selective Signals v. Palo Alto	USPTO (IPR)	2018	Petitioner	0	1	0	1
Rovi Tivo v. Various	Canada	2018-Now	Plaintiff	0	0	0	0
Implicit v. Netscout	Eastern District of Texas	2018	Defendant	0	0	0	0
Rovi Tivo v. Comcast	USPTO (IPR)	2019-2020	Plaintiff	0	4	0	5
Rovi Tivo v. Comcast	ITC	2019-2020	Petitioner	2	1	1	0
American Patents v. Various	Eastern District of Texas	2019-Now	Plaintiff	0	0	0	0
BBiTV v. OpenTV	California	2019	Defendant	0	0	0	0
Roku v UEI	USPTO (IPR)	2019-Now	Petitioner	0	6	0	8
MV3 v Roku	Western District of Texas	2019-2020	Defendant	2	2	1	0
BBiTV v Dish	USPTO (IPR)	2019-2020	Petitioner	0	3	0	6
PMC v Google	Eastern District of Texas	2020	Plaintiff	1	1	0	0
PMC v Netflix	Southern District of New York	2020	Plaintiff	2	0	0	0
UEI v Roku <i>et al.</i>	ITC	2020-2021	Respondent	1	1	1	0
BBiTV v Dish	Western District of Texas	2020-2021	Defendant	1	5	0	6
Flexiworld v Amazon <i>et al.</i>	Western District of Texas	2020-Now	Defendant	0	0	0	0
Flexiworld v Roku	Western District of Texas	2021-2022	Defendant	2	0	0	0
Roku v UEI <i>et al.</i>	ITC	2021-2022	Complain.	1	0	1	0
Roku v DivX	USPTO (IPR)	2021	Petitioner	0	0	0	1
Apple v Identity	USPTO (IPR), Ex Parte Re-Exam, Western District of Texas	2022-2024	Patent Owner	2	1	1	6
Audi v Aprese	USPTO (IPR)	2022	Petitioner	0	0	0	2
paSafeShare	Ex Parte Re-Exam, Western District of	2021-Now	Patent Owner,	1	0	0	3

Case	Jurisdiction and Case Number	Year(s)	Witness for	Expert Reports	Dep.	Trial	IPR* Rep.
	Texas		Plaintiff				
PerfectVision v PPC Broadband	USPTO (IPR)	2022-Now	Petitioner	0	0	0	1
Google v Nobots	USPTO (IPR)	2022	Patent Owner	0	2	0	4
Fall Line v Various	Eastern District of Texas	2022-Now	Plaintiff	2	1	0	0
Assa Abloy v CPC	USPTO (IPR)	2023-	Patent Owner	0	2	0	4
IOENGINE v Roku	Western District of Texas	2023-2024	Defendant	5	2	1	0
Gratuity v Toast	USPTO (IPR)	2023-Now	Petitioner	0	1	0	1
Lexos v Ebay	USPTO (IPR)	2024-Now	Patent Owner	0	1	0	1
Entropic v Various	USPTO (IPR)	2024-Now	Patent Owner	0	1	0	7
Roku v Anonymous	USPTO (IPR)	2024-Now	Petitioner	0	0	0	6
Adeia v Videotron	Fed. Court of Canada	2024-Now	Plaintiff	2	0	0	0

* Includes reports submitted for IPRs, CBMRs, and Re-Exams

Invited Lectures

“Real-time Monitoring in MPI”, Florida A&M University, Tallahassee, FL, August 1998

Consulting

Consulted with various investment firms concerning telecommunications market, 2007 – ongoing
Cox Communications – DVR Reliability – 2015

Society Memberships

Institute of Electrical and Electronic Engineers (Senior Member), IEEE Consumer Electronics Society, IEEE Computer Society, Eta Kappa Nu, Tau Beta Pi, Phi Kappa Phi, Gamma Beta Phi
Secretary, Northeast Mississippi Subsection of the IEEE, 1994-95

Article Reviewer

Computers and Electrical Engineering, Supercomputing '94, IEE Transactions, Concurrency: Practice and Experience, International Parallel Processing Symposium, IEEE Transactions on Education, IEEE Transactions on Parallel and Distributed Systems, ICT 2012, ICAEE 2012

Proposal Reviewer

Army Research Office, National Science Foundation, U.S. Department of Energy

University Service

Chair, Search Committee, ECE Department – Search for Assistant Professor positions, 2008-now (chair since 2011)

Member, ECE Department Undergraduate Affairs Committee, 2015-now

Member, College of Engineering Academic Standards Committee, 2016-now

Member, Search Committee, College of Engineering – Search for 2 Associate Dean positions, 2015

Member, Search Committee, College of Engineering – Search for Associate Dean positions, 2008-09
Member, University of South Alabama Council on International Education, 2008-15
Chairman, ECE Department “Open House” Committee, 2008-09
Conducted open house and other demonstrations for the department, 2008-now
Chairman, ECE Department Recruiting and Retention Committee, 2007-08
Updated and conducted experiment for ACE program, 2007
Member, Hearin Undergraduate Committee, 1998
Department Coordinator, United Way Campaign, 1995
Award Referee, 1995 Instructional Paper Award, MSU College of Engineering

APPENDIX 2: MATERIALS CONSIDERED IN THE PREPARATION OF THIS DECLARATION

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”)
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,” <i>Communications of the ACM</i> , 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

Exhibit No.	Description
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
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 Grief, et al. (“Grief”)
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

Exhibit No.	Description
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_arm_servers_all_over_open_compute/ , published on January 17, 2013.

APPENDIX 3: CHALLENGED CLAIMS

[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.

APPENDIX 4: UNDERSTANDING OF THE LAW

I have applied the following legal principles provided to me by counsel in arriving at the opinions set forth in this report.

Legal Standard for Prior Art

I am not an attorney. I have been informed by attorneys of the relevant legal principles and have applied them to arrive at the opinions set forth in this declaration.

I understand that the petitioner for inter partes review may request the cancelation of one or more claims of a patent based on grounds available under 35 U.S.C. § 102 and 35 U.S.C. § 103 using prior art that consists of patents and printed publications.

Anticipation and Prior Art

I understand that § 102 specifies when a challenged claim is invalid for lacking novelty over the prior art, and that this concept is also known as “anticipation.” I understand that a prior art reference anticipates a challenged claim, and thus renders it invalid by anticipation, if all elements of the challenged claim are disclosed in the prior art reference. I understand the disclosure in the prior art reference can be either explicit or inherent, meaning it is necessarily present or implied. I understand that the prior art reference does not have to use the same words as the challenged claim, but all of the requirements of the claim

must be disclosed so that a person of ordinary skill in the art could make and use the claimed subject-matter.

In addition, I understand that § 102 also defines what is available for use as a prior art reference to a challenged claim.

Under § 102(a)(1), a challenged claim is anticipated if it was patented, described in a printed publication, or in public use, on sale, or otherwise available to the public before the effective filing date of the challenged claimed invention.

Under § 102(a)(2), a challenged claim is anticipated if it was described in an issued patent, or in an application for patent that was published by the USPTO, in which the patent or application, as the case may be, names another inventor and was effectively filed before the effective filing date of the challenged claimed invention.

I understand that the filing date of a patent is generally the filing date of the application filed in the United States that issued as the patent. However, I understand that a patent may be granted an earlier effective filing date if the patent owner properly claimed priority to an earlier patent application.

I understand that when a challenged claim covers several structures, either generically or as alternatives, the claim is deemed anticipated if any of the structures within the scope of the claim is found in the prior art reference.

I understand that when a challenged claim requires selection of an element from a list of alternatives, the prior art teaches the element if one of the alternatives is taught by the prior art.

Legal Standard for Obviousness

I understand that even if a challenged claim is not anticipated, it is still invalid if the differences between the claimed subject matter and the prior art are such that the claimed subject matter would have been obvious to a person of ordinary skill in the pertinent art at the time the alleged invention.

I understand that obviousness must be determined with respect to the challenged claim as a whole.

I understand that one cannot rely on hindsight in deciding whether a claim is obvious.

I also understand that an obviousness analysis includes the consideration of factors such as (1) the scope and content of the prior art, (2) the differences between the prior art and the challenged claim, (3) the level of ordinary skill in the pertinent art, and (4) “secondary” or “objective” evidence of non-obviousness.

Secondary or objective evidence of non-obviousness includes evidence of: (1) a long felt but unmet need in the prior art that was satisfied by the claimed invention; (2) commercial success or the lack of commercial success of the claimed invention; (3) unexpected results achieved by the claimed invention; (4)

praise of the claimed invention by others skilled in the art; (5) taking of licenses under the patent by others; (6) deliberate copying of the claimed invention; and (7) contemporaneous and independent invention by others. However, I understand that there must be a relationship between any secondary evidence of non-obviousness and the claimed invention.

I understand that a challenged claim can be invalid for obviousness over a combination of prior art references if a reason existed (at the time of the alleged invention) that would have prompted a person of ordinary skill in the art to combine elements of the prior art in the manner required by the challenged claim. I understand that this requirement is also referred to as a “motivation to combine,” “suggestion to combine,” or “reason to combine,” and that there are several rationales that meet this requirement.

I understand that the prior art references themselves may provide a motivation to combine, but other times simple common sense can link two or more prior art references. I further understand that obviousness analysis recognizes that market demand, rather than scientific literature, often drives innovation, and that a motivation to combine references may come from market forces.

I understand obviousness to include, for instance, scenarios where known techniques are simply applied to other devices, systems, or processes to improve them in an expected or known way. I also understand that practical and common-

sense considerations should be applied in a proper obviousness analysis. For instance, familiar items may have obvious uses beyond their primary purposes.

I understand that the combination of familiar elements according to known methods is obvious when it yields predictable results. For instance, obviousness bars patentability of a predictable variation of a technique even if the technique originated in another field of endeavor. This is because design incentives and other market forces can prompt variations of it, and predictable variations are not the product of innovation, but rather ordinary skill and common sense.

I understand that a particular combination may be obvious if it was obvious to try the combination. For example, when there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions, a person of ordinary skill has good reason to pursue the known options within his or her technical grasp. This would result in something obvious because the result is the product not of innovation but of ordinary skill and common sense. However, I understand that it may not be obvious to try a combination when it involves unpredictable technologies.

It is further my understanding that a proper obviousness analysis focuses on what was known or obvious to a person of ordinary skill in the art, not just the patentee. Accordingly, I understand that any need or problem known in the field of

endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed.

It is my understanding that the Manual of Patent Examining Procedure §2143 sets forth the following as exemplary rationales that support a conclusion of obviousness:

- Combining prior art elements according to known methods to yield predictable results;
- Simple substitution of one known element for another to obtain predictable results;
- Use of known technique to improve similar devices (methods, or products) in the same way;
- Applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;
- Choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;
- Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to one of ordinary skill in the art;

- Some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

A person of ordinary skill in the art looking to overcome a problem will often use the teachings of multiple publications together like pieces of a puzzle, even though the prior art does not necessarily fit perfectly together. Therefore, I understand that references for obviousness need not fit perfectly together like puzzle pieces. Instead, I understand that obviousness analysis takes into account inferences, creative steps, common sense, and practical logic and applications that a person of ordinary skill in the art would employ under the circumstances.

I understand that a claim can be obvious in light of a single reference, if the elements of the challenged claim that are not explicitly or inherently disclosed in the reference can be supplied by the common sense of one of skill in the art.

I understand that obviousness also bars the patentability of applying known or obvious design choices to the prior art. One cannot patent merely substituting one prior art element for another if the substitution can be made with predictable results. Likewise, combining prior art techniques that are interoperable with respect to one another is generally obvious and not patentable.

In order for a claim to be found invalid based upon a modification or combination of the prior art, there must be reasonable expectation that a person of ordinary skill would have successfully modified or combined the prior art to arrive at the claimed arrangement. This does not mean that it must be certain that a person of ordinary skill would have been successful – the law only requires that the person of ordinary skill in the art would have perceived a reasonable expectation of success in modifying or combining the prior art to arrive at the claimed invention.

In sum, my understanding is that obviousness invalidates claims that merely recite combinations of, or obvious variations of, prior art teachings using understanding and knowledge of one of skill in the art at the time and motivated by the general problem facing the inventor at the time. Under this analysis, the prior art references themselves, or any need or problem known in the field of endeavor at the time of the invention, can provide a reason for combining the elements of or attempting obvious variations on prior art references in the claimed manner.

Legal Standard for Claim Construction

I understand that before any invalidity analysis can be properly performed, the scope and meaning of the challenged claims must be determined by claim construction.

I understand that a patent may include two types of claims, independent claims and dependent claims. I understand that an independent claim stands alone

and includes only the limitations it recites. I understand that a dependent claim depends from an independent claim or another dependent claim. I understand that a dependent claim includes all the limitations that it recites in addition to the limitations recited in the claim (or claims) from which it depends.

In comparing the challenged claims to the prior art, I have carefully considered the patent and its file history in light of the understanding of a person of skill at the time of the alleged invention.

I understand that to determine how a person of ordinary skill would have understood a claim term, one should look to sources available at the time of the alleged invention that show what a person of skill in the art would have understood disputed claim language to mean. It is my understanding that this may include what is called “intrinsic” evidence as well as “extrinsic” evidence.

I understand that, in construing a claim term, one should primarily rely on intrinsic patent evidence, which includes the words of the claims themselves, the remainder of the patent specification, and the prosecution history. I understand that extrinsic evidence, which is evidence external to the patent and the prosecution history, may also be useful in interpreting patent claims when the intrinsic evidence itself is insufficient. I understand that extrinsic evidence may include principles, concepts, terms, and other resources available to those of skill in the art at the time of the invention.

I understand that words or terms should be given their ordinary and accepted meaning unless it appears that the inventors were using them to mean something else or something more specific. I understand that to determine whether a term has special meaning, the claims, the patent specification, and the prosecution history are particularly important, and may show that the inventor gave a term a particular definition or intentionally disclaimed, disavowed, or surrendered claim scope.

I understand that the claims of a patent define the scope of the rights conferred by the patent. I understand that because the claims point out and distinctly claim the subject matter which the inventors regard as their invention, claim construction analysis must begin with and is focused on the claim language itself. I understand that the context of the term within the claim as well as other claims of the patent can inform the meaning of a claim term. For example, because claim terms are normally used consistently throughout the patent, how a term is used in one claim can often inform the meaning of the same term in other claims. Differences among claims or claim terms can also be a useful guide in understanding the meaning of particular claim terms.

I understand that a claim term should be construed not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the entire specification. I understand that because the

specification is a primary basis for construing the claims, a correct construction must align with the specification.

I understand that the prosecution history of the patent as well as art incorporated by reference or otherwise cited during the prosecution history are also highly relevant in construing claim terms. For instance, art cited by or incorporated by reference may indicate how the inventor and others of skill in the art at the time of the invention understood certain terms and concepts. Additionally, the prosecution history may show that the inventors disclaimed or disavowed claim scope, or further explained the meaning of a claim term.

With regard to extrinsic evidence, I understand that all evidence external to the patent and prosecution history, including expert and inventor testimony, dictionaries, and learned treatises, can also be considered. For example, technical dictionaries may indicate how one of skill in the art used or understood the claim terms. However, I understand that extrinsic evidence is considered to be less reliable than intrinsic evidence, and for that reason is generally given less weight than intrinsic evidence.

I understand that in general, a term or phrase found in the introductory words or preamble of the claim, should be construed as a limitation if it recites essential structure or steps, or is necessary to give meaning to the claim. For instance, I understand preamble language may limit claim scope: (i) if dependence

on a preamble phrase for antecedent basis indicates a reliance on both the preamble and claim body to define the claimed invention; (ii) if reference to the preamble is necessary to understand limitations or terms in the claim body; or (iii) if the preamble recites additional structure or steps that the specification identifies as important.

On the other hand, I understand that a preamble term or phrase is not limiting where a challenged claim defines a structurally complete invention in the claim body and uses the preamble only to state a purpose or intended use for the invention. I understand that to make this determination, one should review the entire patent to gain an understanding of what the inventors claim they invented and intended to encompass in the claims.

I understand that 35 U.S.C. § 112 ¶ 6 created an exception to the general rule of claim construction called a “means plus function” limitation. These types of terms and limitations should be interpreted to cover only the corresponding structure described in the specification, and equivalents thereof. I also understand that a limitation is presumed to be a means plus function limitation if (a) the claim limitation uses the phrase “means for”; (b) the “means for” is modified by functional language; and (c) the phrase “means for” is not modified by sufficient structure for achieving the specified function.

I understand that a structure is considered structurally equivalent to the corresponding structure identified in the specification only if the differences between them are insubstantial. For instance, if the structure performs the same function in substantially the same way to achieve substantially the same result. I further understand that a structural equivalent must have been available at the time of the issuance of the claim.