

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 Owner.

Case No. IPR2025-00486

U.S. Patent 10,873,753

Declaration of Donald Alpert, Ph.D

TABLE OF CONTENTS

I.	OVERVIEW OF THE TECHNOLOGY	6
A.	Scalable Computing Systems	6
B.	Packet Based Networks	9
C.	Memory Hierarchies and Computer Architecture.....	11
II.	THE '753 PATENT	15
A.	Prosecution History	16
B.	Priority Date	18
C.	Challenged Claims	18
D.	Patent Owner's District Court Allegations	19
III.	LEVEL OF ORDINARY SKILL IN THE ART	22
IV.	CLAIM CONSTRUCTION	24
A.	“memory superposition” (claims 4, 7, and 15)	25
B.	“interleaved memory” (claims 4, 7, and 15)	25
C.	“swapped memory over a specified address range” (claims 4, 7, and 15)	27
D.	“memory allocated to specified address ranges according to parameterized task needs” (claims 4, 7, and 15).....	28
V.	INVALIDITY GROUNDS.....	31
A.	Ground 1: Claims 1-3, 13, and 14 Are Rendered Obvious by Stroud in View of DaSilva	31
1.	U.S. Patent Application Publication No. 2013/0346667 A1 (“Stroud”) (Ex-1004)	32
2.	U.S. Patent No. 9,148,389 (“DaSilva”) (Ex-1005).....	36
3.	Independent Claim 1	39

4.	Dependent Claim 2	93
5.	Dependent Claim 3	100
6.	Independent Claim 13	108
7.	Dependent Claim 14	121
B.	Ground 2: Claims 7 and 9-10 Are Rendered Obvious by Stroud in View of Cragon	122
1.	Cragon, “Memory Systems and Pipelined Processors” (“Cragon”) (Ex-1006)	123
2.	Independent Claim 7	125
3.	Dependent Claim 9	147
4.	Dependent Claim 10	149
C.	Ground 3: Claims 7 and 9-10 Are Rendered Obvious by Stroud in View of Cragon and Saulsbury	150
1.	U.S. Patent Application Publication No. 2002/0087821 (“Saulsbury”) (Ex-1007)	151
2.	Independent Claim 7	153
3.	Dependent Claims 9-10.....	166
D.	Ground 4: Claims 4, 8, and 15 Are Rendered Obvious by the Combination of Stroud in View of DaSilva, Cragon, and Saulsbury	166
1.	Dependent Claim 4	168
2.	Dependent Claim 8	168
3.	Dependent Claim 15	169
VI.	BACKGROUND AND QUALIFICATIONS.....	169
A.	Compensation.....	172
B.	Materials and Other Information Considered	172

VII. UNDERSTANDING OF THE LAW	172
VIII. RESERVATION OF RIGHTS	172
APPENDIX 1: CURRICULUM VITAE OF DONALD ALPERT	174
APPENDIX 2: MATERIALS CONSIDERED IN THE PREPARATION OF THIS DECLARATION.....	180
APPENDIX 3: CHALLENGED CLAIMS	182
APPENDIX 4: UNDERSTANDING OF THE LAW	198

I, Donald Alpert declare as follows:

1. My name is Donald Alpert. I am an independent consultant with Camelback Computer Architecture, LLC. 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. 10,873,753 (“the ’753 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.

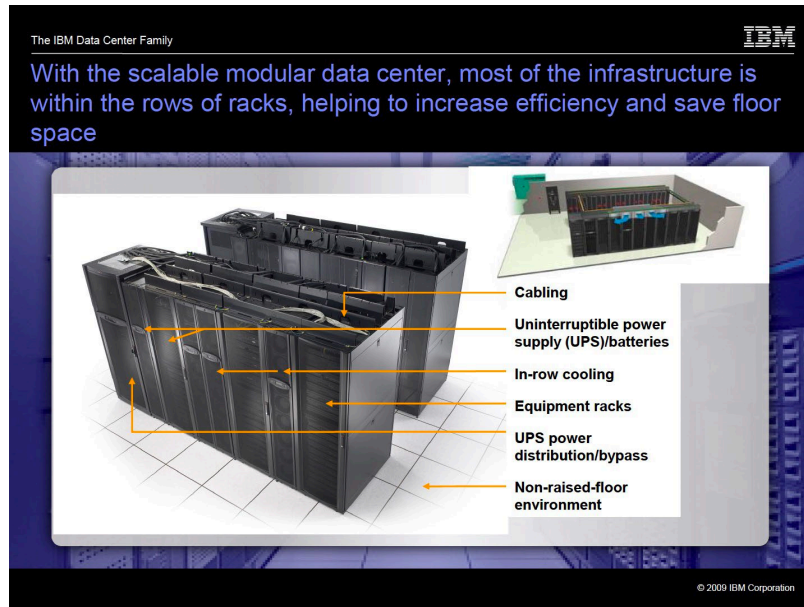
3. As I will explain below, the ’753 patent is directed to a common form factor in data centers: a chassis with swappable cards. However, since this had long been known in the art by 2018, the claims were amended to include particular packet header identifiers and selectable memory arrangements that were originally recited in dependent claims. Even with these additional limitations, the claims are plainly invalid because the prior art teaches the precise identifiers and memory arrangements recited in the ’753 patent. In my following opinions, and to the

extent necessary, I discuss how a person skilled in the art would have been motivated and found it obvious to combine these prior art teachings. Moreover, Patent Owner alleged that the claimed packet header and memory arrangements are satisfied by the use of conventional Ethernet and TCP/IP packets, along with the use of more than one type of memory on a card.

I. OVERVIEW OF THE TECHNOLOGY

A. Scalable Computing Systems

4. Blade servers and chassis platforms were common in data centers by the in the 1990s and 2000s. With the exponential growth of the Internet fueling ever-growing demand for server infrastructure (Ex-1014, 00009), cloud computing and storage emerged as a mainstream service in the 2000s. Associated with the rise of this on-demand computing paradigm was increased performance and storage requirements for data center systems. The following illustration shows an example configuration of equipment in server rooms:



Ex-1014, 00015.

5. Various solutions were developed to maximize computing power as a function of resources, in terms of space, energy, maintenance, and cost. *See, e.g.,* Ex-1014, 00014. The most common approach, going back to the 1990s, was, and still is, to utilize modular platforms that concentrate large numbers of modules (e.g., cards/blades/boards/pods) in an enclosure (e.g., racks/chasses/cabinets) based on component and system standards. *E.g.,* Ex-1015, 000002, Fig. 1:

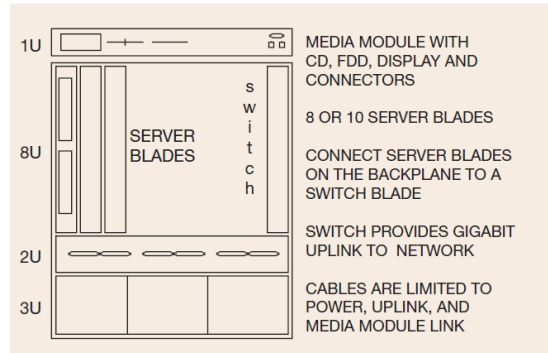


Figure 1

The earliest concept drawing of the BladeCenter architecture, created on October 20, 1999.

It was common to use multiple chassis in such data centers. *E.g.*, Ex-1040, ¶12:

“An enterprise computing system is a data center architecture that integrates computing, networking and storage resources. Enterprise computing systems comprise groups of components or nodes interconnected by a network so as to form an integrated and large scale computing entity. More specifically, an enterprise computing system comprises multiple chassis, commonly referred to as rack or blade server chassis, which include server computers (rack or blade servers) that provide any of a variety of functions. The blade servers in a plurality of blade server chassis are generally interconnected by a plurality of switches.

6. For example, IBM’s BladeCenter architecture provided rows of racks in a chassis. The racks were loaded with multiple blades, each blade being “general- or special-purpose servers that contain processors, memory, optional local...disk drives, Gigabit Ethernet controllers....” Ex-1016, 000001. A mid-plane board provided I/O connections between the blades, allowing the servers to

work in concert and provided a shared connection to the wider network. *See* Ex-1016, 000003-4 (discussing “I/O fabric”).

B. Packet Based Networks

7. Packet-based networks have been the dominant form of computer networking for decades. Packets include a payload containing the data being transmitted and a header portion with information used for routing the packet, such as source and destination addresses.

8. By the 1990s, data centers generally used Ethernet to connect servers, including blade servers, to the network. Servers sent higher-level packets over Ethernet, most commonly using the TCP/IP protocol that had become ubiquitous since the advent of the Internet. These protocols were conceptualized as part of a seven-layer networking model known as the Open Systems Interconnection (OSI) model.

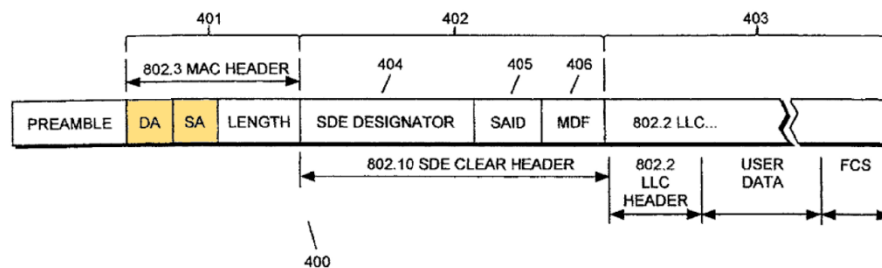
9. Layer 1 of the OSI model was the physical layer. It dealt with the actual physical transmission medium, for example electrical voltages, timing, etc.

10. Layer 2 included the medium access control (“MAC”) layer, which dictated how devices gained access to the medium. Ethernet was the predominant layer 2 protocol for data centers and nearly any network that used wired connections. Servers were connected using Ethernet, and blade servers were no

different. Ethernet devices communicated by sending packets, called frames, with source and destination MAC addresses in the packet header. *See* Ex-1024, Fig. 4:¹

FIG. 4

(PRIOR ART)



11. Layer 3 is the network layer, most commonly the Internet Protocol (“IP”). IP packets included a source and destination IP address in the packet header.

12. Layer 4 was the transport layer. A widely used layer 4 protocol was the Transmission Control Protocol (“TCP”). When two devices communicated over the Internet, they most often formed a TCP connection, which was defined by a “five-tuple,” referring to a set of five fields that identify the particular connection: (1) source IP address, (2) source port, (3) destination IP address, (4) destination port, and (5) transport protocol. *See* Ex-1023, 000018. The

¹ All annotations added unless otherwise noted.

following diagrams depict typical packet formats for IPv4 and IPv6 packets, also including the ethernet frame headers. Ex-1025, Figs. 3A (Ethernet frame header 320, IPv4 header 340, TCP header 360) & 3B (Ethernet frame header 320, IPv6 header 350, TCP header 370):

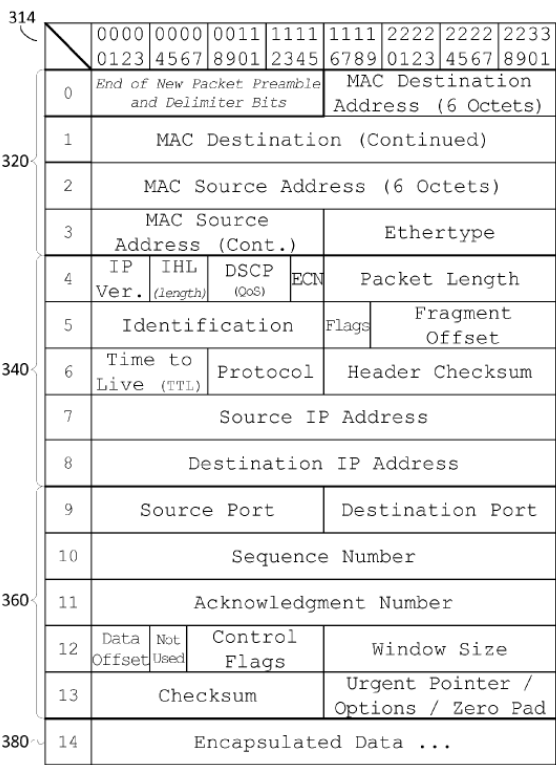


Figure 3A

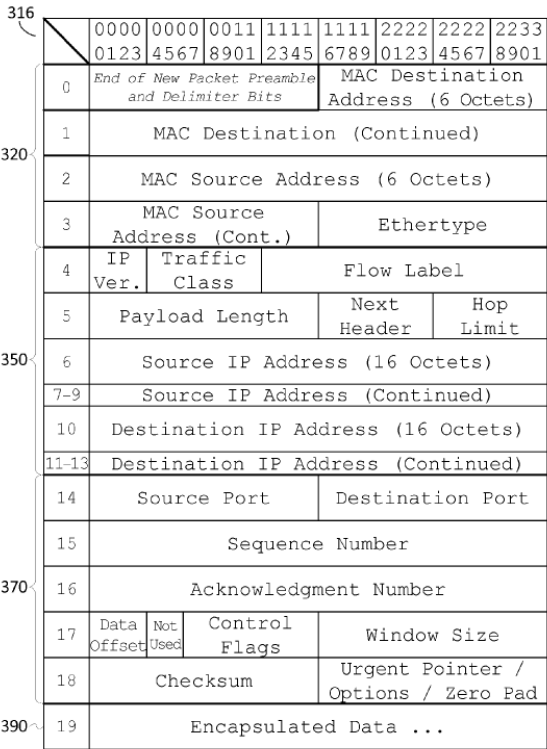


Figure 3B

13. In summary, servers within data centers were connected using Ethernet, and they sent TCP/IP packets over Ethernet. Those packets contained various header fields including destination MAC address.

C. Memory Hierarchies and Computer Architecture

14. Blade servers included various processing elements and memory types. The processing elements included, for example, general purpose central-

processing units (CPUs), application-specific integrated circuits (ASICs), and programmable logic devices (PLDs).

15. PLDs represented a class of integrated circuit (“IC”) devices that provide logic circuit area that can be configured after manufacturing, in contrast to general-purpose CPUs and ASICs with logic circuits that are fixed when the IC is manufactured. PLDs allowed users to define the function of hardware circuits in the IC by specifying the logical interconnections of those circuits, thereby creating custom hardware to perform custom functions. Ex-1010, 000004, 000009-11; Ex-1008, 1:30-37; Ex-1011, Fig. 4. Popular types of PLDs included Field Programmable Gate Arrays (“FPGAs”). Ex-1010, 000002-3, Abstract; Ex-1017, 000087; Ex-1018, §1, §3; Ex-1019, 000001. PLDs were also considered a type of configurable logic device (CLD). Ex-1004, ¶107.

16. Computers used different types of memories for different purposes. Like most computers, blade servers used multiple types of memory, including for example RAM, non-volatile storage, and other memories.

17. For many decades, these memory types have been organized into a pyramid, called a memory hierarchy, with faster but smaller memory (typically more expensive for unit price) at the top and larger but slower memory (typically cheaper for unit price) at the bottom. SRAM was near the top of the pyramid. SRAM was used for cache memory or applications that required low latency, but it

was limited in size. DRAM was lower on the hierarchy; it was often used as main memory. Flash memory, also referred to as solid-state disk (“SSD”), was used for large, non-volatile storage. Hard Disks were typically at the bottom of the memory hierarchy because it was slow, with high latency, but provided large capacity for storage.

18. Memory was often used in selectable arrangements. For example, even by the 1980s, it was known for memory to be selectable to be arranged either as cache memory or boot memory: “a memory coupled to the processing unit, and an arrangement coupled to the memory for selectively causing the memory...to selectively serve as either a boot memory or a cache memory for the processing unit.” Ex-1026, 1:48-58. By the 1990s, it was known for processors to have “on-chip memory having standard and cache operating modes.” Ex-1027, [54] Title, Abstract, Fig. 4. The concept of selectable arrangement was known and applied to various types of memory and to select a variety of configurations. *E.g.*, Ex-1028, (54) Title (“Multi-level system memory having near memory space capable of behaving as near memory cache or fast addressable system memory depending on system state”), ¶5, Fig. 2; Ex-1029, Abstract (“reconfiguration control circuitry to switch between a first mode that uses the second cache memory as a cache memory in a lower layer than the first cache memory and a second mode that uses the first cache memory and the second cache memory as cache memories in an

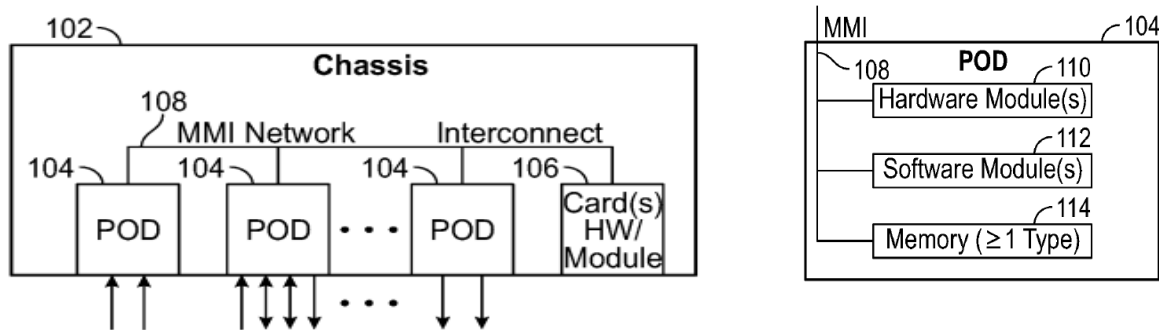
identical memory layer”), Fig. 6; Ex-1030, Abstract (“[A] storage system uses a page...as a cache area.”), ¶8 (“using a portion of an area of a flash memory device as a caching memory for data, which is stored in a disk device, and for using another area in this flash memory device as an area for permanently storing data”).

19. Selectable memory arrangements were common because memory was a limited resource, and each mode of operation had known benefits. For example, cache memory was useful when a set of data was repeatedly accessed because it provided fast access to a small amount of data. By allowing the same memory to be used as cache in some circumstances but standard memory in other situations, the prior art provided flexibility and optimized memory performance for a wide variety of potential use cases. Ex-1030, ¶83 (“[T]he cache capacity can be expanded relatively easily by dynamically allocating pages to the cache volume for the purpose of enhancing the hit ratio. Alternatively, in a case where the hit ratio is not improved much even though the cache capacity has been increased, the cache capacity can be decreased relatively easily by releasing a page from the cache volume.”)

20. Some of the most common memory arrangements included multilevel cache, interleaved memory, swap files, and memory allocated to specified address ranges. These arrangements were basic concepts taught in textbooks. *E.g.*, Ex-1006.

II. THE '753 PATENT

21. The '753 patent is directed to a “platform for data flow processing” with “one or more swappable pods or cards in one or more chassis, coupled through a module messaging interface network.” Ex-1001, Abstract. The pods/cards contain software modules and hardware modules (such as FPGAs or PLDs). Ex-1001, Figs. 1A, 2:



22. The messaging interface network can be over “PCIe, *Ethernet*, direct serial or parallel connection, or any other suitable connection.” Ex-1001, 3:49-46. The purportedly novel feature of the '753 patent is the use of several identifiers in the packet header: “[a] header on Module Message Interface packets specifies *chassis, board, module, instance and type identifiers*, so that packets in the MMI network are routed to a hardware or software module in a pod or card in the chassis or one of multiple chassis.” Ex-1001, 3:4-7, Figs. 6A, 6B-1, 6B-2, 6B-3:

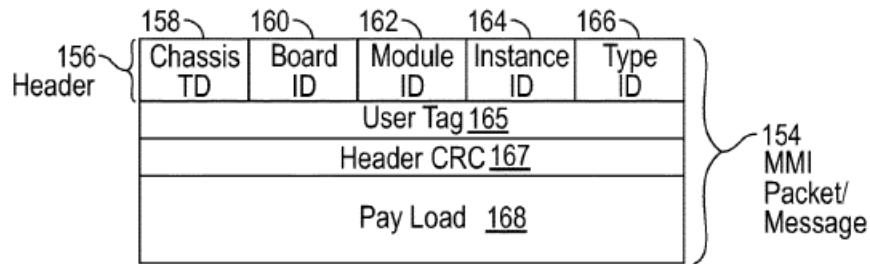


FIG. 6A

23. The '753 patent also refers to something it calls “memory superposition” where “two or more types of memory, such as SRAM (static random access memory) 144, DRAM (dynamic random access memory) 146, FRAM (ferroelectric random-access memory) 148, RLDRAM (reduced latency dynamic random access memory) 150, flash 152 (electrically erasable programmable read-only memory or EEPROM), 3D Xpoint, etc., are made available for a data flow or other data processing operation.” Ex-1001, 7:1-15.

24. However, as explained above, it was common for blade server chassis to include swappable cards or pods connected through a backplane using Ethernet (*Supra* §I.A), and the use of the recited packet header identifiers was taught by Stroud and DaSilva. Moreover, it was common for blades to include multiple types of memory for data flow and other data processing operations, given the network context in which they were used. *Supra* §I.C.

A. Prosecution History

25. During prosecution, the '753 patent faced an Office Action rejection based on a combination of prior art. Ex-1002, 000083-92. Particularly, the Office

rejected then-pending claims 1, 3-4, 8, 10, 11, 15, and 17 under 35 U.S.C. 103 as being unpatentable over U.S. Patent No. 4,890,832 (“Komaki”) in view of U.S. Publication No. 2008/0016505 (“Bucklew”). Ex-1002, 000086. The Office also rejected claims 6, 13, and 19 under 35 U.S.C. 103 as being unpatentable over Komaki in view of Bucklew and further in view of U.S. Patent No. 7,353,448 (“Barash”). Ex-1002, 000088.

26. However, the Office indicated that as-filed dependent claims 2, 9, and 16 were allowable because “the closest prior art of record, either alone or in combination with other prior art of record, does not teach, suggest, or disclose packets each having *a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.*” Ex-1002, 000090. The Office also indicated that as-filed dependent claims 5, 12, and 18 are allowable because “prior art of record, does not teach, suggest, or disclose at least one of the one or more swappable pods or cards having a plurality of types of memory and memory superposition with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.” Ex-1002, 000091.

27. I understand that the Applicant simply incorporated the subject matter indicated as allowable by the Examiner into the independent claims, thereby conceding that the remaining elements of the independent claims are taught by the prior art references cited by the Office. Ex-1002, 000107-114. Applicant did not argue against the prior art references presented by the Examiner. Ex-1002, 000115. However, as I have explained below, the purportedly novel features of the claims are taught by the prior art in Grounds 1-3. *Infra* §V.A.3.[1e], §V.B.2.[7d].

B. Priority Date

28. The '753 patent was filed on July 15, 2018. For purposes of this Declaration, I have analyzed obviousness as of July 15, 2018. I do not offer an opinion as to whether the '753 patent is entitled to a certain priority date. My invalidity opinions would not change if a later date was determined to be the correct priority date because the prior art relied upon in this declaration would still be prior art.

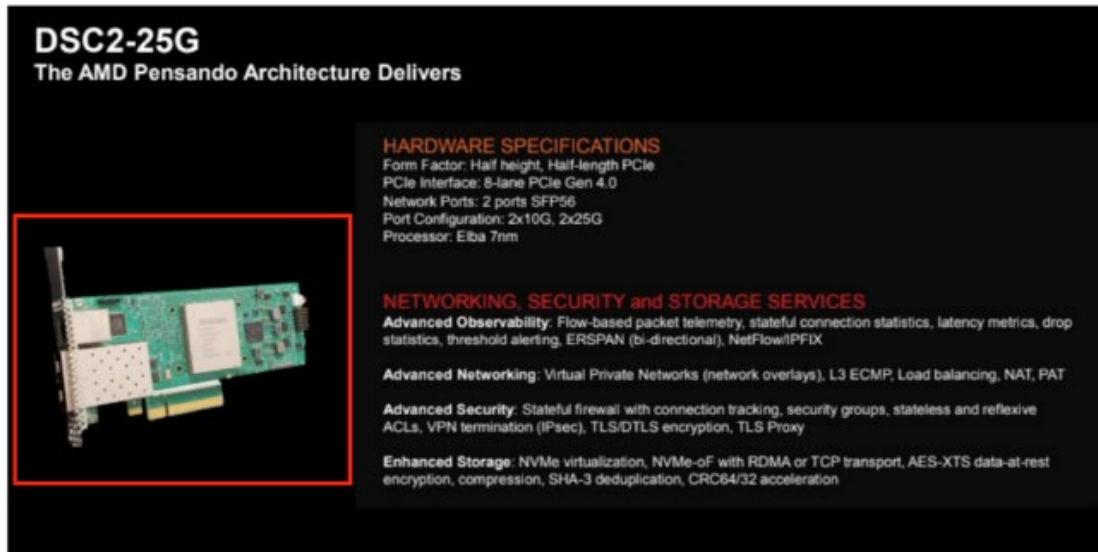
C. Challenged Claims

29. I understand that Petitioners are challenging the validity of the Challenged Claims (listed in the chart at §V) in the Petition for *Inter Partes Review* to which this declaration will be attached. Those claims are reproduced in Appendix 3. While this Petition and declaration are directed to the Challenged

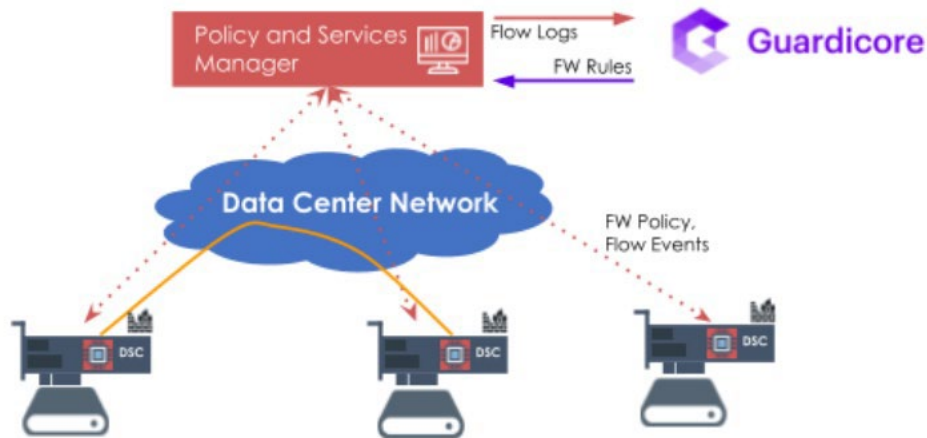
Claims, I have considered all claims 1-17 of the '753 patent, as well as portions of the '753 patent prosecution history in forming my opinions.

D. Patent Owner's District Court Allegations

30. In district court, Patent Owner asserted the claims broadly, alleging that the '753 patent is satisfied by a network card based on its support for conventional TCP/IP packets and Ethernet. Patent Owner accused Pensando networking cards of infringing the '753 patent. Ex-1035, ¶¶51:



31. Patent Owner's infringement contentions allege that networking cards infringe the '753 patent because they are swappable cards coupled to an Ethernet network. Ex-1021, 000004:



32. According to Patent Owner, the claimed packet header identifiers (“packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier”) are satisfied by traditional TCP/IP packet headers, with reference to the “*five-tuple*” that defines a TCP/IP connection along with extended information such as Ethernet “*MAC addresses*.” Ex-1042, 000021. This allegation was repeated in Patent Owner’s infringement contentions, which again pointed to the TCP/IP five-tuple with IP addresses, ports, and extended information including the Ethernet “Destination MAC” address. Ex-1021, 000005:

Table: Packet 1 - Redirect Action - Test Packet Header Information

Receiving Port	Destination MAC	Destination IP	Destination Port	Source IP	Source Port	Protocol
Eth 1/2	00:00:00:00:10:02	1.2.3.4	456	2.3.4.5	345	UDP

33. In supplemental contentions, Patent Owner reiterated the allegation that a “MAC address” and “IP” address are “examples of ‘chassis identifier[s]’ and

‘board identifiers’” while expanding the allegations to encompass a wide range of other fields as potential chassis and board identifiers (e.g., “VLAN/VNIC ID”). Ex-1041, 000005. Patent Owner also broadly interpreted the claims such that the recited packet header identifiers can be satisfied by any data in a packet, even if not part of the packet header, or data used within a chip, even if not part of a packet at all. *Id.*, 000005, 000006 (pointing to “devicename,” “serialnumber,” “unitid,” “P4 to Arm-CPU header” which are allegedly part of a data structure called a packet header *vector* that is not part of a packet, and also payload data used for IEEE “Link Layer Discovery Protocol (LLDP)”).

34. Regarding the memory superposition described in claims 4, 7 and 15, Patent Owner alleged the entire claim limitation (“memory superposition with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs”) was satisfied by the presence of multiple types of memory on the network card, such as SRAM and DRAM. According to Patent Owner, the accused products satisfy claim 4 and limitation [7d] because “[t]he Accused Products support/handle various types of memory to optimize performance (e.g., various memory types like SRAM, TCAM, DRAM memory cache, DDR4/5).” Ex-1021, 000006; Ex-1041, 000009-11.

III. LEVEL OF ORDINARY SKILL IN THE ART

35. I have analyzed the '753 patent and determined that the field of the patent is data processing and streaming architectures including user-defined hardware and software modules. The specification discusses the need to improve prior art systems “for many data flow and data processing applications.” Ex-1001, 1:6-21. The specification further explains that it provides “a variety of flexible features for hardware-defined and software-defined functionality in data streaming and data processing systems, including video.” Ex-1001, 2:47-3:20. The '753 patent summarizes its alleged invention as “[a] platform for data flow processing...having...user-definable hardware modules or user-definable software modules.” Ex-1001, Abstract. The '753 patent is titled “Hardware defined anything in a platform with swappable pods, message interface, sandboxes and memory superposition” which I understand is a summary of the techniques reflected in the patent’s claims and specification to configure data flow processing architectures.

36. In determining the characteristics of a hypothetical person of ordinary skill in the art (“POSITA”) of the '753 patent at the time of the claimed invention, I considered several things, including various prior art techniques relating to data processing (such as data processing/streaming architectures for servers with a variety of flexible hardware-defined and software-defined features), 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 '753 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 data processing.

37. I came to the conclusion that a person of ordinary skill in the field of art of the '753 patent would have been a person with a bachelor's degree in electrical engineering, computer engineering, or a similar major with at least two years of experience in the field (described above) or a person with a master's degree in electrical engineering, computer engineering, or a similar major with a specialization in the field. A person with less education but more relevant practical experience may also meet this standard.

38. In the 2018 timeframe, a POSITA would have known and had the skills necessary to:

- Design, analyze, evaluate, and modify data processing and network systems, including but not limited to implementing the open systems interconnection (OSI) model;

- Design computer systems and networks that support common networking protocols, including Ethernet, IP, and TCP/IP protocols;
- Utilize the Ethernet protocol and modify Ethernet packets;
- Utilize IP protocol and modify IP packets;
- Use and implement known network traffic management principles, including but not limited to routing network packets based on a medium access control (MAC) address;
- Program CPUs and network processors for data flow and data processing applications;
- Utilize common memory arrangements and memory hierarchies, including multilevel cache, interleaved memory, swap files, and memory allocations;
- Implement known memory arrangements in known computer platforms;
- Modify existing CPU designs to implement known memory architectures;
- Design printed circuit board cards; and
- Implement CLD, PLD, and FPGA designs.

IV. CLAIM CONSTRUCTION

39. 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.

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

A. “memory superposition” (claims 4, 7, and 15)

41. In the parallel district court proceeding, Patent Owner has proposed the construction for this term as “two or more types of memory that can be made available for a data flow or other data processing operation.” Ex-1036, 000003. It is my understanding that Petitioners have adopted this construction for the purposes of this Petition. Therefore, I have applied this construction to my analysis of the prior art.

B. “interleaved memory” (claims 4, 7, and 15)

42. The prior art teaches interleaved memory irrespective of whether the below construction is adopted.

43. Based on my review of the claims and specification of the ’753 patent, it is my opinion that a POSITA would have understood that “interleaved memory” means an arrangement of different types of memory that are alternately accessed to achieve an average access bandwidth.

44. The claims explain how “interleaved memory” is one of the selectable arrangements of “the plurality of types of memory” in a “memory superposition.” See Ex-1001, Claims 4, 7, and 15. The specification states that, “[f]or memory superposition...*two or more types of memory* are made available for a data flow or other data processing operation.” Ex-1001, 7:1-9. In its only discussion of interleaved memory, the specification explains that “[a]nother arrangement of memory superposition 140 is *interleaved memory, to achieve an average access bandwidth*. Input bandwidth, output bandwidth, or power could be blended among *different types of RAM or drives* such as rotational or disk drives versus solid-state drives, storage media modules, and different drive interfaces.” *Id.*, 8:35-40. By alternating access between different types of memory, this arrangement achieves an access bandwidth that is a blended average of each type of memory. *Id.* For example, when DRAM and solid-state drives are used as interleaved memory, the resulting access bandwidth would be the average between the bandwidth of DRAM and that of the solid-state drive. *See id.*

45. Therefore, the claims and specification are clear: “interleaved memory” as used in the ’753 patent refers to an arrangement of different types of memory that are alternately accessed to achieve an average access bandwidth.

C. “swapped memory over a specified address range” (claims 4, 7, and 15)

46. The prior art teaches this claim term irrespective of whether the below construction is adopted.

47. Based on my review of the claims and specification of the '753 patent, it is my opinion that a POSITA would have understood that “swapped memory over a specified address range” means memory with a specified address range where information is moved between the memory and non-volatile storage.

48. The claims of the '753 patent recite this term without further guidance as to its meaning. *See* Ex-1001, Claims 4, 7, and 15. The written description is likewise brief, simply stating that “swapped memory over a specified address range” is “another arrangement of memory superposition.” Ex-1001, 8:41-42; *see also id.*, 14:25-31 (reciting claim language). Because it is an arrangement of memory superposition, the term at least involves the use of “two or more types of memory” (Ex-1001, 7:1-9), but the specification does not provide any additional detail.

49. Swapped memory is a term of art that a POSITA would have recognized in the context of computer memory. Therefore, extrinsic evidence provides guidance. The Microsoft Computing Dictionary explains that swapping “involves moving blocks of information in units known as pages between memory and disk as they are needed during the execution of the application.” Ex-1038,

000003. This technique was used in connection with a “swap file” on non-volatile storage. Ex-1039, 000003 (“swap file *n (comput)* an area of the hard disk of a computer into which any information not immediately required may be moved, thus maximizing the available memory.”).

50. These definitions were consistent with the conventional meaning of “swapped memory” and the ’753 patent, which as explained above states that swapped memory involves the use of two or more types of memory. Therefore, a POSITA with experience in the field would have recognized “swapped memory over a specified address range” as memory with a specified address range where information is moved between the memory and non-volatile storage, such as disk.

D. “memory allocated to specified address ranges according to parameterized task needs” (claims 4, 7, and 15)

51. The prior art teaches this claim term irrespective of whether the below construction is adopted.

52. Based on my review of the claims and specification of the ’753 patent, it is my opinion that a POSITA would have understood that “memory allocated to specified address ranges according to parameterized task needs” means different types of memory allocated to specified address ranges according to parameters for task needs.

53. The claims explain how this term refers to one of the selectable arrangements of “the plurality of types of memory” in a “memory superposition.”

See Ex-1001, Claims 4 & 7. The specification states, “Memory...superposition 140 offers multiple types of memories...that are in a sense virtualized and can be assigned or *allocated to the same or differing address ranges according to bandwidth, density, latency or other readily parameterized task needs*, and reconfigured for subsequent tasks.” Ex-1004, 8:22-31, 7:1-9. Therefore, the specification and claims confirm that this claim term requires different types of memory allocated to specified address ranges according to parameters for task needs.

54. The specification provides several examples of different types of memory having different bandwidth, density, and latency characteristics:

Each type of memory has an optimal use. SRAM is expensive to produce per bit, thus densities are low. However SRAM is fast and pays no penalty (e.g., latency, delay) for random-access reads nor writes. By contrast, flash is fast on read but slower on write, and less expensive per bit than SRAM and so has higher densities. With DRAM there is a penalty to be paid for random-access, as a result of page open and close requirements. In addition DRAM must be refreshed periodically, which affects usable bandwidth. However, DRAM is fast and inexpensive per bit compared to SRAM, and is thus available in much higher densities at relatively low cost (e.g., DDR3 (double data rate type three synchronous DRAM) memory in personal computers), further reducing costs due to high volume. RDRAM is between SRAM and DRAM in density, speed and cost. RDRAM has a higher penalty for random-

access than SRAM, but a lower penalty than DRAM, and cost per bit is lower than with SRAM but higher than for DRAM. Thus available densities for RLDRAM are larger than SRAM but lower than DRAM.

Ex-1001, 7:21-56.

55. The specification then provides examples of parameters for task needs. For the task of identifying and classifying network traffic, “it is important to have deterministic and guaranteed memory access times, as well as random-access since traffic can come in any order.” Ex-1001, 7:57-66. Based on these parameters (deterministic and guaranteed access times), “SRAM is well-suited for this use.” *Id.* For the subsequent task of maintaining tables and metadata for classified traffic, “there is a need for more memory capacity to store metadata, while still maintaining high random-access.” *Id.*, 7:66-8:5. Based on these parameters (high capacity and high random-access), “RLDRAM is well-suited to this use.” *Id.* For later tasks of digital video encoding, “even more memory” is needed, while “random-access can be significantly reduced due to sequencing and prioritization.” *Id.*, 8:5-21. Based on these parameters, “DRAM is well-suited to this need for the highest memory capacity and potentially reduced random-access.” *Id.*

56. Each of these examples, along with the rest of the intrinsic record, confirm that this term requires different types of memory allocated to specified

address ranges according to parameters for task needs, such as parameters for bandwidth, density, or latency.

V. INVALIDITY GROUNDS

57. Based on my review and analysis of the materials cited herein, my opinions regarding the understanding of a POSITA in the relevant timeframe (*supra* § II.C), and my training and experience, it is my opinion that the challenged claims of the '753 patent are invalid based on the following grounds:

Grounds	Challenged Claims	Statutory Basis	Prior Art
1	1-3, 13, 14	§103	Stroud in view of DaSilva
2	7, 9-10	§103	Stroud in view of Cragon
3	7, 9-10	§103	Stroud in view of Cragon and Saulsbury
4	4, 8, 15	§103	Stroud in view of DaSilva, Cragon, and Saulsbury

The reasons for my conclusions are explained more fully below.

A. Ground 1: Claims 1-3, 13, and 14 Are Rendered Obvious by Stroud in View of DaSilva

58. It is my opinion that a POSITA would have found the Challenged Claims 1-3, 13, and 14 obvious based on the teachings of Stroud and DaSilva. Stroud teaches a network testing platform that comprises a chassis with multiple cards. Ex-1004, Abstract, Fig. 3. The cards communicate through management switches and a backplane using packets, e.g., over ethernet. Ex-1004, ¶¶99, ¶101, ¶113. The packet header includes various fields to indicate, among other things, a

particular card slot and the particular type and instance of processor or CLD on the card. Ex-1004, ¶367-372.

59. DaSilva teaches an improvement for a network with multiple chassis, where each is assigned a chassis ID, and packet headers include a chassis ID to indicate a particular chassis. Ex-1005, 4:23-33, 4:53-65, 5:13-24. Since it was common to use multiple chassis in a networked environment, for example, in data centers (*see, e.g.*, Ex-1040, ¶2 (“An enterprise computing system is a data center architecture that integrates computing, networking and storage resources....[A]n enterprise computing system comprises multiple chassis, commonly referred to as rack or blade server chassis, which include server computers (rack or blade servers) that provide any of a variety of functions.”)), a POSITA would have been motivated to combine Stroud’s teachings with DaSilva’s by using a chassis ID in the packet header to allow multiple chassis to communicate with each other. The motivation to combine and reasonable expectation of success are explained in more detail below. *Infra* §V.A.3.[1e] (Motivation to Combine subsection).

1. U.S. Patent Application Publication No. 2013/0346667 A1 (“Stroud”) (Ex-1004)

60. I have reviewed the Stroud reference. I understand that Stroud was not cited or considered during the prosecution of the ’753 patent, based primarily on the fact that Stroud is not cited on the face of the patent. I also have not seen the reference discussed in the prosecution history.

61. Stroud is entitled “Serial port access system and method in a computing system.” Ex-1004, (54) Title. Stroud was published December 26, 2013. Ex-1004, (43) Pub. Date. Based on the priority date of July 15, 2018 of the ’753 patent, I understand Stroud is prior art to the ’753 patent under at post-AIA 35 U.S.C. §102(a)(1).

62. Stroud teaches a modular platform for testing and simulating various portions of a network. Ex-1004, ¶¶91-92, Figs. 2D-2E. The platform includes a chassis with “modular” cards; each card includes multiple configurable logic devices (such as FPGAs) and network processors. Ex-1004, ¶¶98-107, Fig. 3:

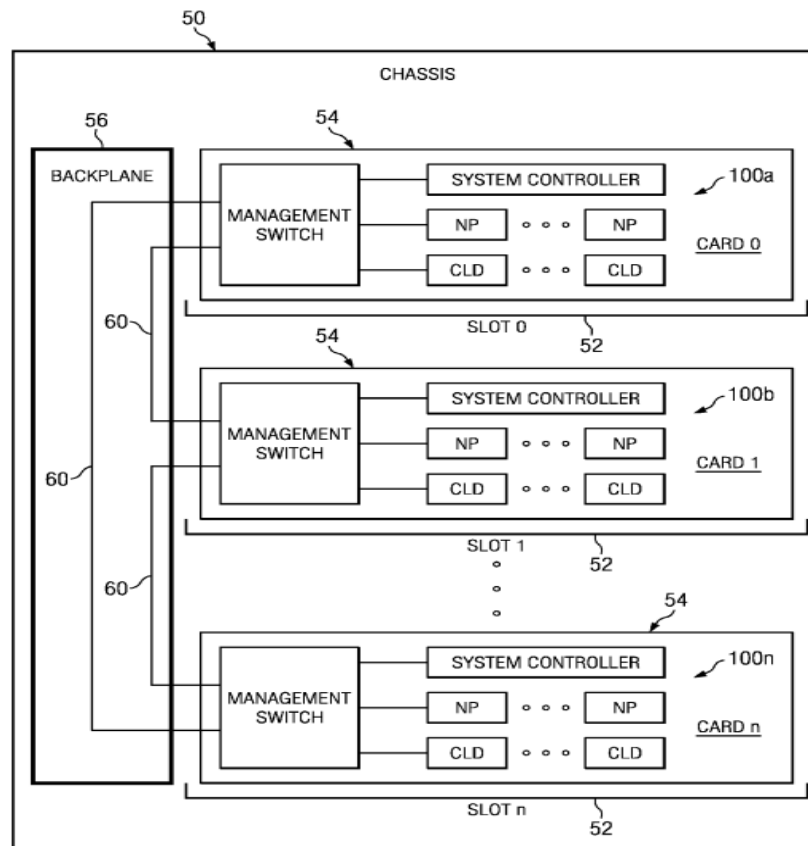


FIG. 3

More than one such chassis can be used together to compose the testing platform.

Ex-1004, ¶343 (“The hardware devices, software, CLD configuration information, and/or firmware of each respective sub-system may be embodied in a single device of system 16, or distributed across multiple devices of 16, as appropriate.”).

63. The cards communicate over an Ethernet backplane interface network.

Ex-1004, ¶99, ¶101. To route packets to the correct destination, Stroud teaches subfields within the MAC address that identify for each processing element, the device type (“chip type”), module type (“processor type or model”), board (slot number), instance of a module (“processor or CLD number”), and the interface to the module. Ex-1004, ¶¶366-372.

64. I note that Stroud uses several examples to explain different aspects of its network testing platform. The teachings are meant to be used together, as Stroud presents these examples as part of the same overall embodiment/platform. E.g., Ex-1004, ¶85 (“These systems may be configured to operate...in concert to provide a full-spectrum solution for testing and Verifying network performance, application and security traffic scenarios.”), ¶86 (“[T]he architecture of network testing system 16 may allow for some or all of subsystems 20-28 to operate simultaneously and cooperatively within the same software and hardware platform.”), ¶79 (“[P]referred embodiments and their advantages over the prior art are best understood by reference to FIGS. 1-70.”). For example, Stroud teaches “a

chassis 50 including any suitable number of slots” to “receive a modular card, or blade.” Ex-1004, ¶98. Various teachings relate to the same modular chassis platform, and Stroud’s interrelated figures describe common components in the chassis, such as CLD 102 and switch 110 in Figures 4-9A, and throughout the disclosure. Various portions of Stroud explain how the cards operate within a chassis (e.g., *id.*, ¶¶66, ¶98, ¶176, ¶376) and how the chassis operates within a network (e.g., *id.*, ¶¶87-97). Stroud explains that the subsystems of the platform are distributable across different cards and chassis. Ex-1004, ¶343 (“The hardware devices, software, CLD configuration information, and/or firmware of each respective sub-system may be...distributed across multiple [chassis] devices of 16, as appropriate.”), ¶365. Several exemplary card architectures are presented (Ex-1004, ¶¶100-101; e.g., *id.*, 6A, 7A, 8A, 9A), which work together in the same chassis. E-1004, ¶176. Therefore, a POSITA looking to implement Stroud’s platform would naturally have looked at all of Stroud’s teachings, which describe Stroud’s chassis architecture and its usage in networks in data centers.

65. Using Stroud’s teachings together would have had several advantages. For example, Stroud’s combined teachings would have provided a POSITA with details for implementing a multi-card chassis architecture for network processing and simulation, with details and features regarding the configuration of chassis systems and the packet format and headers for platforms having multiple cards.

Ex-1004, ¶¶98-102, ¶322-342, Figs. 3, 54, 55A, 55B, 56A. The application of Stroud’s teachings would have provided the benefits Stroud describes: the ability to test new types of network equipment with “realistic, randomized traffic testing at real-time speeds.” Ex-1004, ¶5. Therefore, in summary, a POSITA would have had a motivation to combine Stroud’s teachings and reasonable expectation of success in doing so.

66. Stroud is analogous art because it is in the same field of endeavor as the ’753 patent, which is data processing and streaming architectures including user-defined hardware and software modules. Ex-1001, Abstract. Stroud is directed to a network simulation platform that generates a stream of network traffic data and processes network data. Ex-1004, ¶86, ¶298. Stroud is reasonably pertinent to the problem addressed by the ’753 patent (Ex-1001, 1:6-44) because Stroud presents an architecture for processing data flows using a similar approach that combines hardware modules (e.g., FPGAs) and software modules. Ex-1004, ¶86 (“system 16 comprises custom hardware and software arranged and programmed to deliver performance and measurement abilities not achievable with conventional software or hardware solutions.”), ¶101, ¶¶106-113, ¶144, ¶373.

2. U.S. Patent No. 9,148,389 (“DaSilva”) (Ex-1005)

67. I have reviewed the DaSilva reference. I understand that DaSilva was not cited or considered during prosecution of the ’753 patent, based primarily on

the fact that DaSilva is not cited on the face of the patent. I also have not seen the reference discussed in the prosecution history.

68. DaSilva is entitled “System and method for a virtual chassis system.” Ex-1005, (54) Title. DaSilva was filed on November 12, 2012 and issued on September 29, 2015. Ex-1004, (22) Filed, (45) Date of Patent. Based on the priority date of July 15, 2018 of the ’753 patent, I understand DaSilva is prior art to the ’753 patent under post-AIA 35 U.S.C. §§102(a)(1) and (2).

69. DaSilva discloses chassis with multiple cards. To distinguish between multiple chassis, DaSilva teaches the use of a chassis ID. Ex-1005, Abstract, 1:40-42, 4:51-54, 9:1-6, Fig. 4:

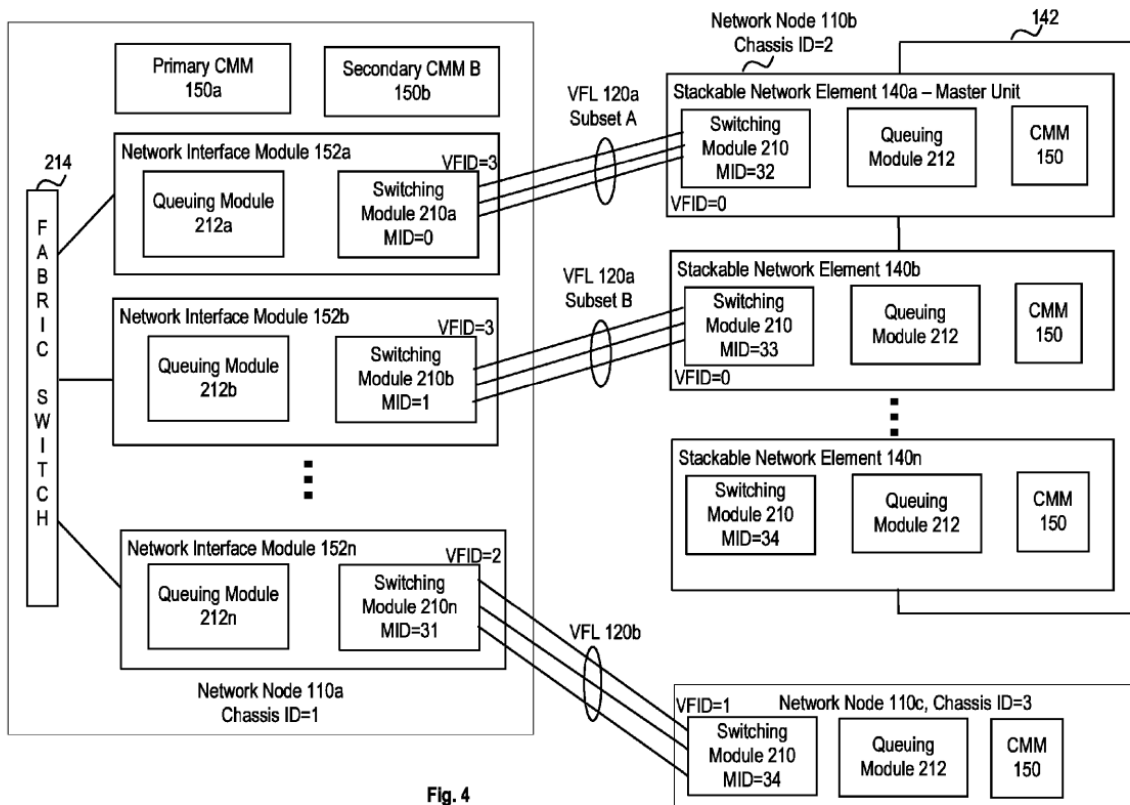


Fig. 4

70. DaSilva is analogous art because it is in the same field of endeavor as the '753 patent, which is data processing and streaming architectures including user-defined hardware and software modules. Ex-1001, Abstract. For example, DaSilva is directed to architectures “for providing topological redundancy and resiliency between nodes of one or more data networks.” Ex-1005, 1:40-43. DaSilva’s architecture allows for the flexible configuration of packet processing and routing nodes in a network, including for data streaming flows such as “Voice over-IP (VoIP), data and video applications.”. Ex-1005, 1:45-2:26. DaSilva is reasonably pertinent to the problem addressed by the '753 patent (Ex-1001, 1:6-44)

because DaSilva recognizes “it is sometimes difficult to upgrade or transition” processing elements used in network data processing architectures. Ex-1005, 2:9-22.

3. Independent Claim 1

[1pre]. A platform for data flow processing, comprising:

71. I understand that a preamble generally does not state a claim limitation. However, to the extent that Patent Owner argues that the preamble states a limitation, it is my opinion that Stroud teaches the preamble.

72. For example, Stroud teaches “a hardware- and software-based testing and simulation platform” (Ex-1004, ¶¶85-87) that processes “data flows” in a network (Ex-1004, ¶298). Ex-1004, ¶¶80-81, Fig. 1:

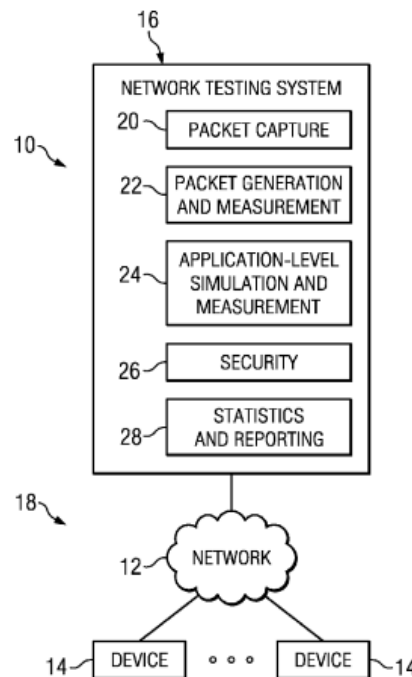


FIG. 1

The network testing platform processes data flows by, for example, simulating network traffic to test the “traffic-handling performance” and “security” of devices in a network. Ex-1004, ¶¶80-81. “In some embodiments, network testing system 16 is configured to model and simulate network traffic.” Ex-1004, ¶84. The platform also processes data flows when “synthetic data flows [are] generated by system 16” (Ex-1004, ¶298), as generating synthetic data flows requires the processing of data flows. Generated data flows are then used to “drive the operation of a test system” that is configured to run specific data processing applications (“e.g., database server, Web server,” etc.). Ex-1004, ¶84, ¶¶89-93. The platform carries out various processing on the generated packet data flowing through the network, such as “generat[ing] and analyz[ing] packets at line rate, while simultaneously capturing that same traffic, performing application simulation, and security testing.” Ex-1004, ¶86. Stroud also teaches an application with deep packet inspection for “analyz[ing] traffic beyond individual packets” and “instead look[ing] at application flows.” *Id.*, ¶219. Stroud teaches various details regarding for exemplary data flow processing using the network testing platform. *Id.*, ¶¶220-250.

73. The platform further comprises the elements recited by subsequent limitations of claim 1, as I explain below.

74. For at least these reasons, [1pre] is obvious in view of Ground 1.

[1a] one or more swappable pods or cards in one or more chassis, coupled through a messaging interface network;

75. Stroud teaches limitation [1a]. Stroud teaches **one or more swappable pods or cards in one or more chassis**. For example, Stroud teaches “a chassis 50 including any suitable number of slots 52, each configured to receive a modular card, or blade, 54.”² Ex-1004, ¶98, ¶365 (“a chassis 50 with multiple slots 52, and each containing a blade 54”), ¶376 (“chassis 50 that interconnects multiple cards 54”), 377 (“the cards 54 in chassis 50”), Fig. 3:

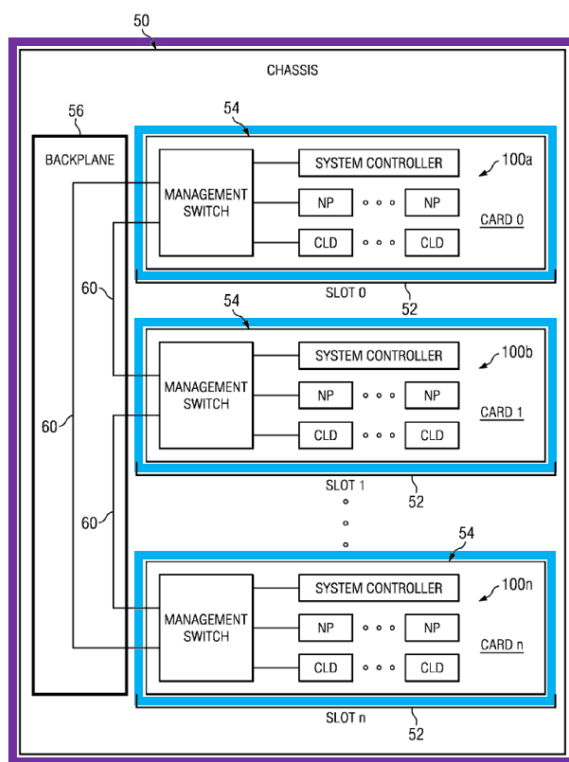


FIG. 3

² All annotations/emphasis added unless otherwise noted.

76. Stroud provides an example in figure 54 that “illustrates a three-dimensional view of an example network testing system having three blades installed in a chassis.” Ex-1004, ¶66, Fig. 54:

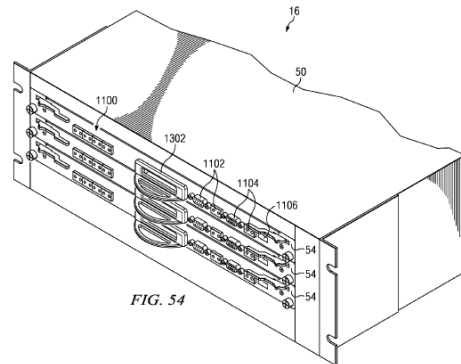
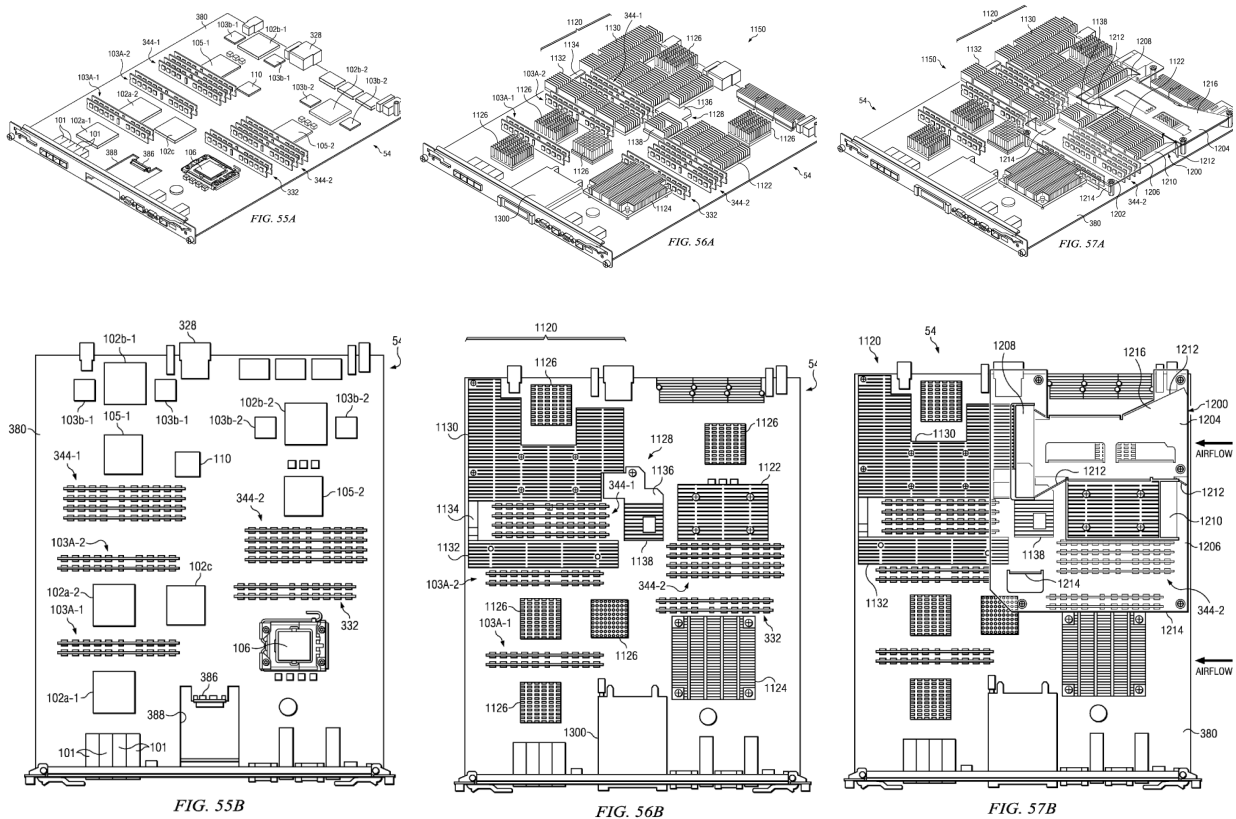


Figure 54 depicts “a chassis 50 having three slots 52 configured to receive three cards 54.” Ex-1004, ¶714.

77. For simplicity, I will refer to Stroud’s “cards” in this Declaration, but Stroud’s cards are blades that satisfy the claimed “swappable cards” and “swappable pods.” When explaining its teachings, Stroud uses the terms “card” and “blade” interchangeably. *E.g.*, Ex-1004, ¶¶98, ¶13 (“architecture of a card or blade”), ¶33 (“boot process for a card or blade”), ¶176 (“any suitable number of cards or blades”). Both “[a] card or blade 54 may comprise one or more printed circuit boards (e.g., PCB 380 discussed below).” *Id.*, ¶98. Thus, each blade comprises a large card. *Id.*, ¶115, ¶176, ¶376, ¶382, Figs. 54, 55A-55B, 56A-56B, 57A-57B:

Card/Blade



78. Stroud explains that the modular cards/blades are swappable. For example, Stroud discloses that the cards “can...be hot-swapped.” Ex-1004, ¶382, ¶378. To support this functionality, the chassis includes multiple slots for receiving “modular” cards. Ex-1004, ¶98 (chassis “may include Slot 0 configured to receive Card 0, Slot 1 configured to receive Card 1,...and Slot n configured to receive Card n, where n equals any suitable number”). “For example, in some embodiments, chassis 50 is a 3-slot chassis, a 5-slot chassis, or a 12-slot chassis.” *Id.* The cards are modular and are thus meant to be flexibly arranged, for example by swapping the cards to reconfigure them in different combinations or numbers.

Ex-1004, ¶176 (the chassis “may be configured to support any suitable number of cards”). The cards are designed to be swappable and have features to support their insertion and removal from the slots as desired. Ex-1004, ¶714 (“Each card 54 may also include a power button and any suitable handles, latches, locks, etc., for inserting, removing, and/or locking card 54 in chassis 50.”), ¶382 (“In some embodiments, network testing system 16 is configured such that cards 54 can boot and operate independently if desired, and be hot-swapped without affecting other the operation of the other cards 54, without the need for additional redundant hardware.”).

79. Stroud teaches the swappable cards are **coupled through a messaging interface network** (e.g., through Ethernet over the backplane interface network). For example, the cards are connected to each other through an Ethernet network that goes through physical backplane interfaces. Ex-1004, ¶101 (“Cards 54 may be communicatively coupled to each other via the backplane 56”), ¶113, ¶176, ¶¶353-354, ¶¶376-378, ¶382, ¶393, ¶99:

Each card 54 may be plugged into a backplane 56, which may include physical connections 60 for communicatively connecting cards 54 to each other, as discussed below. While cards may be interconnected, each card is treated for some purposes as an independent unit. Communications within a card are considered to be “local” communications. Two different cards attached to the same backplane

may be running different versions of software so long as the versions are compatible.

See also Ex-1004, ¶115 (“[M]ultiple blades 54 may be connected together via backplane 54 to create larger systems.”), Fig. 3:

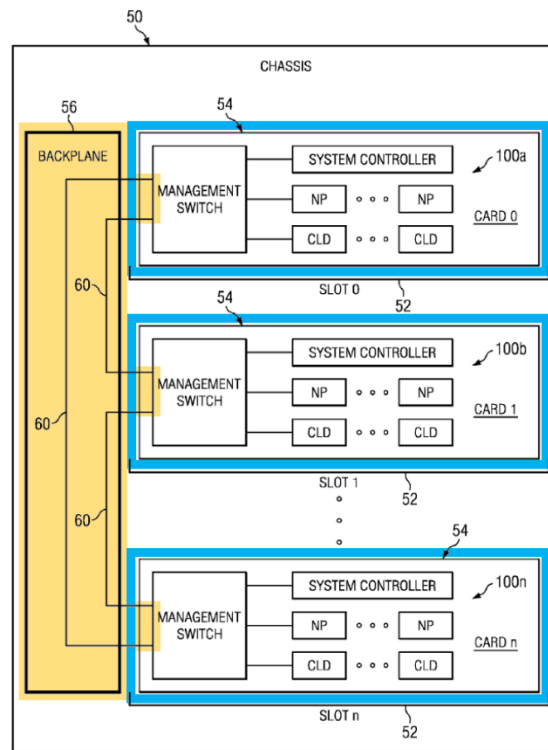


FIG. 3

80. As Stroud explains, the cards are connected to each other through a packet-based message network, e.g., an Ethernet network, that goes through the backplane. “[T]he cards 54 in chassis 50 are physically connected to each other via Ethernet over the backplane 56.” Ex-1004, ¶377, ¶176 (“Backplane 56 may provide a system for interconnecting the management Ethernet provided by the management switches 110 of multiple cards 54....”). Stroud teaches configuring the backplane network with “mesh” and “star” topologies. Ex-1004, ¶393 (“[S]lots

52 are connected in a mesh fashion by backplane 56....”), ¶389 (“different backplane design, e.g., by using a ‘star’ configuration as opposed to a ‘mesh’ configuration”).

81. Stroud further describes how the cards in the chassis are “communicatively coupled to each other via the backplane 56.” Ex-1004, ¶101. Each card contains a “management switch” that connects to the interface on the backplane. Ex-1004, ¶113 (“Management switch 110 allows and manages communications among...between components of testing architecture 100A and components of one or more other cards 54 (e.g., via backplane 56 as discussed above with respect to FIG. 3)”), ¶114 (“management switch 110 coupled to network processors 105 and CLDs 102A, 102B, and 102C, as well as to backplane 56 (e.g., for connection to other cards 54)”); *see also id.*, Figs. 4-5 (showing connection to backplane). Stroud discloses an example where the management switch of the card connects to the backplane with a serial gigabit media-independent interface (SGMII). Ex-1004, ¶¶314-319 (“management switch 110 is connected to the following other devices:...backplane 328 (three SGMII connections)”). SGMII is a serial variant of GMII that was developed by Cisco as an extension of IEEE 802.3z and provides a serial low-voltage, differentially signaled (“LVDS”) interface for Ethernet packets. Ex-1043, 2:10-35; Ex-1044,

000001-3, 000008. SGMII was used in high-speed networking applications such as Ethernet switches and routers.

82. The cards send messages (e.g., Ethernet packets) through the backplane interface network. Ex-1004, ¶174:

The management switch 110 connects to every CLD 102, network processor 105, and control CPU 106 in the system 100A. In some embodiments, management switch 110 comprises a management Ethernet switch configured to allow communication of for 1-10 Gbit traffic both between blades 54 and between the various processors 105, 106 and CLDs 102 on each particular blade 54. *Management switch 110 may route packets* based on the MAC address included in each packet passing through switch 110. Thus, management switch 110 may essentially act as a router, *allowing control CPUs 106 to communication with network processor 105 and CLD 102 on the same card 54 and other cards 54 in the system 16*. In such embodiment, all subsystems are controllable via Ethernet, such that additional processors and CLDs may be added by simply chaining management switches 110 together.

See also Ex-1004, ¶¶113-115, ¶365, Fig. 3. As Stroud explains, the cards generate messages that are communicated to each other through the Ethernet backplane interface network via the management switch. As Stroud explains, the modules on the cards generate messages. *See, e.g.,* Ex-1004, ¶349 (“control messages”), ¶361 (“error message”). The messages include local communications within a card and

messages that go outside the card, either to other cards or other devices on the network. Ex-1004, ¶399 (“a particular network processor sends a network message to a device under test”), ¶99 (“[c]ommunications within a card are considered to be ‘local’ communications” in contrast to the messages sent to other cards over the Ethernet backplane interface).

83. In summary, limitation [1a] is obvious in view of Ground 1.

[1b] each of the one or more swappable pods or cards having one or more hardware modules or one or more software modules;
--

84. Stroud teaches **each of the one or more swappable pods or cards having one or more hardware modules** (e.g., CLD modules) **or one or more software modules** (e.g., controller or network processor modules). For example, Stroud teaches that each **card** “may include any architecture 100 of **hardware**, **software**, and/or firmware components for providing the functionality of network testing system 16.” Ex-1004, ¶100, Fig. 3. “[N]etwork testing system 16 comprises a *hardware*- and *software*-based testing and simulation platform that includes of a number of interconnected subsystems” that “operate independently or in concert” in an interconnected manner. *Id.*, ¶85.

85. The **hardware** modules include **CLD** modules, while the **software** modules include Stroud’s teachings for **network processors** (“NP”) and **controllers**. Ex-1004, Fig. 3. Figure 4 illustrates an exemplary card architecture (Ex-1004,

¶¶103-113) where, consistent with Figure 3, each card comprises “a **controller** 106, two **network processors** 105 and **multiple CLDs** 102 coupled to a management switch 110.” Ex-1004, ¶103, Fig. 4.

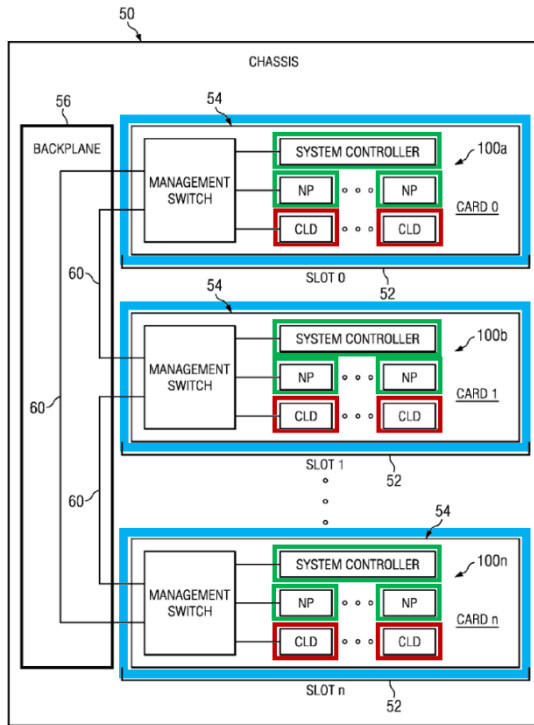


FIG. 3

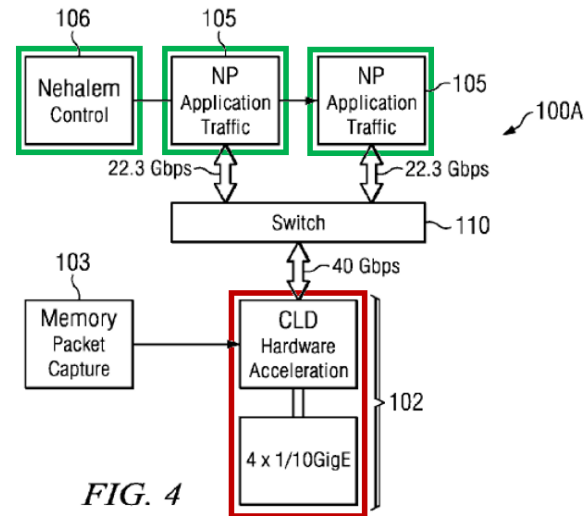


FIG. 4

Likewise, Figure 5 illustrates a controller, multiple NPs, and multiple CLDs on each card. Figure 5 provides “a more detailed illustration of the example testing and simulation architecture 100A shown in FIG. 4.” Ex-1004, ¶114, Fig. 5:

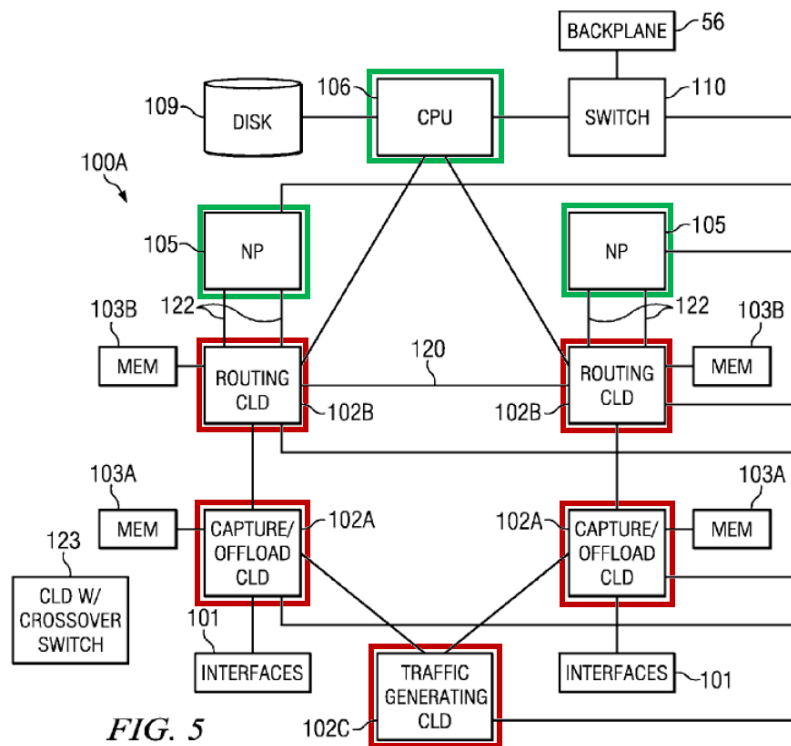
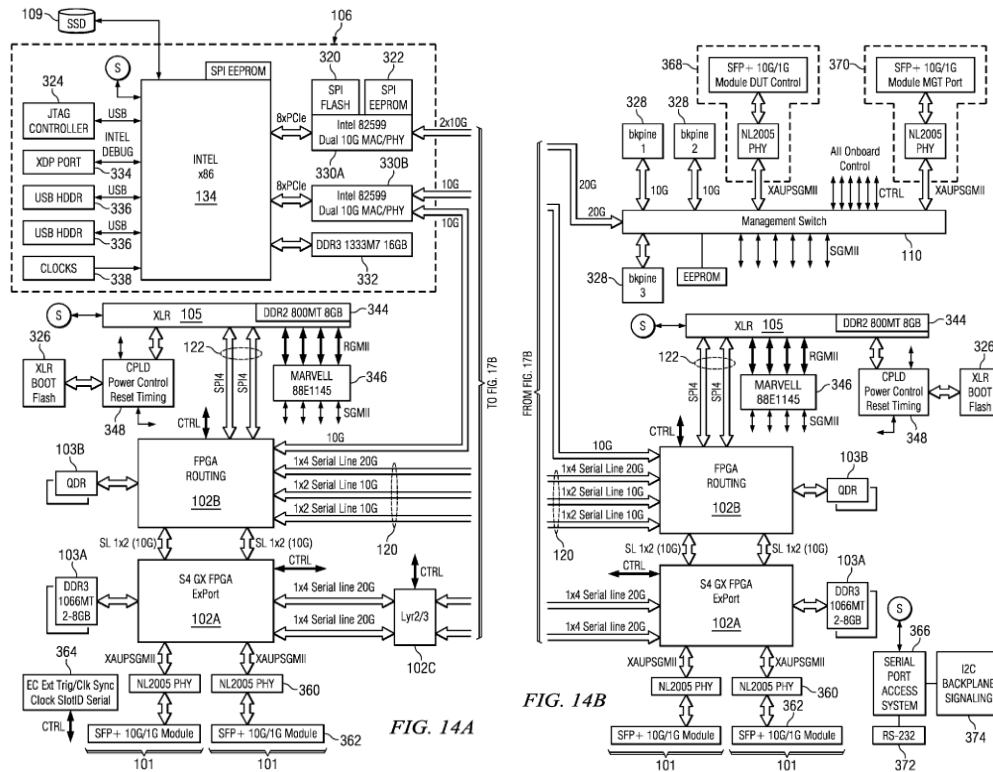


FIG. 5

As shown in Figure 5, each card includes:

...controller 106; memory 109 coupled to controller 106; two *network processors* 105; *various CLDs* 102 (e.g., capture and offload CLDs 102A, router CLDs 102B, and a traffic generation CLD 102C); memory devices 103A and 103B coupled to CLDs 102A and 102B; respectively; management switch 110 coupled to network processors 105 and CLDs 102A, 102B, and 102C, as well as to backplane 56 (e.g., for connection to other cards 54); test interfaces 101 for connecting testing architecture 100A to a system 18 to be tested; and/or any other suitable components for providing any of the various functionality of network testing system 16 discussed herein or understood by one o[f] ordinary skill in the art.

Ex-1004, ¶114. Additional details are shown in Figs. 14A-14B. Consistent with Figures 3-4, Figures 14A-14B teach multiple CLDs, network processors, and a controller on each card. Ex-1004, ¶¶456-459, Figs. 14A-14B.



86. Regarding hardware modules, Stroud teaches that the configurable logic devices (“CLDs”) are hardware modules with “programmable logic units,” for example “programmable logic devices (PLDs)” and “FPGAs.” Ex-1004, ¶107 (“As used herein, the term configurable logic device (CLD) means a device that includes a set of programmable logic units, internal memory, and high-speed internal and external interconnections: Examples of CLDs include field programmable gate arrays (FPGAs) (e.g., ALTERA STRATIX family, XILINX

VIRTEX family, as examples), programmable logic devices (PLDs), programmable array logic devices (PAL), and configurable programmable logic devices (CPLDs) (e.g., ALTERA MAXII, as an example”), ¶¶110-112, ¶¶311-313. Stroud’s CLDs include user-programmable logic, where the user—rather than the chip manufacturer—programs and alters the connection pattern of the circuits, creating custom circuits for performing custom functions in hardware. Ex-1004, ¶107, ¶¶110-112. Therefore, Stroud teaches that the CLDs are hardware modules, where user-programmable circuits perform the desired operations in user-programmed hardware, rather than software modules where software programs run on fixed hardware (with fixed circuits).³ Ex-1004, ¶¶107-112 (contrasting “processor and CPU [which] means general computing devices with fixed instruction sets” from “configurable logic device (CLD) [which] means a device that includes a set of programmable logic units”), ¶¶311-313 (discussing configuring FPGAs to perform different functions such as “routing,” “offload/capture,” and “layer 2 or layer 3 egress”).

³ The ’753 patent confirms that Stroud’s FPGAs within a card fall within the scope of the claimed “hardware modules” because it is one of the ’753 patent’s explicit examples. Ex-1001, 5:19-29.

87. Unlike ASICs or conventional processors where the logical connections of circuits are fixed when the IC is manufactured, Stroud’s CLDs include “configurable logic” where those logical circuit connections can be programmed and reprogrammed by the user. *Id.*, ¶109 (“A configurable logic device may also be programmed electronically at manufacture time, initialization time, or dynamically.”), ¶117 (“An CLD (Field Programmable Gate Array) is a reprogrammable device that can be modified to simulate many types of hardware. Being reprogrammable, it can be continually expanded to offer new acceleration and network analysis functionality with firmware updates.”). In other words, the functions of the hardware circuits are user-programmable. *Id.*

88. Regarding software modules, Stroud teaches network processor modules that run application software. Ex-1004, ¶309 (“Network processor 105 provides *software programmable computing* that may be optimized for network applications....Boot flash 326 may also store *application software to be executed on network processor 105*.”). Stroud teaches several exemplary applications, including application-level simulation and measurement with deep packet inspection for “analyz[ing] traffic beyond individual packets” and “instead look[ing] at application flows.” *Id.*, ¶219. In these applications, software runs on network processors. *Id.*, ¶224 (“Software 142 of network processor 105 configured to generate application traffic and generate statistics.”), ¶229 (“In some

embodiments, the network processors 105 execute software 142 that implements both the networking stack (Ethernet, TCP/IP, routing protocols, etc.) and the application stack that is typically present on a network device. In this sense, the software 142 can simulate network clients (e.g., Desktop PCs), servers, routers, and a whole host of different applications. This programmable ‘application engine’ software 142 is given instructions on how to properly simulate a particular network or application by an additional software layer.”). For bandwidth management applications, “software on a network processor assigns SPI4 interfaces to processor cores.” *Id.*, ¶464.

89. In general, the network processors run software programs to generate, receive, and process network data packets. Ex-1004, ¶105 (“In general, network processors 105 are programmed to generate outbound network data in the form of one or more data packets and are programmed to receive and process inbound network data in the form of one or more data packets.”). The network processors also execute software for the TCP/IP stack and user- and application-layer simulations. *Id.*, ¶149 (“1. The primary TCP/IP stack used for network traffic simulation executes on the network processor 105.”), ¶150 (“2. IP and Ethernet-layer address allocation and routing protocols are handled by the network processor 105.”), ¶151 (“3. User and application-layer simulation also run on the network processor 105.”), ¶¶152-153.

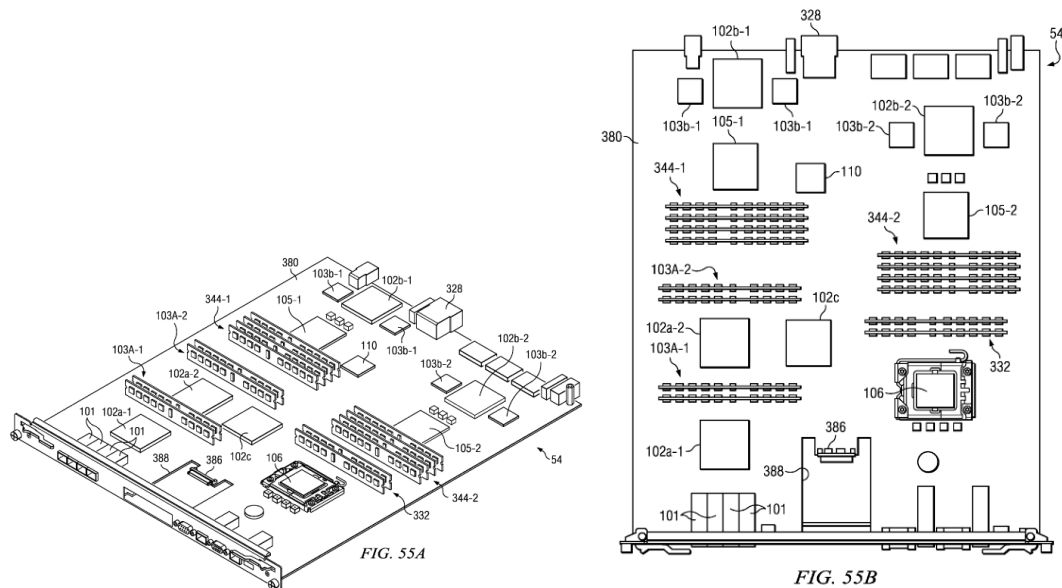
90. Each card also includes software modules running on controller 106. Ex-1004, ¶188 (“management processor 134 of controller 106 configured to execute controller software 132”), ¶208, ¶227, ¶262 (“management processor 134 on controller 106 configured to execute the controller software 132, collect and store statistics, and/or generate malicious application traffic”), ¶263 (“the management processor 134 of controller 106 can generate malicious traffic packet-by-packet and forward these to the network processor 105 as if they were generated locally”), ¶266 (“management processor 134 of controller 106, in addition to providing a place for much of the control software for various subsystems to execute, may also host a statistics database 160 and reporting engine 162”). The controller runs software applications and further provides software for execution on network processors. Ex-1004, ¶234:

The details on how to simulate applications reside in software 144 that runs on the management processor 134 on controller 106. A user can model an application behavior in a user interface (see, e.g., Figs. 13A-13D) that provides high-level application primitives, such as to make a database query or load a web page, for example. These high-level behaviors are translated by software 144 into low-level instructions, such as “send a packet, expect 100 bytes back,” which are then executed by the application engine 142 running on the network processor 105. New applications can be implemented by a user (e.g., a customer or in-house personnel), without any changes to the application engine 142

itself. Thus, it is possible to add new functionality without upgrading software.

91. The software module for each network processor and controller includes a processor for executing software (Ex-1004, ¶144 (“Each network processor (NP) 105 may be a general purpose CPU with multiple cores, security, and network acceleration engines.”), ¶105 (“In some embodiments, network processors 105 may be general purpose CPUs”), ¶188) and the software itself. Each card runs a “software image.” Ex-1004, ¶376 (explaining how cards can run the same or different software images). Stroud teaches storing the software for a card on, e.g., a hard drive connected to the card. *Id.*, ¶158 (“In some embodiments, all firmware or software used by the rest of the system 100A, except for firmware required to start the controller 106 itself (BIOS) resides on the disk drive 109.”), Fig. 5, 6A, 7A.

92. Stroud provides further details of the physical layout of components on a card, including controllers, network processors, and CLDs. Ex-1004, Figs. 55A-55B:



Ex-1004, ¶717:

...card 54 includes a printed circuit board 380 that houses a pair of capture and offload CLDs 102a-1 and 102a-2 and associated DDR3 SDRAM memory modules (DIMMs) 103A-1 and 103A-2, a pair of routing CLDs 102b-1 and 102b-2 and associated QDR SRAMs 103b-1 and 103b-2, a traffic generation CLD 102C, a pair of network processors 105-1 and 105-2 and associated DDR2 SDRAM DIMMs 344-1 and 344-2, a control processor 106 and associated DDR3 SDRAM DIMMs 332, a management switch 110, four test interfaces 101, a backplane connector 328, a notch or bay 388 that locates a drive connector 386 for receiving a disk drive assembly 1300 that houses a disk drive 109, and various other components (e.g., including components shown in FIGS. 14A and 14B).

These correspond to the controller, network processors, and CLDs “shown in FIGS. 14A and 14B” (*id.*) that comprise hardware and software modules in the swappable cards.

93. For at least these reasons, limitation [1b] is taught by Ground 1.

[1c] one or more of the plurality of swappable pods or cards having a portion for user-definable hardware modules or user-definable software modules;
--

94. Stroud teaches **one or more of the plurality of swappable pods or cards having a portion for user-definable hardware modules** (e.g., CLD modules) or **user-definable software modules** (e.g., controller or network processor modules). For example, Stroud teaches that each **card** (annotated in cyan) has one portion with multiple **configurable logic device (“CLD”)** modules (annotated in red) and another portion having **multiple network processor (“NP”)** modules and/or a **controller** module (annotated in green). Ex-1004, ¶83, ¶¶103-109 (“FIG. 4 is an high-level illustration of an example architecture 100A of a card 54 of network testing system 16...As shown, example architecture 100A...’ may include a controller 106, two network processors 105 and multiple CLDs 102 coupled to a management switch 110, and memory 103 coupled to the CLDs 102”), ¶188, Figs. 3-4:

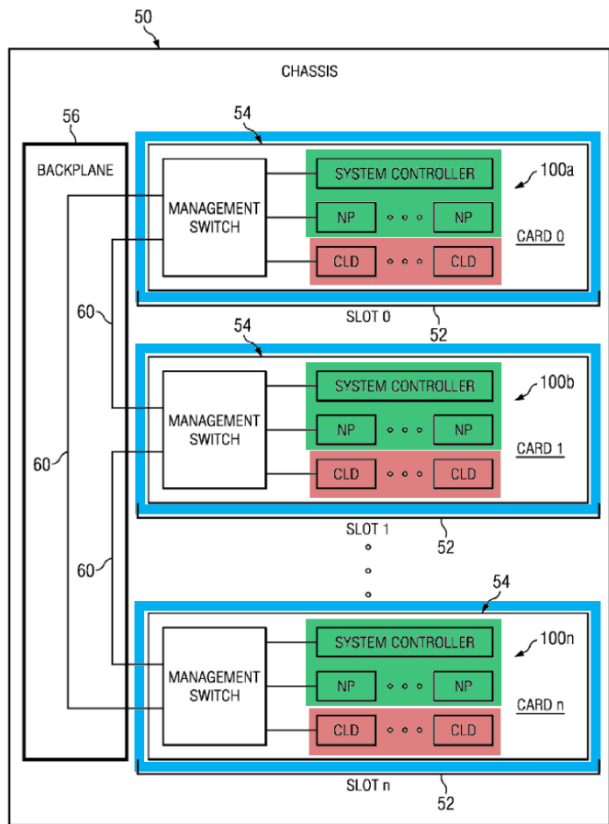


FIG. 3

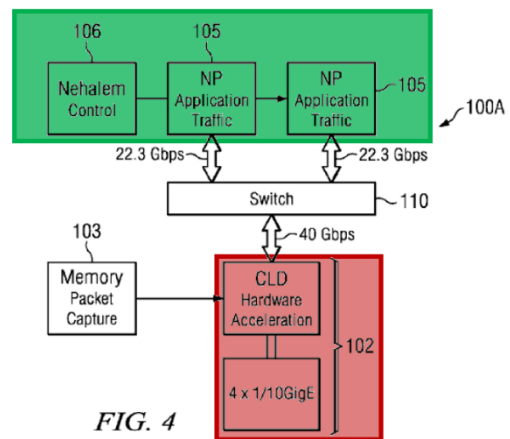


FIG. 4

95. Stroud teaches exemplary physical layouts showing the portions of the card with hardware or software modules. Ex-1004, Figs. 55A-55B:

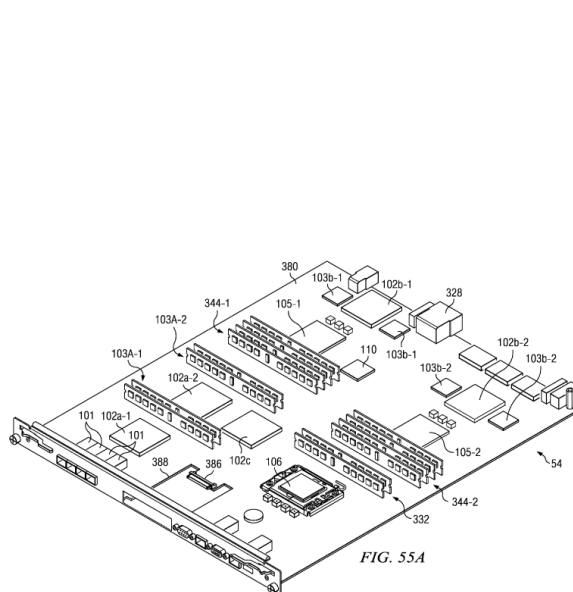


FIG. 55A

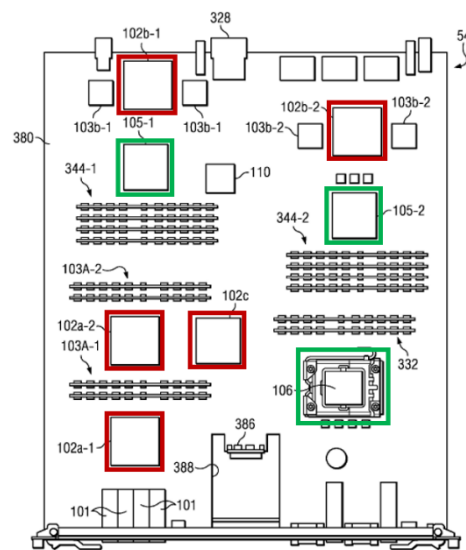


FIG. 55B

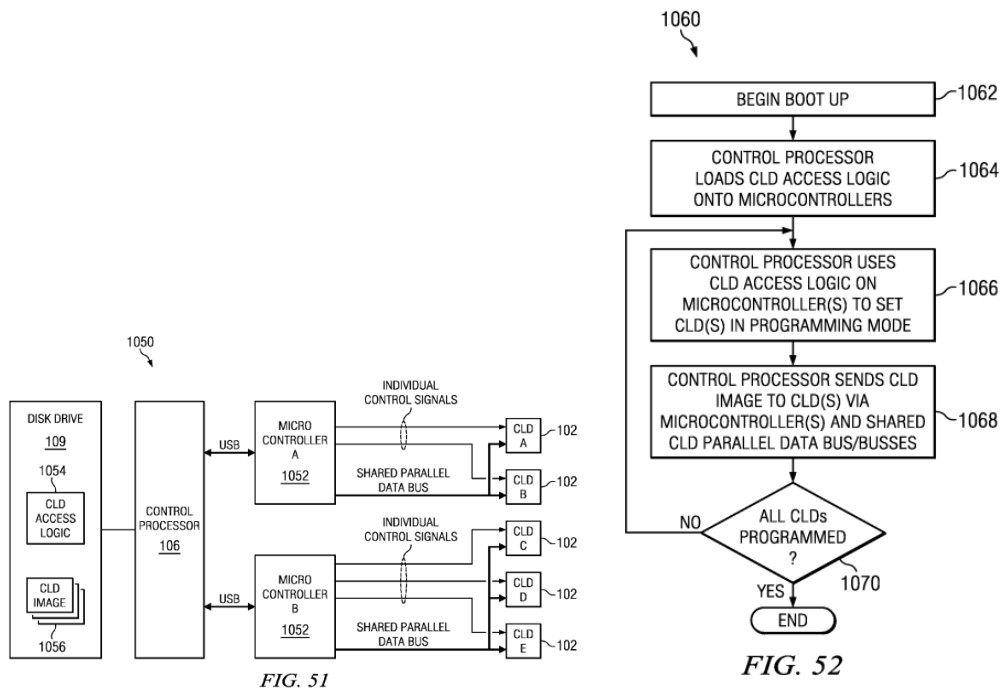
Ex-1004, ¶717:

card 54 includes a printed circuit board 380 that houses *a pair of capture and offload CLDs 102a-1 and 102a-2* and associated DDR3 SDRAM memory modules (DIMMs) 103A-1 and 103A-2, *a pair of routing CLDs 102b-1 and 1[02]b-2* and associated QDR SRAMs 103b-1 and 103b-2, *a traffic generation CLD 102C*, *a pair of network processors 105-1 and 105-2* and associated DDR2 SDRAM DIMMs 344-1 and 344-2, *a control processor 106* and associated DDR3 SDRAM DIMMs 332, a management switch 110, four test interfaces 101, a backplane connector 328...

96. The card has one portion for user-definable hardware modules. As explained previously, Stroud’s CLD modules are hardware modules with user-programmable logic, including, for example, “FPGAs” and “PLDs”; that analysis is incorporated here. *Supra* §V.A.3.[1b]; Ex-1004 ¶85, ¶100, ¶107, ¶¶109-112, ¶114, ¶117, ¶¶457-459, Figs. 3-5, 14A-14B. With Stroud’s CLD modules, the user defines the function of the hardware by altering the logical interconnections of the circuits in the chip. *Id.* Unlike ASICs or conventional processors where the logical connections of circuits are fixed when the IC is manufactured, Stroud’s CLDs include “blocks of configurable logic” where those logical circuit connections can be programmed and reprogrammed by the user. Ex-1004, ¶110, ¶109 (“A configurable logic device may also be programmed electronically at manufacture time, initialization time, or dynamically.”), ¶117 (“An CLD (Field

Programmable Gate Array) is a reprogrammable device that can be modified to simulate many types of hardware. Being reprogrammable, it can be continually expanded to offer new acceleration and network analysis functionality with firmware updates.”). This is possible because the CLD is “a device that includes a set of programmable logic units, internal memory, and high-speed internal and external interconnections” such as FPGAs, PLDs, PALs, and CPLDs. Ex-1004, ¶107. By configuring the hardware logic as desired, such devices “provide high-performance, specialized computation, data transfer, and data analysis capabilities to process certain data or computation intensive tasks at or near the network line rates.” Ex-1004, ¶106. Therefore, the user configures Stroud’s CLDs by defining the actual hardware, rather than by writing software.

97. Stroud teaches detailed steps for loading custom logic onto the CLD. Figure 51 teaches circuitry, and Figure 52 teaches steps, for programming the user-definable CLDs. Ex-1004, ¶¶63-64, ¶¶689-700, Figs. 51-52:



For example, Stroud teaches that the CLDs can be programmed using the microcontroller and the control processor. Ex-1004, ¶¶691-692, ¶694.

98. Each card also has a portion for user-definable software modules. As explained previously, each **card** has a portion for multiple **network processor modules** and a **controller** that run user-defined application software; that analysis is incorporated here. *Supra* §V.A.3.[1b]; Ex-1004, ¶103 (“FIG. 4 is an high-level illustration of an example architecture 100A of a card 54...As shown, example architecture 100A,...may include a controller 106, two network processors 105 and multiple CLDs 102 coupled to a management switch 110, and memory 103 coupled to the CLDs 102.”), ¶105, ¶144, ¶149, ¶¶150-153, ¶158, ¶188, ¶208, ¶219, ¶223, ¶227, ¶229, ¶234, ¶¶262-263, ¶266, ¶309, ¶376, ¶464, Figs. 3-5, 14A-14B:

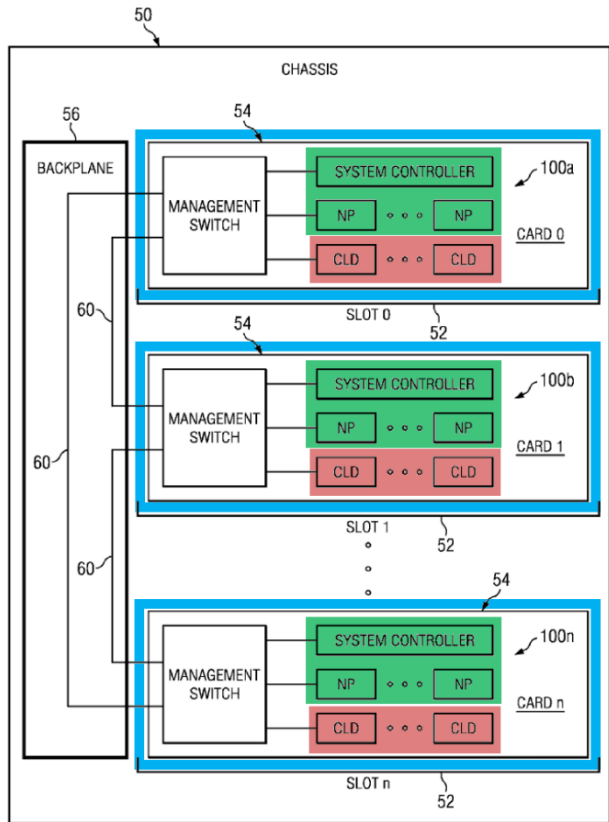


FIG. 3

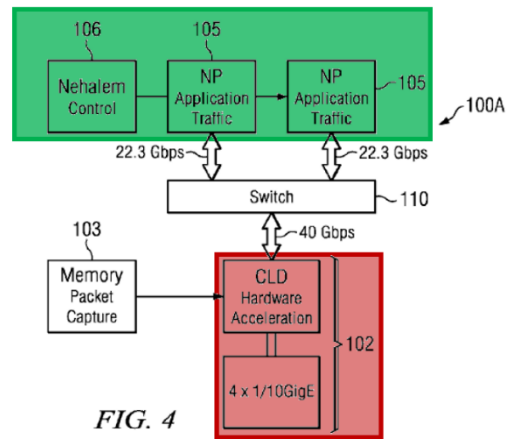
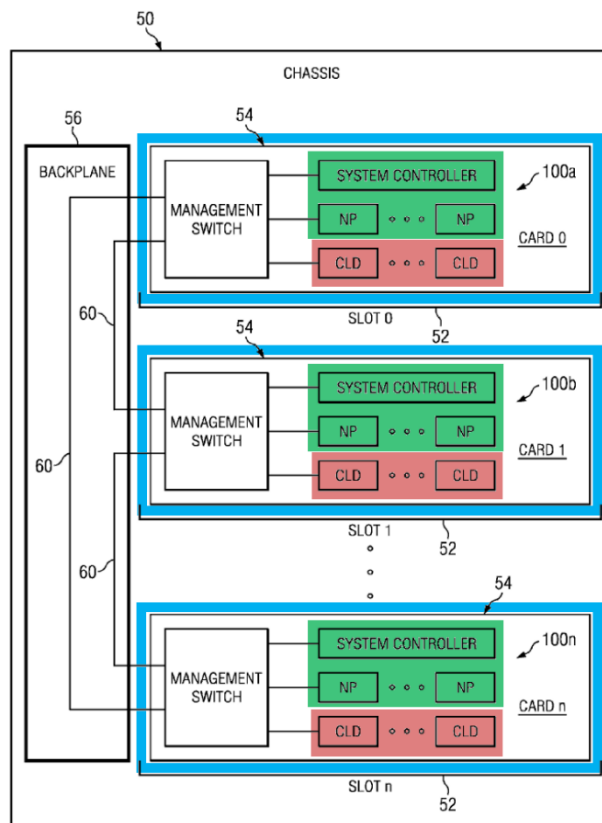


FIG. 4

99. The user defines Stroud's software modules with software programs that run on the network processors and the controller for the user's application, including for example network packet generation and processing, TCP/IP stack, user and application-layer simulation, deep-packet analysis, and/or collecting network statistics. Ex-1004, ¶158, ¶105, ¶¶219-220, ¶223, ¶144, ¶¶149-153, ¶188, ¶¶227-229, ¶208, ¶234, ¶¶262-263, ¶266, ¶309, ¶464, Figs. 3-5, 14A-14B. The software programs are stored on non-volatile storage, e.g., "removable disk storage109," attached to the card. Ex-1004, ¶158, ¶¶170-173 ("For example, memory device 109 may comprise a removable, solid-state drive (SSD) in a custom carrier that allows hot-swapping and facilitates changing *software* or

database contents on an installed board.”). Accordingly, each of Stroud’s network processors and the controller is a user-definable software module.



Additionally, Stroud teaches configuring the CLDs to implement general purpose processors. Ex-1004, ¶112 (“The term instruction executing device

includes...CLDs that have been programmed to implement an instruction set.”),
¶321.

101. For at least these reasons, limitation [1c] is obvious in view of Ground
1.

[1d] the plurality of swappable pods or cards being user-configurable to implement data flow processing architectures; and

102. Stroud teaches the **plurality of swappable pods or cards being user-configurable**. As explained above, Stroud teaches a plurality of swappable pods or cards; each card has user-definable hardware and software modules. That analysis is incorporated here. *Supra* §§V.A.3.[1b]-[1c] (analysis incorporated by reference).

103. Each hardware module comprises a “*configurable logic device* (CLD)” that “includes a set of programmable logic units” where the user alters the connection pattern of the circuits to define the function of the hardware. Ex-1004, ¶107, ¶117; *supra* §V.A.3.[1b]. Stroud teaches FPGAs and “programmable logic devices (PLDs)” as examples—these were common CLDs where the user programmed and configured the hardware circuits, rather than the manufacturer. Ex-1004, ¶109 (“A configurable logic device may also be programmed electronically at manufacture time, initialization time, or dynamically.”), ¶117 (“An CLD (Field Programmable Gate Array) is a reprogrammable device that can

be modified to simulate many types of hardware. Being reprogrammable, it can be continually expanded to offer new acceleration and network analysis functionality with firmware updates.”). Stroud teaches that configurable logic devices are user-configurable because the user configures the programmable logic units by altering the logical connections of circuits, creating custom hardware (with custom circuits) for performing specific functions. Ex-1004, ¶¶107-112. Therefore, Stroud teaches that the CLDs are user-configurable for performing the user’s applications. *See id.*

104. The user configures software modules using software programs that run on the network processors and the controller, including, for example, software configured to perform network packet generation and processing, TCP/IP stack, user and application-layer simulation, deep-packet analysis, and/or collecting network statistics. Ex-1004, ¶158, ¶105, ¶¶219-220, ¶223, ¶144, ¶¶149-153, ¶188, ¶¶227-229, ¶208, ¶234, ¶¶262-263, ¶266, ¶309, ¶464, Figs. 3-5, 14A-14B.; *supra* §V.A.3.[1c]. The user’s software programs are stored on non-volatile storage, e.g., a hard disk, attached to the card. Ex-1004, ¶158, ¶¶170-173.

105. Therefore, Stroud’s cards are user-configurable because the user configures the hardware modules and software modules on each card. Additionally, the user configures the overall function of the card and the way in which the hardware and software modules work together to perform the user’s

application. *E.g.*, Ex-1004, ¶84 (“e.g., database server, Web server,” etc.), ¶¶89-93.

106. Stroud teaches the plurality of swappable pods or cards being user-configurable to **implement data flow processing architectures**. As I explained above, Stroud’s platform is used for data flow processing. *Supra* §V.A.3.[1pre] (analysis incorporated by reference). For example, Stroud teaches using “[c]onfigurable logic devices” on the swappable cards to “provide high-performance, specialized computation, data transfer, and data analysis capabilities to process certain data or computation intensive tasks at or near the network line rates.” Ex-1004, ¶106. Here, “network line rates” refers to the speed of data transmission over a network connection, for example 1 Gigabit or 10 Gigabit Ethernet. By processing data at network-line rates, Stroud’s user-configurable cards implement data-flow processing architectures that perform computation and analysis on packetized data flows at the speed in which data is transmitted over the network. *Id.* Stroud uses the swappable cards in “the operation of a test system” architecture (Ex-1004, ¶84) that generates and processes “synthetic data flows” (Ex-1004, ¶298).

107. In particular, the cards in Stroud’s architecture are configured to implement data flow processing architectures such as:

(1) high-speed, high-resolution network packet capture, (2) high-speed packet generation and measurement, (3) application-level simulation and measurement, (4) security and exploit simulation and analysis, and (5) statistics collection and reporting. Thus, as discussed above with respect to FIG. 1, network testing system 16 may comprise a high-speed, high-resolution network packet capture subsystem 20, a high-speed packet generation and measurement subsystem 22, an application-level simulation and measurement subsystem 24, a security and exploit simulation and analysis subsystem 26, and/or a statistics collection and reporting subsystem 28. The architecture of system 16 (e.g., example architecture 100A discussed above or example architecture 100B discussed below) may allow for some or all of these subsystems 20-28 to operate simultaneously and cooperatively within the same software and hardware platform. Thus, system 16 may be capable of generating and analyzing packets at line rate, while simultaneously capturing that same traffic, performing application simulation and security testing.

Id., ¶177. Therefore, Stroud configures its cards to implement several data flow processing architectures that process packetized data flows by, for example, capturing packets, generating and measuring packets, simulating and measuring network traffic at the application level, analyzing security in network data flows, simulating security exploits in packet traffic, and processing data flows to collect and report statistics of the data flow. *Id.*

108. For at least these reasons, limitation [1d] is obvious in view of Ground

1.

[1e] a network coupled to the one or more swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.

109. Stroud teaches **a network coupled to the one or more swappable pods or cards**. For example, Stroud teaches that the cards in the chassis are “communicatively coupled to each other via the backplane 56” using an Ethernet network. Ex-1004, ¶101, ¶99 (“Each card 54 may be plugged into a backplane 56...”), ¶115 (“[M]ultiple blades 54 may be connected together via backplane 54 to create larger systems.”), ¶¶174-176. As I explained above for limitation [1a] (*supra* §V.A.3.[1a] (analysis incorporated by reference)), the cards are coupled to the Ethernet backplane interface network; therefore the backplane interface network is coupled to the cards. The cards are coupled to the backplane via the management switch on each card. *Id.*, ¶101, ¶176 (“Backplane 56 may provide a system for interconnecting the management Ethernet provided by the management switches 110 of multiple cards 54...”).

110. Stroud teaches that the **network supports messaging based communication using packets**. For example, Stroud teaches the backplane

network being an Ethernet network. Ex-1004, ¶174 (“In some embodiments, management switch 110 comprises a management Ethernet switch configured to allow communication of for 1-10 Gbit traffic both between blades 54 and between the various processors 105, 106 and CLDs 102 on each particular blade 54.”). A POSITA looking to implement a scalable network simulation or testing platform would have been motivated to apply Stroud’s teaching and use Ethernet because of its prevalence and widespread adoption in data centers and networks around the world. Moreover, Ethernet provided low-latency, high-throughput communication, making it well-suited for scalable platforms, such as those taught by Stroud. Stroud teaches techniques for monitoring and troubleshooting, including packet sniffing, traffic analysis, and error checking, using Ethernet. *E.g.*, Ex-1004, ¶322, ¶495, ¶279, ¶503.

111. Stroud teaches that its Ethernet communications use “Ethernet packets.” Ex-1004, ¶349, ¶181, ¶236, ¶242, ¶¶345-346, ¶¶349-350, ¶¶356-357, ¶121, ¶119. Therefore, the “communication” through Stroud’s Ethernet switches (Ex-1004, ¶174) comprises messaging-based communication because it includes Ethernet messages (e.g., packets/frames) exchanged between cards, processors, CLDs, and other components. The Ethernet protocol used packetized messages, called frames, where small segments of data were transmitted. Switches directed packets to their destination. A POSITA would have understood this because

Ethernet had been the most common wired networking protocol since the 1990s and was taught and used in universities worldwide. Stroud taught management switches that directed Ethernet packets between cards. Ex-1004, ¶314 (“Management switch 110 is a high-speed Ethernet switch capable of cross connecting various devices on a single blade or across blades in the network testing system.”), ¶174 (“Thus, management switch 110 may essentially act as a router, allowing control CPUs 106 to communication with network processor 105 and CLD 102 on the same card 54 and other cards 54 in the system 16. In such embodiment, all subsystems are controllable via Ethernet, such that additional processors and CLDs may be added by simply chaining management switches 110 together.”), ¶176 (“management Ethernet provided by the management switches 110”). In Stroud’s platform, cards used Ethernet to communicate with destinations both internal and external to the chassis. Ex-1004, ¶174, ¶¶113-115, ¶365, ¶¶413-426, Figs. 3, 12, 31.

112. Additionally, Stroud teaches the use of IP packets and therefore IP-based messages. Ethernet was a layer 2 protocol that supported communication using Internet Protocol (“IP”) packets for layer 3 in the OSI model, which Stroud teaches when referencing the “packet’s layer 2/3/4 headers.” Ex-1004, ¶¶124-125. For example, Stroud teaches that the CLD generates “layer2/layer 3 traffic.” Ex-1004, ¶164. Therefore, Stroud teaches the use of the IP protocol. A POSITA

would have understood this because IP packets had been the most common layer 3 protocol used in conjunction with Ethernet at least since the rise of the Internet in the 1990s, and a POSITA would have understood how Ethernet and IP were used for communication, including in the context of Stroud's teachings, because Ethernet and IP packets had been among the most common ways to connect devices to a network or the Internet for decades. *Supra* §I.B.

113. Stroud teaches the **packets each having a header**. For example, Stroud teaches that its packets have a header, including an Ethernet header. Ex-1004, ¶124 (“[T]he capture and offload CLD 102A may parse the packet's layer 2/3/4 headers, validate all checksums (up to 2 layers), insert a receive timestamp, and forward the packet to the closest router CLD 102B for further processing.”), ¶349:

The first portion of the packet is the IEEE standard header for Ethernet packets, including the destination MAC address, the source MAC address, and the Ethernet packet type field. The type field is set to value unused the IEEE standard to avoid conflicts with existing network protocols, especially within the networking stack on the control CPU. Immediately following the standard header is an access directive format including a sequence identifier, a count, a command field, and data to be used in executing the directive. The sequence number is an identifier used by the originator of the directive for tracking completion and/or timeout of individual directives. The count specifies the number of

registers accessed by the command and the command field specifies the type of directive.

114. Stroud further teaches using distinct sub-fields within the destination MAC address of the Ethernet header. Ex-1004, ¶¶366-372. Each sub-field holds a different identifier, as further explained below. The sub-fields within the MAC address are stored within the header of an Ethernet packet: “The first portion of the packet is the IEEE standard header for Ethernet packets, including the destination MAC address, the source MAC address, and the Ethernet packet type field.” Ex-1004, ¶349, Fig. 17:

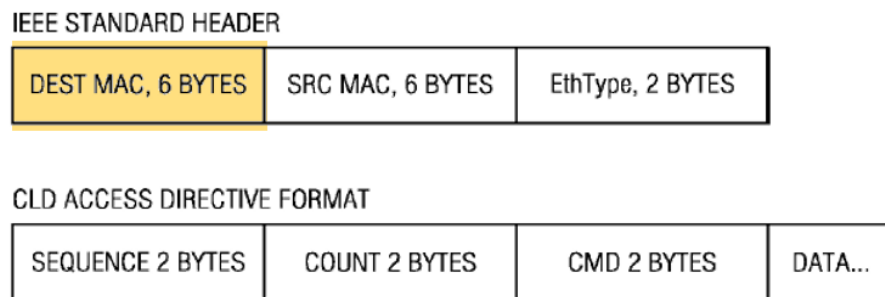


FIG. 17

115. The packet header contains each identifier recited in the claim. These will be addressed in sequence below, with chassis ID explained at the end.

116. Stroud teaches the packet header with a **board identifier**. For example, Stroud teaches a “**board slot ID**” (annotated in orange) as a “[s]ub-field[] within the destination MAC address” of the packet header. Ex-1004, ¶370, ¶347 (“Sub-fields within the destination MAC address (6 bytes total in length)

identify...a *board slot ID*.”). The board slot ID is a distinct sub-field that identifies a particular board, which resides in a particular slot in a chassis. As Stroud explains, each card is placed in a *slot* within a *chassis*. Ex-1004, ¶98, Fig. 3.

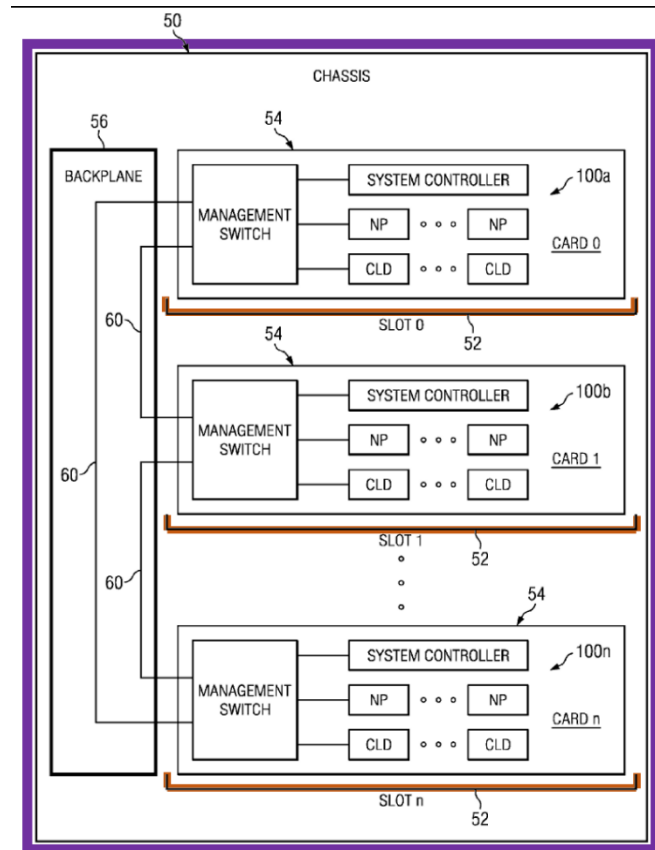


FIG. 3

“For example, in some embodiments, chassis 50 is a 3-slot chassis, a 5-slot chassis, or a 12-slot chassis.” Ex-1004, ¶98. Stroud teaches that the CLDs of a card are “on a board.” Ex-1004, ¶344. To ensure that the packet reaches one of the slots within the chassis and to identify the correct board, Stroud teaches that “[t]he MAC address for each device 102, 105, and 106 on a particular card 54 may identify the chassis slot 52 in which that card 54 is located, as well as other

identifying information.” Ex-1004, ¶381. The particular card in a chassis slot is identified by the board slot ID. Ex-1004, ¶370. Therefore, Stroud teaches that the “board slot ID” is used to route packets to the correct card slot; accordingly, Stroud teaches the header of a packet with a board identifier to route packets using the board identifier in the header.⁴

117. Stroud teaches the packet header with **a module identifier**. For example, Stroud teaches a distinct sub-field in the 6th byte of the MAC address that stores an “interface” identifier for the interface to the management switch. Ex-1004, ¶372. Stroud’s card includes a number of controller, NP, and CLD modules; each module is connected to the management switch. *Id.*, ¶101 (“Each architecture 100 may include a system controller, one or more network processors, and one or more CLDs connected to a management switch 110”), ¶314 (“management switch 110 is connected to the following other devices: controller 106 (two SGMII connections); each network processor 105 (four SGMII connections); each FPGA 102A, 102B, and 102C (one control connection); backplane 328 (three SGMII connections)”), ¶103, ¶113, ¶174, ¶176, ¶189, Figs. 3 & 5.

⁴ The ’753 patent specification describes a similar example where the packet header specifies the board identifier so that packets are routed to the correct “pod or card in the chassis...” Ex-1001, 3:4-8.

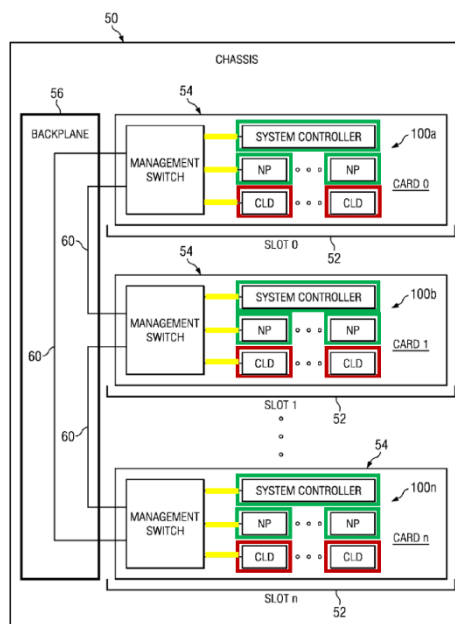


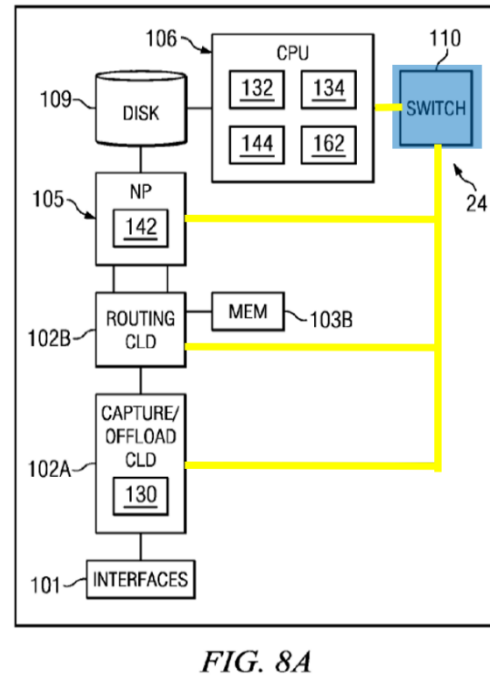
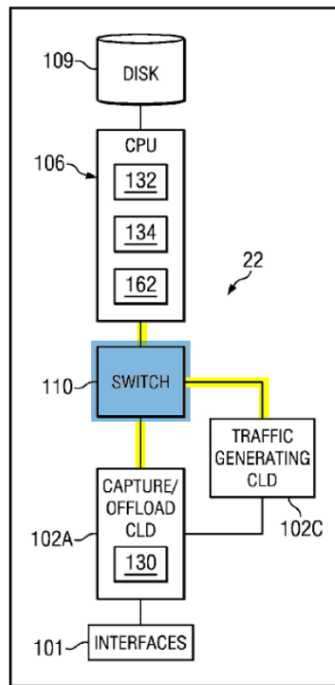
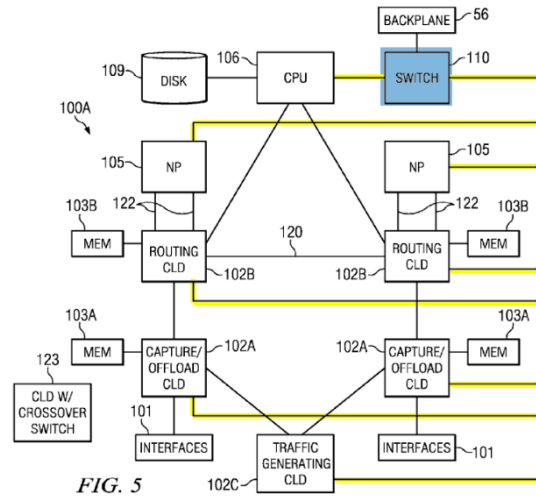
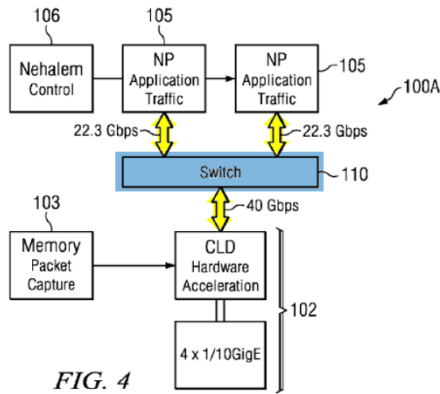
FIG. 3

118. As Stroud explains, “each interface to the management switch has its own MAC address” because the value for the field in the 6th byte of the MAC address is unique to the module within each card. Ex-1004, ¶372. Therefore, the interface identifier in the 6th byte identifies each module on a card (e.g., each controller, network processor, or CLD⁵) by its interface to the management switch (annotated yellow above). Ex-1004, ¶100, ¶372, Fig. 3.

⁵ Stroud’s interface identifier is assigned to the controller, NP, and CLD modules. As Stroud explains, each interface to the management switch has its own unique identifier. Ex-1004, ¶372. While Stroud does refer to the “processor” interface (*id.*), Stroud’s teachings are not limited to just the network processors because, controllers, NPs, and CLDs all have an interface to the management switch (*id.*,

119. Each module has a connection to the management switch (highlighted in yellow below); a module's interface to the management switch sits at the boundary between the module and the connection to the switch. Additionally, Stroud teaches each NP and CLD having a dedicated connection to the management switch on a card, illustrated by Figures. 4, 5, 7A, and 8A. Ex-1004, ¶¶174-176, ¶134, ¶¶146-147, ¶¶156-157, ¶167, ¶189, ¶310, ¶¶314-319, Figs. 4-5, 6A-9A:

Figs. 4-5, 7A-8A), which uses the unique interface identifier—and therefore unique MAC address—of each NP and CLD to route packets to them. Ex-1004, ¶366 (“Hardware and software of system 16 dynamically assigns each other Ethernet device in system 16, (including *each network processor 105, each CLD 102, and local/internal Ethernet interfaces of control CPU 106*) a MAC address that is unique within system 16”), ¶354 (“Ethernet switch 110 maintains a memory associating each port with a list of one or more MAC addresses of the device or devices....”).



120. Stroud teaches the packet header with an **instance identifier**. For example, Stroud teaches that the 5th Byte of the MAC address includes a subfield to indicate a “processor or CLD number” that is used “to *distinguish between multiple instances* of the same type of processor or CLD on the same card (e.g., two network processors 105 or two capture/offload CLDs 102 a).” Ex-1004, ¶371.

When there are multiple instances of the same type of network processor or CLD on a card, the processor or CLD number identifies the unique instance of the CLD or processor on that card.⁶

121. Stroud teaches the packet header with **a type identifier**. For example, the 2nd Byte within the MAC address includes a subfield to indicate a “chip type: e.g., processor, CLD, or other type of device.” Ex-1004, ¶368. Stroud uses this sub-field to identify the type of device, which for example could assign NP to type 0, CLD to type 1, system controller to type 2, etc. *Id.*

122. Stroud also teaches that the identifiers of the packet header—e.g., the subfields in the destination MAC address—function **so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header**. As I explained in this section, the combination of Stroud and DaSilva teaches a packet header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier. These identifiers are stored as distinct subfields within the destination

⁶ The '753 patent describes a similar example. “For example, an MMI packet 154 could be destined to chassis number two, board number three in that chassis, module number seven on that board, instance number three of that module on that board, and the module being hardware type.” Ex-1001, 8:61–65.

MAC address of a packet. Ex-1004, ¶¶366-372. The management switch of each card routes Ethernet packets to each module on the board using the destination MAC address. Ex-1004, ¶174, ¶354, ¶113. Therefore, each type of module (e.g., processor or CLD), each instance, and each module on each board can be addressed through the destination MAC address of the Ethernet header, using the particular sub-fields that Stroud teaches for that purpose. Ex-1004, ¶¶365-373, ¶¶347-349; *see also* Ex-1005, 5:13–18, 11:55-12:14, Fig. 7. Ethernet switches route packets to their destination using the destination MAC address. Ex-1004, ¶354:

Ethernet switch 110 may be implemented as a store and forward device receiving at least part of an incoming Ethernet packet before making a routing decision. The switch examines the destination MAC address and compares that destination MAC address with entries in the switch's routing table. If a match is found, the packet will be resent to the assigned port. If a match is not found, the switch may broadcast the packet to all ports. Upon receipt of a packet, the switch will also examine the source MAC address and compare that address to the switch's routing table. If the routing table does not have an entry for the source MAC address, the switch will create an entry associating the source MAC address with the port on which the packet arrived.

A POSITA would have understood this because it is a basic networking concept taught in undergraduate courses, in addition to the widespread use of Ethernet since the 1990s.

123. DaSilva teaches a packet header with a **chassis identifier**. For example, DaSilva teaches a “chassis ID.” Ex-1005, 9:58-10:14 (“The hardware device information includes, for example, *chassis ID*...”), 5:13–18 (“network nodes 110 exchange packets with pre-pended headers...that include...source *chassis ID*...”), Fig. 6 (chassis ID in Destination HDI 304):

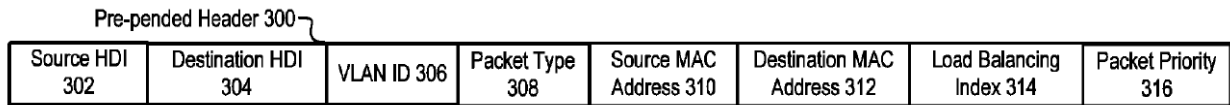


Fig. 6

The chassis identifier is a distinct sub-field that identifies a particular chassis among multiple chassis. For example, DaSilva presents an example of three chassis, assigned “Chassis ID” of 1, 2, and 3, respectively. Ex-1005, 7:58-64, Fig. 4:

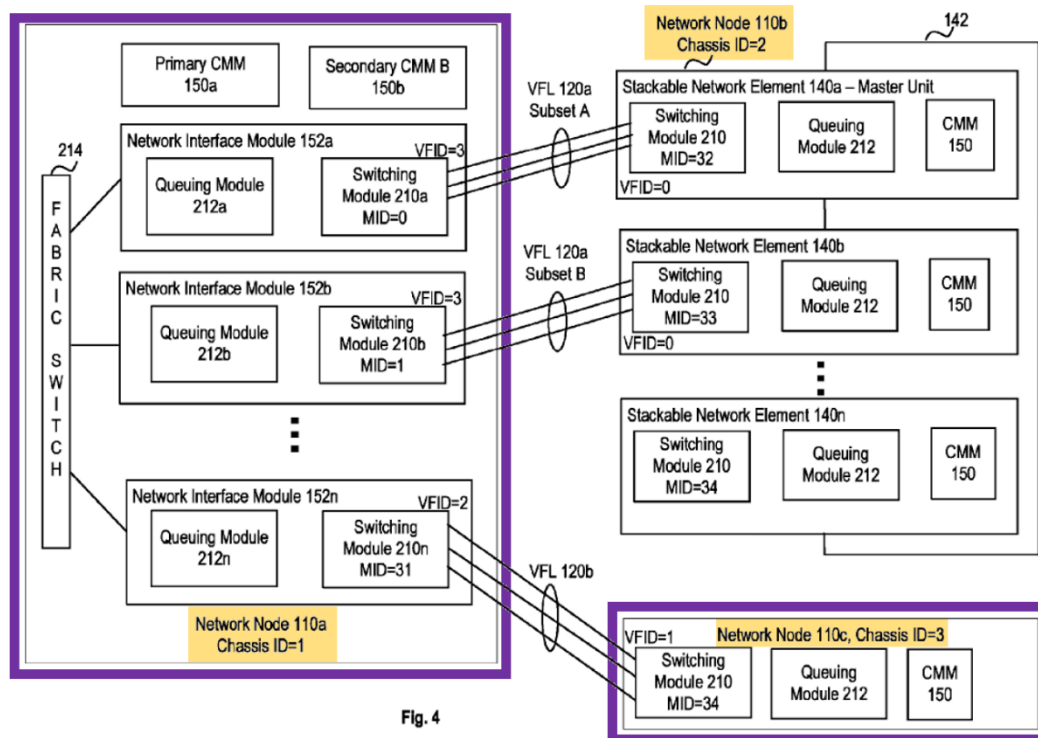


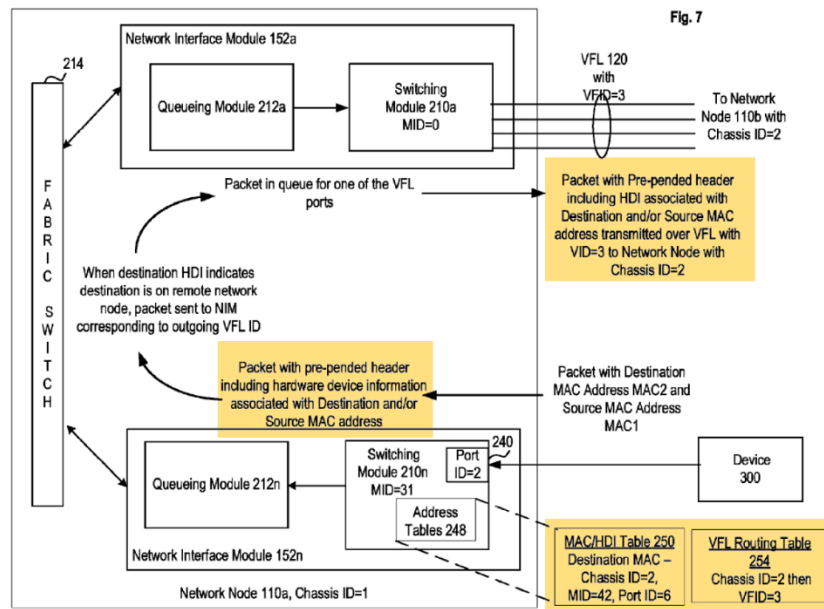
Fig. 4

Similar to Stroud’s chassis, each chassis includes cards with modules connected through a backplane interface network. *See id.*

124. DaSilva teaches 2 ways of adding the chassis ID to a packet, both of which satisfy the claim: “add[ing] a pre-pended header (or otherwise modif[y]ing the packet header) to include hardware device information (HDI) associated with the source and/or destination” where the HDI includes “chassis ID.” Ex-1005, 9:58-10:14. With the first option, the chassis ID is added to the beginning of a packet header (i.e., prepended); therefore, the packet header includes a chassis ID. *Id.* Regarding the second option, Stroud teaches a mechanism to modify the existing packet header to include identifiers in sub-fields of the destination MAC

address (e.g., by using a local MAC address). Ex-1004, ¶¶366-373. Since Stroud already includes a number of subfields, the combination would simply add another sub-field for chassis ID. In particular, the second byte of the MAC address is used to indicate the type. Ex-1004, ¶368 (“chip type”). However, Stroud only has a limited number of chip types: “processor, CLD or other.” *Id.* Two bits would be sufficient for representing three choices, leaving six bits to be appropriated for the chassis ID. Even accounting for reserving a third bit to the chip types, that would allow for five more chip types to be used in the future, that would leave at least five bits. Sixty four, or thirty-two is a reasonable number to allow for scalable expansion of Stroud’s test system. Given that Stroud allows for 256 cards per chassis (one byte dedicated to slot number) and 256 modules (one byte dedicated to the processors and CLDs) per card, thirty two chassis would account for more than 2 million and sixty four chassis would support more than 4 million processing elements to be exact (respectively 2,097,152 and 4,194,304 to be exact). The motivations for combining DaSilva and Stroud in this manner are explained below.

125. DaSilva further explains that the “chassis ID for each network node 110 is unique.” Ex-1005, 9:1-5, 4:53–54 (“a network node 110 is assigned a globally unique identifier called a *chassis identifier or chassis ID*.”). Because it is unique to the chassis, DaSilva teaches packets being routed to a particular chassis based on the chassis ID. Ex-1005, 11:55-12:14, Fig. 7:



126. As shown in Figure 7, DaSilva teaches that when a packet is received, the destination MAC address within the packet is parsed. Ex-1005, 11:60-65.

When a packet is received at a node's switching module, the destination MAC address within the packet header is matched with the corresponding hardware device information (HDI) in a lookup table. Ex-1005, 11:60-12:3. The packet's destination HDI includes the chassis ID. Ex-1005, 11:66-12:3 ("The destination HDI may include, e.g., destination chassis ID and device module identifier (MIDs) and port identifiers associated with the destination MAC address."). For example, when the destination chassis ID is determined to be "Chassis ID=2", DaSilva routes the packet to the destination which is the node with the chassis identifier 2. *Id.*, 12:26-55. To facilitate this routing, the packet header is prepended or modified

to add the correct HDI for the destination chassis ID=2. *Id.*, 12:1-14. Thus, DaSilva teaches routing packets to the chassis specified by the chassis ID.

127. **Motivation to Combine Stroud and DaSilva.** A POSITA would have been motivated to apply DaSilva's teachings to Stroud for several reasons I will discuss below.

128. Stroud's embodiments describe a platform architecture where capacity can be expanded by adding "modular card[s], or blade[s]" to a chassis, with varying numbers of slots available for cards. Ex-1004, ¶98 ("For example, in some embodiments, chassis 50 is a 3-slot chassis, a 5-slot chassis, or a 12-slot chassis. In other embodiments, system 16 comprises a single card 54."), ¶176 ("Network testing system 16 may be configured to support any suitable number of cards or blades 54. In one embodiment, system 16 is configured to support between 1 and 14 cards 54 in a single chassis 50."). When a chassis reaches its maximum capacity, there are two ways to add additional cards/blades: either replace the chassis with a larger one or add more chassis. For example, one way to add more processing power to a 3-slot chassis would be to replace it with a 5- or 12-slot chassis. This, however, creates waste, as the 3-slot chassis is no longer used. A more common way to add capacity in networked settings, such as the one taught by Stroud, was to simply add additional chassis. For example, rather than throw away an existing 3-slot chassis, additional capacity was easily added by adding an

additional 5-slot chassis to the network. This had the advantage of higher capacity (8 slot total versus 5) without wasting the previous chassis. Stroud teaches this type of expansion, suggesting that the various subsystems implemented on the cards can be “distributed across multiple [chassis] devices of 16.” Ex-1004, ¶343; *see, e.g., id.*, Fig. 2E.

129. It was common to use multiple chassis in a networked environment, for example, in data centers. *See, e.g.*, Ex-1040, ¶2 (“An enterprise computing system is a data center architecture...[and] comprises multiple chassis...”); Ex-1014, 000014-18, 000023. A POSITA looking to scale up Stroud’s architecture to serve a larger network load would therefore look to increase the number of chassis in the platform. Moreover, since Stroud’s cards were already connected using Ethernet through a backplane (Ex-1004, ¶113, ¶176, ¶377), adding additional chassis would have simply used an Ethernet connection on the same network. Ex-1004, ¶321 (“the scalability of the FPGA pipelining and interconnected FPGA/network processor architecture utilizing Ethernet as a common internal communication channel”), ¶175 (“Ethernet may facilitate more types of communication between more types of devices than a local bus or PC.”).

130. DaSilva provides teachings for networked, multi-chassis architectures, including implementations with both single and multiple chassis, each supporting multiple cards like Stroud. Ex-1005, 7:51-8:22, 9:1-6. In particular, DaSilva

explains that when there are multiple chassis on a network, a “chassis ID” can be used to distinguish them. Ex-1005, 9:1-6. DaSilva uses a chassis ID so that the different components, in different chassis, communicate with each other. Ex-1005, 7:58-64, 9:1-6 (“The chassis ID for each network node 110 is unique and global and through the topology discovery, the network nodes 110 are aware of the chassis ID of its peer network nodes 110 in the virtual chassis system 100.”). Therefore, DaSilva provides teachings for scaling a multi-card, chassis-based architecture, such as the one taught by Stroud, to utilize multiple chassis, consistent with how data centers were often organized.

131. The references both provide motivation to combine. DaSilva provides express motivation to add hardware device information including chassis ID to existing packet formats, either by prepending the information to the header or otherwise modifying the packet header. Ex-1005, 9:58-10:14. For the first option, variations on ethernet frames were known in the art, and it would have been trivial to add a layer of processing for the prepended header as taught by DaSilva. Indeed, Stroud already teaches adding a “prepend header” to provide routing and segmentation instructions to offload/capture CLDs. Ex-1004, ¶¶530-535, ¶¶405-426. It would have been a simple matter to add an additional field for DaSilva’s chassis ID to Stroud’s prepend header. For the second option, Stroud teaches a mechanism to implement DaSilva’s teaching, e.g., for modifying the packet header

with sub-fields within the destination MAC address for various identifiers. Ex-1004, ¶¶365-372. The combination would therefore simply have added one more sub-field to the destination MAC address to hold the chassis ID. Ex-1004, ¶347, ¶¶364-372. I explained the combination above and how Stroud’s MAC address format could easily accommodate DaSilva’s chassis ID. *Supra* ¶124.

132. The references provide additional motivation to combine. Stroud contemplates “creat[ing] larger systems” (Ex-1004, ¶115) and provides express teaching, suggestion, and motivation to modify its embodiments: “[a]lthough the disclosed embodiments are described in detail in the present disclosure, it should be understood that various changes, substitutions and, alterations can be made to the embodiments without departing from their spirit and scope.” Ex-1004, ¶746. Likewise, DaSilva teaches that “[w]hile particular combinations of various functions and features of the present invention are expressly described herein, other combinations of these features and functions are likewise possible.” Ex-1005, 25:37-42.

133. A POSITA looking to add capacity to Stroud’s platform would naturally have looked to DaSilva’s teaching of using multiple chassis and having a chassis ID to distinguish them. First, the references teach similar architectures for similar purposes. Stroud features a multi-card chassis architecture for network testing and security applications, (Ex-1004, ¶81), where each card has a

management switch and is connected to a backplane (*id.*, ¶101, ¶99). Similarly, DaSilva also proposes a multi-card chassis architecture for routing network traffic over various network topologies, where each card (e.g., “network interface modules (NIMs) 152a-n, such as line cards or port modules”) has a switching module and is connected to a backplane. Ex-1005, 7:58–67. Second, Stroud and DaSilva were concerned with similar goals—providing reliable and flexible networking and data processing infrastructure. For example, Stroud discloses that “[o]rganizations are increasingly reliant upon the performance, security, and availability of networked applications to achieve business goals. At the same time, the growing popularity of latency-sensitive, bandwidth-heavy applications is placing heavy demands on network infrastructures.” Ex-1004, ¶2. Similarly, DaSilva discloses that “[o]ne of the key challenges faced by data networks is the need for network resiliency, i.e., the ability to maintain high availability despite eventual component failures, link failures or the like, which is critical to providing satisfactory network performance.” Ex-1005, 1:55–59.

134. The combination would have entailed minor additions to Stroud that would not have changed the principle of operation for either reference. Stroud taught a networked environment with multiple devices, with a chassis that used Ethernet for communications. Ex-1004, ¶116, ¶129, ¶422, ¶474, ¶¶174-176. The combination simply adds additional chassis to the network using Ethernet,

consistent with Stroud and the conventional use of Ethernet networks.

Additionally, Stroud already included several identifiers for processing elements, boards, types, interfaces, etc. in its MAC address; adding one more identifier for the chassis would have been consistent with this usage. *See* Ex-1004, ¶¶366-372. DaSilva taught a chassis ID for identifying multi-card chassis, which is precisely how the combination of Ground 1 uses DaSilva’s teachings. *See* Ex-1005, 9:58-10:14, 11:55-12:14, 5:13–18, 7:58-64, 4:53–54, Figs. 4 & 7.

135. Additionally, while the combination does not necessarily rely on virtualization, both references teach it; therefore the inclusion of virtualization—though not needed for Ground 1—would not have precluded the combination either. Stroud teaches that the platform can be implemented as a virtual infrastructure. Ex-1004, ¶84. Stroud further explains that the platform can communicate with virtual machines in data centers to evaluate the performance and resiliency in the network traffic. Ex-1004, ¶89. DaSilva teaches a “virtual chassis system” that allowed multiple physical chassis to be grouped together. Ex-1005, 3:64-4:7, 4:23-38 (“the network nodes 110 within the virtual chassis system 100 are viewed as a single logical node by external network nodes 112”). This would have been compatible with Stroud’s teachings because Stroud already contemplated implementing the chassis architecture in a virtual environment; therefore, additional virtualization would not have changed its underlying

principles. Regardless, the combination uses DaSilva's concept of a chassis ID and does not require the additional features of aggregating nodes as virtual chassis.

136. Moreover, the combination would not have adversely affected the compatibility of either Stroud's or DaSilva's teachings with other standard Ethernet compatible equipment. Stroud teaches a technique for encapsulating a packet with a prepend header inside another Ethernet frame (Ex-1004, ¶426), allowing for modified packets/frames to be processed and transmitted by standard routers and switches. Thus, a non-standard Ethernet frame with prepended headers as each taught by both Stroud and DaSilva do not create a problem, nor do the packets combined using the prepended header option. Even with the second option, Stroud's packets are used as local MAC addresses, instead of global MAC addresses, which allows customization as Stroud teaches. Modification to the meaning or interpretation of Stroud's sub-fields, or the addition of one more, would not have affected network routing using Stroud's local MAC addresses. When crossing network boundaries, a POSITA would have understood Stroud to use techniques such as encapsulation which is taught by Stroud (Ex-1004, ¶426).

137. A POSITA would have had **a reasonable expectation of success** in combining teachings from Stroud and DaSilva. First, the combination would have been simple, as explained above, with the addition of one or more chassis to the network. Adding DaSilva's chassis ID requires minimal modification of Stroud,

and as explained above would have used either a simple prepended header or one additional sub-field within the destination MAC address. *Supra* ¶124. These are minor changes that follow the direct teachings of DaSilva and Stroud. Ex-1005, 9:58-10:14; Ex-1004, ¶¶366-372. And as explained above, the combination would not have changed the principle of operation for either reference. Therefore, the combination would have yielded the predictable result of a packet header including a chassis ID, and the addition of network devices on a network would have been trivial by the relevant timeframe.

138. Indeed, Stroud teaches modular cards communicating over Ethernet for parallel processing. Ex-1004, ¶¶98-99, ¶102, ¶86. Adding more cards, either in the same chassis or over Ethernet to another chassis, would have been entirely consistent with Stroud's parallel architecture. *See* Ex-1004, ¶¶98-99, ¶102, ¶86..

139. At bottom, the combination of Ground 1 deals with basic networking concepts for Ethernet and packet headers. A POSITA with experience in data streaming and processing, and with the requisite education would have had more than enough skill to combine Stroud and DaSilva, with a reasonable expectation of success in doing so, because these concepts are taught in undergraduate courses on computer networking; they are used in lab courses which use networked computers, often a Linux setting; and they are utilized in most work or production settings involving data streaming. *Supra* §I.B.

140. For at least these reasons, limitation [1e] is taught by Ground 1.

4. Dependent Claim 2

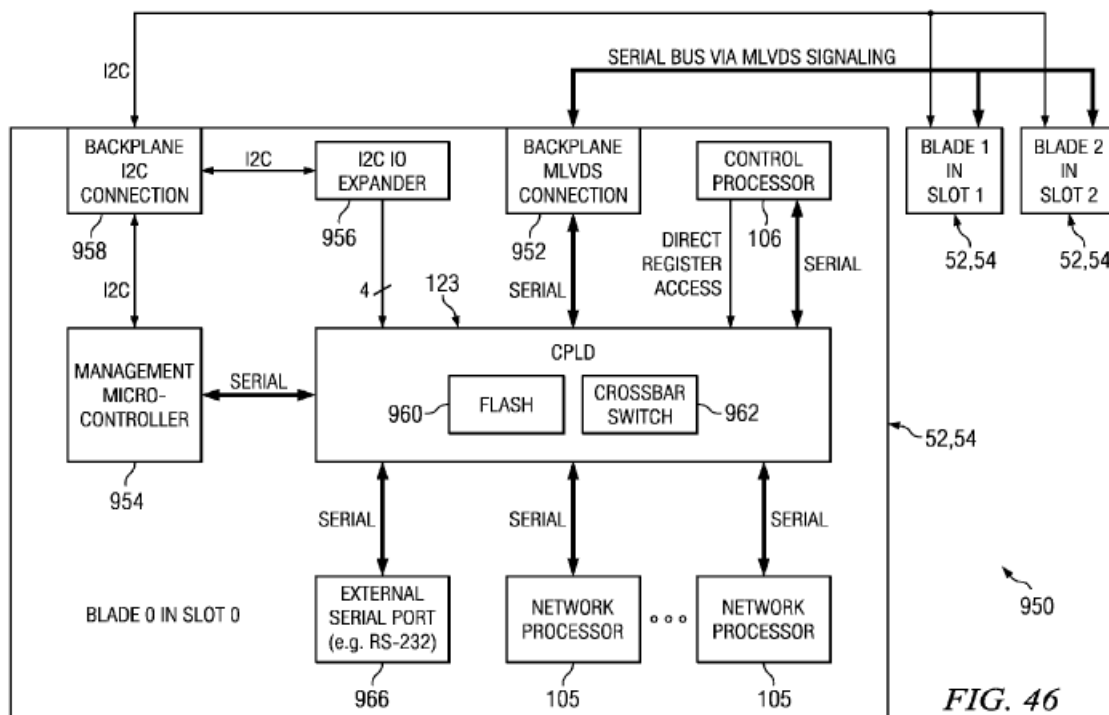
2. **The platform of claim 1, further comprising: at least a subset of the one or more swappable pods or cards having an option for direct connection to one or more pods, cards or components, bypassing the module messaging interface network.**

141. Ground 1 teaches claim 1, including the preamble. *Supra*

§§V.A.3.[1pre]-[1e]. Stroud teaches the additional limitations.

142. Stroud further teaches **at least a subset of the one or more swappable pods or cards having an option for direct connection to one or more pods, cards or components, bypassing the module messaging interface network** (e.g., Ethernet backplane interface network). For example, Stroud's cards have "an "inter-card serial bus, and an...inter-card serial port access system." Ex-1004, Abstract, ¶¶6-8, ¶320 ("[s]erial port access system 366"). Serial port access system 950 provides these two interfaces (inter-card serial bus and inter-card serial port access) to create direct serial connections for communication between the cards as seen in Figure 46. Ex-1004, ¶671 ("Thus, serial port access system 950 (a) provides each processor in system 16 *direct serial access [to] each other processor in system 16*, and (b) provides a user direct serial access to any processor in system 16,[] by way of control processor 106...."). "FIG. 46 illustrates an example serial port access system 950 of system 16 that *provides direct serial access to any processor on any card 54* in system 16 (e.g., control processors 106

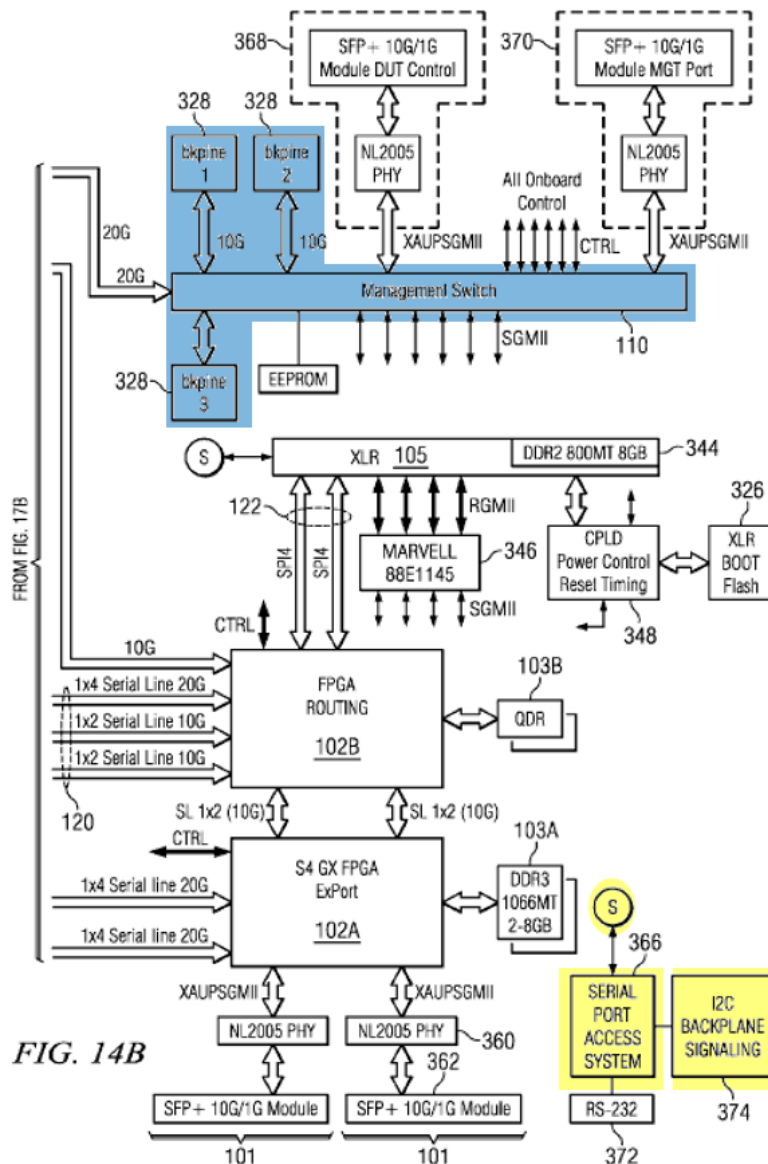
and network processors 105) via the control processor 106 on any card 54 or via an external serial port on any card 54 (e.g., when control processors are malfunctioning).” Ex-1004, ¶665, Fig. 46.



Therefore, Stroud teaches this limitation in two ways: with the I2C connection or serial bus MLVDS connection. Both provide an option for direct connection that bypasses the Ethernet network and management switches.

143. Stroud explains how the I2C and serial bus connections pass through separate connections in the backplane, distinct from the Ethernet network. “In some embodiments, backplane 56 include physical connections for connecting each card 54 directly to each other card 54.” Ex-1004, ¶101. The backplane has

an “I2C backplane signaling system 374” that “manag[es] inter-card serial connections,” allowing cards to choose which other card to be directly connected to. Ex-1004, ¶320. As shown in Figure 14B, the I2C connections crossing through the serial port access system 366 and the backplane serial bus connections (highlighted yellow below) are separate from the Ethernet-based messaging interface network accessed through the management switch 110 (highlighted blue below). Ex-1004, ¶320, Fig. 14B:



144. The I2C connection is also used to control and set up direct serial-bus connections. Ex-1004, ¶¶669-671. On receiving a request to connect from one card/blade over the I2C backplane, the receiving card/blade sets a local crossbar switch to connect to the same MLVDS path on the backplane. Ex-1004, ¶670. “This creates a direct serial connection between the requesting device on the first

blade and the target device on the second blade via the MLVDS serial bus bridging the two blades.” *Id.* Stroud illustrates this process for setting up a direct connection over the serial bus in Figure 48. Ex-1004, Fig. 48:

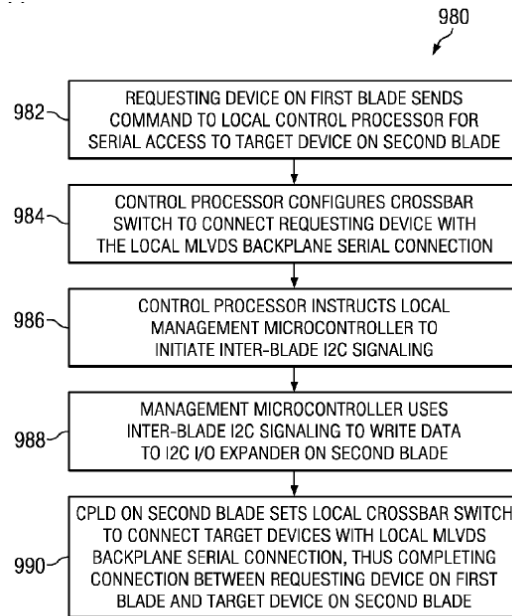
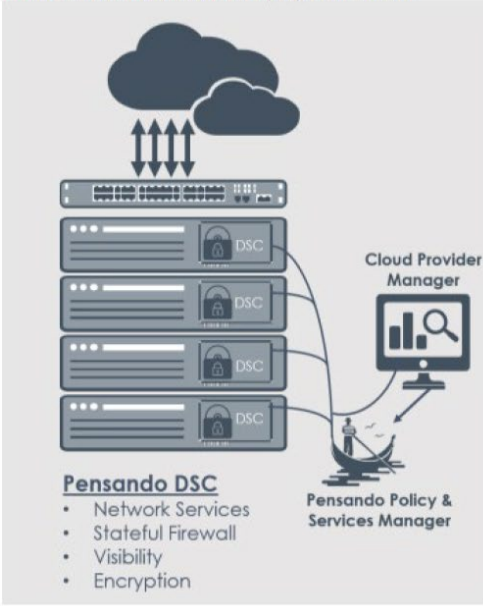
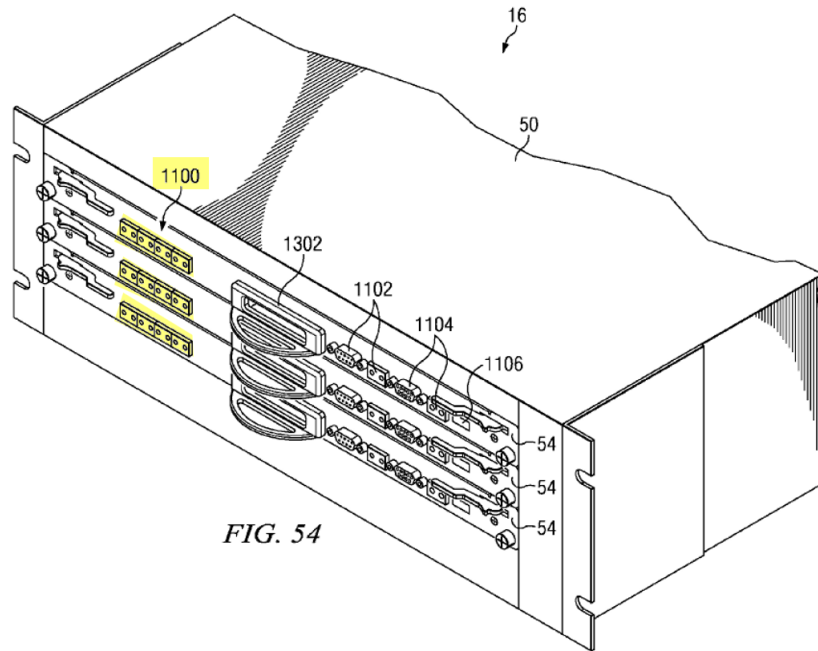


FIG. 48

145. Additionally, Stroud teaches this limitation even under Patent Owner’s apparent interpretation of the claims. In district court, Patent Owner alleged claim 2 is satisfied by “multiple [accused products] be[ing] deployed within a single server and these [accused products] are connected to each other” (Ex-1041, 000007) citing a marketing diagram of blade servers with front-panel Ethernet connections to a policy manager. Ex-1041, 000008:

Claim Language	Infringement Contentions and Exemplary Evidence of Infringement
	<i>Distributed Services Cards Deployed in Servers</i>  https://cmssc28productionstorage.blob.core.windows.net/article-assets/5eeeb987d386d2c61a5e67b0-2.pdf

146. Stroud also teaches front-panel Ethernet interfaces for its blade servers, e.g., Ethernet “test interfaces 101.” Ex-1004, ¶116, Figs. 5, 6A, 7A, 8A, 9A, 14A, 14B, 31. The ports for these interfaces are exposed on the front of the chassis when the blades/cards are installed, as shown in Figure 54. Ex-1004, ¶714 (“[E]ach card 54 has...a number of ports 1100 corresponding to test interfaces 101.”), Fig. 54:



See also Ex-1004, ¶717, Figs. 55A-59B (showing physical placement of sockets for test interfaces 101).

147. The test interfaces provide an option for the cards to directly connect with other cards or other components. As Stroud explains, “test interfaces 101 [are] for connecting testing architecture 100A to a system 18 to be tested.” Ex-1004, ¶114. They provide connections ranging from “10 Mbps 10-BaseT twisted pair copper wiring to 10 Gbps long-range fiber.” Ex-1004, ¶116. The test interfaces are used, for example, so that “packets sent by the traffic generation CLD 102C are forwarded directly to an Ethernet test interface 101 for delivery to the test system 18.” Ex-1004, ¶121, ¶125. They are also used to receive packets from test systems. Ex-1004, ¶124. Stroud further discusses how the various hardware and software modules send and receive network traffic through the test

interfaces. Ex-1004, ¶¶128-132, ¶145. As seen from the subsystem architectures in Figures 6A-9B, and the exemplary network architectures for testing in Figures 2C-2E, the test interfaces are used to communicate with other components in the systems being tested. I explain Stroud’s teachings of potential configurations and the data flow between modules in more detail for limitation [7c] below. *Infra* §V.B.2.[7c].

148. For at least these reasons, claim 2 is taught by Ground 1.

5. Dependent Claim 3

3. The platform of claim 1, wherein the one or more swappable pods or cards comprises at least one processor card having a at least one processor allocatable for offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, and executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.

149. Ground 1 teaches claim 1, including the preamble. *Supra* §§V.A.3.[1pre]-[1e]. Stroud teaches the additional limitations.

150. Stroud teaches the cards including **a processor card having at least one processor**. For example, Stroud teaches the cards/blades have a controller running software to control and configure the functions of the CLDs and network processors. Ex-1004, ¶187 (“Controller software 132 of controller 106 configured to start, stop, and post-process packet captures”). The controller is or comprises a processor, for example, a “general purpose CPU with multiple cores” or “an off-the-shelf processor designed primarily for calculations and database performance.”

Ex-1004, ¶154, ¶234 (“management processor 134”). Stroud’s blades/cards are processor cards because they have a processor (e.g., controller 106). *Compare* Ex-1004, Fig. 6A, *with* Ex-1001, Fig. 4:

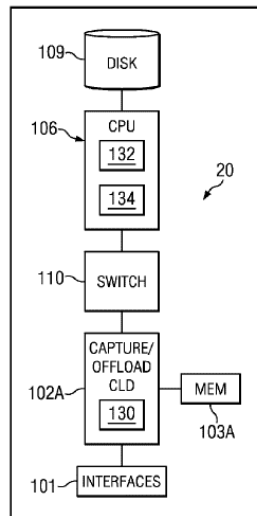


FIG. 6A

om

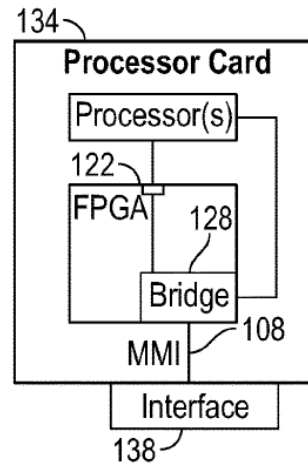


FIG. 4

Ex-1004, Fig. 6A.

Ex-1001, Fig. 4.

The controller executes a number of software modules, including a CLD server software module that handles CLD control packets. Ex-1004, ¶356 (“[C]ontroller software 132 includes a software module called a[] CLD server.”), ¶¶155-163.

151. Stroud teaches that the controller is allocatable for **offloading hardware tasks to software**. For example, Stroud teaches a platform that practices “[i]ncoming packet routing and load balancing” where packets are distributed flexibly among network processors and CLDs (e.g., FPGAs) for processing. Ex-1004, ¶¶245-248, ¶321 (“a single control processor may distribute

workloads across two additional network processors and a total of 14 FPGAs”),
Fig. 15. Stroud explains that this allows “multiple network processors 105 to be
used efficiently in a single system.” *Id.* Therefore, in view of Stroud’s load
balancing between network processors and CLDs, it would have been obvious to
offload hardware tasks to software on a network processor when the CLDs are near
capacity. *Id.* And it would have been obvious to distribute some of these
offloaded workloads to the controller because Stroud teaches using the controller
“as an auxiliary network processor.” Ex-1004, ¶154. Here, Stroud refers to the
controller as an “auxiliary” processor because the controller provides reserve
capacity when other elements (e.g., NPs and CLDs) are occupied. *See id.* This
was a common practice in network processing, where time-consuming or
complicated tasks were offloaded from hardware to CPUs to avoid clogging up the
packet pipeline. Indeed, Stroud suggests that workloads are portable between
network processors and CLDs. *See* Ex-1004, ¶296, ¶¶288-294. Stroud provides
exemplary details of its dynamic routing process in Figure 24. Ex-1004, ¶¶406-
412, Fig. 24. Therefore, Stroud teaches the controller (as an auxiliary network
processor) offloads hardware tasks that are processed by the software running on
the controller.

152. Stroud teaches the controller **orchestrating data flows**. The
“controller 106 is programmed to initiate and coordinate many of the functions of

network testing system 16.” Ex-1004, ¶104. For example the controller orchestrates data flows by directing the flow of control packets to their targets. Stroud “illustrates an example flow 470 of the life of a register access directive” in Figure 19. Ex-1004, ¶357, Fig. 19. A network processor decides “to generate 10,000 packets containing random data to be sent to a network appliance being tested for robustness under heavy load.” Ex-1004, ¶357. The directive packet is sent to the controller (e.g., CPU 106) running the CLD server, which determines a CLD to execute the command and modifies the command packet so it will go to the target CLD. Ex-1004, ¶¶358-359:

The CLD server creates a record of the directive packet and includes in that record the current time and at least the source MAC address and the sequence number of the directive packet. The CLD server modifies the directive packet as follows. The source MAC address is set to the MAC address of control CPU 106 and the destination MAC address is set to the MAC address of traffic generating CLD 102C. In some embodiments, the CLD server replaces the sequence number with its own current sequence number. In some embodiments, the CLD server may keep a copy of the entire modified directive packet to allow later retransmission.

The controller continues to monitor the status of the directive packet (e.g., whether it has timed out or a response is received from the CLD) and notifies the issuing network processor. Ex-1004, ¶360-362. The controller thus orchestrates data flow

by orchestrating the command and control packets that control data flows generated and tested by the network testing platform. *Id.*

153. Moreover, the controller orchestrates the data flows by directing the activities of the NPs and CLDs, including distributing work to them. Ex-1004, ¶321 (“a single control processor may distribute workloads across two additional network processors and a total of 14 FPGAs”). For example, the controller controls when to “start, stop, and post-process packet captures.” Ex-1004, ¶187. I explained above how the controller issues instructions to the NPs to implement modeled simulation settings. Ex-1004, ¶234; *supra* §V.A.3.[1b]. The controller similarly controls the CLDs, as I explain in the following paragraphs discussing the controller’s orchestration of hardware modules.

154. Stroud teaches the controller **orchestrating configurations of hardware modules**. For example, the “controller 106 is programmed to initiate and coordinate many of the functions of network testing system 16.” Ex-1004, ¶104. It “connect[s] to the management switch 110 to configure, boot, and manage all...CLDs 102 in the system 100A.” Ex-1004, ¶156, ¶158 (“A freshly manufactured system 100A can self-program all other system components from the controller 106.”). Stroud explains, for example, that “[c]ontroller 106 may be programmed to configure traffic generation CLD 102C, via management switch 110.” Ex-1004, ¶167. “[T]raffic generation CLD 102C” is “[l]ike other CLDs” on

Stroud's blades/cards, the configuration of which is orchestrated by the controller. Ex-1004, ¶167.

155. I explained above how Stroud's CLDs are hardware modules. *Supra* §V.A.3.[1b].

156. Stroud illustrates the process of the controller configuring CLDs during a boot up sequence in Figure 52. Ex-1004, ¶64, ¶¶691-696, Fig. 52. "At step 1066, control processor 106 communicates with each microcontroller 1052 via CLD access logic to place the CLDs in programming mode." Ex-1004, ¶692. Then, it "locates CLD image 1056" for a CLD at step 1068. Ex-1004, ¶693. "Once the corresponding CLD image has been identified, control processor 106 transfers" it to "microcontroller 1054," which "initiates a programming mode on the CLD." Ex-1004, ¶694.

157. In addition to programming the CLDs and passing data to them, the controller also has "high-level responsibilities" that orchestrate the what and how of the configuration. Ex-1004, ¶159. For example, the controller interfaces with the user to receive and effectuate user settings (e.g., how the user wants the test platform to operate) and coordinates the operation of the whole system. Ex-1004, ¶¶160-163:

1. Controller 106 serves the user-interface (web-based) used for managing the system 100A.

2. Controller 106 runs the middle-ware and server applications that coordinates the rest of the system operation.
3. Controller 106 serves the operating system and application files used by network processors 105.
4. Controller 106 hosts the database, statistics and reporting engine 162 of statistics collection and reporting subsystem 28.

Hence, Stroud teaches the controller orchestrating configuration of the CLDs.

158. Stroud teaches the controller **executing in cooperation with a sandbox** (e.g., in a CLD) **of one of the plurality of swappable pods or cards**.

For example, the controller “coordinates the rest of the system operation,” cooperating with the network processors and CLDs. Ex-1004, ¶161. The architecture of the cards supports such cooperative execution. For example, “controller 106 may have a single 10 gigabit connection to the each router CLD 102B.” Ex-1004, ¶132. The connections are “used both for inter-processor communication and for sending traffic to and from the test interfaces 101.” *Id.*

This allows the controller to execute in cooperation with the router CLD. I explained above how the controller executes in cooperation with traffic-generating CLDs. *Supra* ¶152.

159. Additionally, the controller executes “as an auxiliary network processor.” Ex-1004, ¶154. Stroud teaches several examples of network processors executing in cooperation with CLDs. *E.g.*, Ex-1004, ¶¶122-123, ¶¶128,

¶131, ¶135, ¶224, ¶236, ¶245. Therefore, when the controller functions as an NP, it executes in cooperation with CLDs. *See id.*

160. Stroud intended the controller, network processors, and CLDs to work in cooperation to construct larger test systems, explaining that “each card 54 may be configured to operate by itself, or cooperatively with one or more other cards 54, to provide any of the functionality discussed herein.” Ex-1004, ¶102. The various subsystems of the testing platform, implemented across the controller, network processors, CLDs and other components, “operate simultaneously and cooperatively within the same software and hardware platform.” Ex-1004, ¶86, ¶177 (same).

161. Stroud’s CLDs on the cards are hardware modules that include sandboxes. As I explained above, the CLDs comprise PLDs or FPGAs (Ex-1004, ¶107, ¶¶110-112, ¶¶311-313) with portions for user-definable hardware, where the user programs and alters the connection pattern of the circuits, creating custom circuits for performing custom functions in hardware. Ex-1004, ¶107, ¶¶109-112, ¶117, ¶¶311-313; *supra* §V.A.3.[1b]-[1c]. Therefore, the CLDs include sandbox portions that are designated for user-programmable electronic circuits.⁷ Stroud

⁷ I understand that Patent Owner proposed a construction of “sandbox” as “portion designated for programmable electronic circuits.” Ex-1036, 000004.

teaches the hardware modules are configured with discrete functions, such as “routing CLD 102B,” “capture/offload CLD 102A,” or “through traffic generating CLD 102C.” Ex-1004, ¶462. Each CLD functions independently and its design is isolated from other CLDs.

162. For at least these reasons, claim 3 is taught by Ground 1.

6. Independent Claim 13

163. It is my opinion that Ground 1 renders the limitations of claim 13 obvious.

[13pre] A tangible, non-transitory, computer-readable media having instructions thereupon which, when executed by a processor, cause the processor to perform a method comprising:

164. I understand that a preamble generally does not state a claim limitation. However, to the extent that Patent Owner argues that the preamble states a limitation, it is my opinion that Stroud teaches the preamble.

165. Stroud teaches that the card’s controller is connected to a memory device that stores instructions for controlling the card’s operation. For example, Stroud teaches a “Solid State Disk Drive 109.” Ex-1004, ¶169. Memory device 109 is a tangible, non-transitory, computer-readable media, such as a “solid-state drive (SSD).” Ex-1004, ¶170. SSDs were a well-known storage media, based on flash memory and commonly used for large, non-volatile storage. *Supra* §I.C. Stroud teaches removable disk storage 109 has instructions that, when executed by

the controller, configure and operate the hardware and software modules in Stroud's testing platform. Ex-1004, ¶158 ("All firmware or software used by the rest of the system 100A, except for firmware required to start the controller 106 itself (BIOS) resides on the disk drive 109."). Specifically, Stroud teaches memory device 109 has:

1. Firmware that configures the CLDs 102 and various peripherals;
2. An operating system, applications, and statistics and reporting database utilized by the controller 106; and
3. An operating system and applications used by each network processor 105.

Ex-1004, ¶¶171-173.

166. The instructions in the software and firmware cause the controller to perform a method (including configuring other modules within the system to perform steps) that further comprises the steps recited by subsequent limitations of claim 13, as I explain below.

167. For at least these reasons, [13pre] is obvious in view of Ground 1.

[13a] configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network, to implement a data flow processing architecture;
--

168. Limitation [13a] overlaps with limitations [1a] and [1d]; the overlapping limitations (bolded green and blue below) are satisfied for the same

reasons I explained previously for [1a] and [1d]. *Supra* §§V.A.3.[1a], [1d] (analysis incorporated by reference).

[13a] configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network, to implement a data flow processing architecture	[1a] one or more swappable pods or cards in one or more chassis, coupled through a messaging interface network; [1d] the plurality of swappable pods or cards being user-configurable to implement data flow processing architectures; and
--	--

169. I explained above how Stroud teaches one or more swappable pods or cards in one or more chassis (e.g., blades/cards in chassis), coupled through a messaging interface network (e.g., Ethernet backplane interface network). Ex-1004, ¶¶98-99, ¶113-115, ¶365, ¶376, Figs. 3, 54, 55A-55B, 56A-56B, 57A-57B; *supra* §V.A.3.[1a]. I also explained how Stroud teaches the cards being user-configurable to implement data flow processing architectures. Ex-1004, ¶¶106-112, ¶117; *supra* §V.A.3.[1d]. Therefore, Stroud teaches configuring those pods/cards to implement a data flow processing architecture by configuring the plurality of hardware modules (e.g., CLDs) and software modules (e.g., controller and network processors) (Ex-1004, ¶¶80-81, ¶¶63-64, ¶¶689-700, ¶¶106-107, ¶158, ¶¶170-173, Figs. 51-52; *supra* §V.A.3.[1c]).

170. For at least these reasons, limitation [13a] is obvious in view of Ground 1.

[13b] configuring user-definable hardware modules or user definable software modules in each of one or more of the plurality of swappable pods or cards;

171. Limitation [13b] overlaps with limitations [1c] and [1d]; the overlapping limitations (bolded green and blue below) are satisfied for the same reasons I explained previously for [1c] and [1d]. *Supra* §§V.A.3.[1c], [1d] (analysis incorporated by reference).

[13b] configuring user-definable hardware modules or user-definable software modules in each of one or more of the plurality of swappable pods or cards	[1c] one or more of the plurality of swappable pods or cards having a portion for user-definable hardware modules or user-definable software modules; [1d] the plurality of swappable pods or cards being user-configurable to implement data flow processing architectures; and
---	---

172. I explained above how each of Stroud’s cards includes user-definable hardware modules (e.g., CLDs) and user-definable software modules (e.g., controller and network processors), and how the cards were configured by users to implement data flow processing architectures. *Supra* §§V.A.1.[1c]-[1d]. As I explained above for limitations [1c] and [1d], Stroud teaches configuring hardware

modules via the controller (Ex-1004, ¶¶63-64, ¶¶689-700, ¶¶106-107, Figs. 51-52) and software modules by changing software (Ex-1004, ¶158, ¶¶170-173).

173. Stroud teaches configuring the modules **in each of** the cards. My previous analysis explained how Stroud’s teachings applied to each card. Additionally, Stroud teaches “[e]ach architecture” of “[e]ach card” provides the configurable hardware or software modules to provide the configurable functionality of the network testing system to test the performance and security of a target network. Ex-1004, ¶¶99-102, ¶¶80-81.

174. For at least these reasons, limitation [13a] is obvious in view of Ground 1.

[13c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user definable software modules; and
--

175. Stroud teaches limitation [13c]. Stroud teaches **communicating a data flow in accordance with the implemented data flow processing architecture**. For example, Stroud teaches a network testing application for generating “synthetic data flows” using the network testing platform, with numerous examples, e.g., “HTTP Authenticated” and “HTTP Flash Video.” Ex-1004, ¶298, Fig. 13D. The network testing includes communicating the data flow in accordance with the implemented data flow processing architecture because that

is one of the purposes of the network testing platform: to simulate the network traffic by generating and then sending a data flow (e.g., HTTP, video, and other network data flows) using the parallel architecture of the platform. *Id.*

176. Additionally Stroud’s network testing platform communicates the data flow (e.g., the simulated network traffic) as it would pass through a simulated network, based on the data flow processing architecture implemented in the platform (e.g., network traffic processing for file servers, web servers, mail servers, and database servers in data centers; firewalls; LTE networks; etc.). Ex-1004, ¶¶80-93. Some simple applications for testing include: an “application server...with multiple available services (e.g., mail, Web, SQL, and file sharing)” and a network “switch.” Ex-1004, ¶¶96-97. Stroud illustrates architectures for these network testing applications in Figures 2F and 2G. Ex-1004, Figs. 2F-2G:

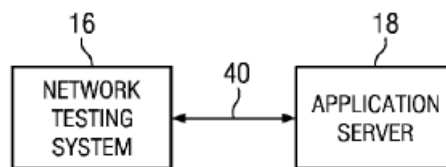


FIG. 2F

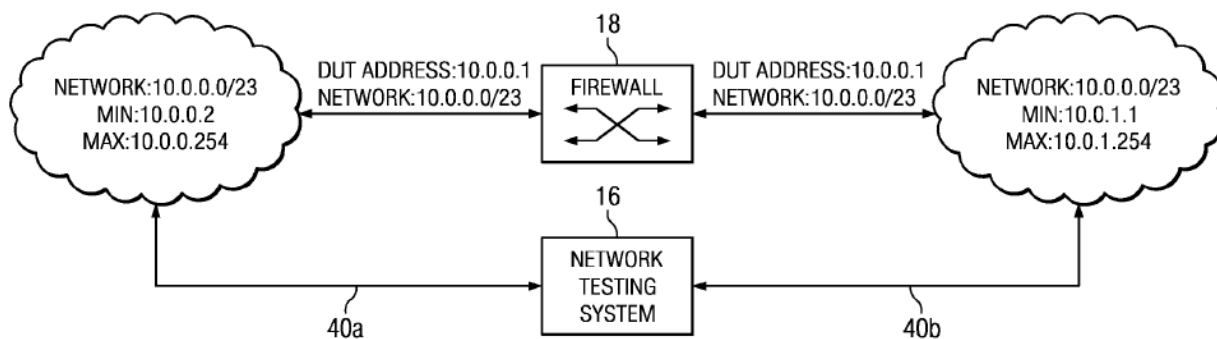
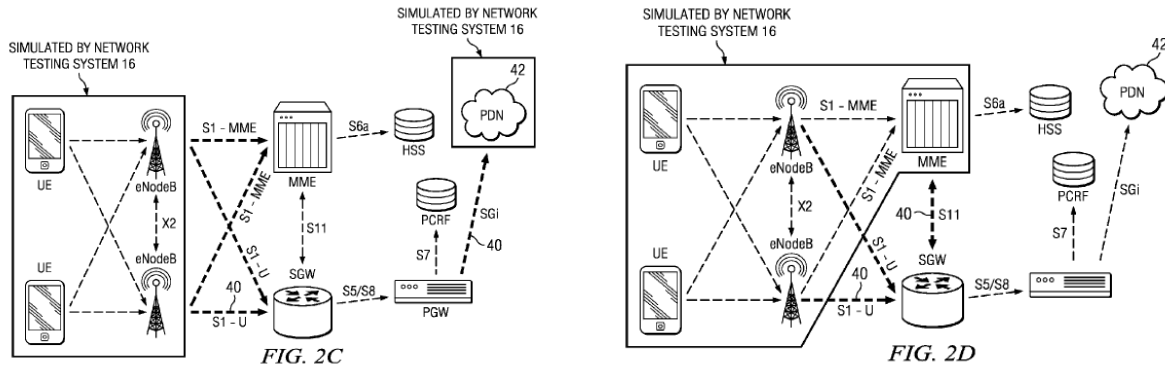
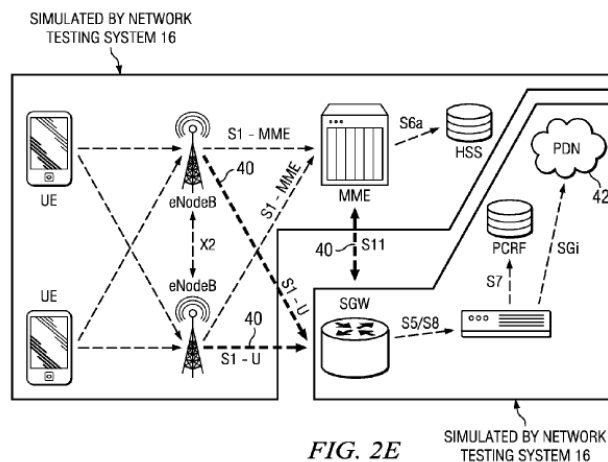


FIG. 2G

177. Figures 2C through 2E illustrate examples of more complex architectures “for testing an LTE network.” Ex-1004, ¶91. The hardware and software modules of Stroud’s test platform are configured to “simulate[] various components of an LTE network in order to test other components or communication links of the LTE network 18.” Ex-1004, ¶92. This includes simulating an architecture with two “user equipment (UEs)” (smart phones) and “eNodeB interfaces” (cell towers) as shown in Figure 2C, and architectures that further include a simulated “MME” (the mobility management entity, which handles subscriber authentication, roaming and handovers of UEs) as shown in Figure 2D. Ex-1004, ¶¶93-94, Figs. 2C & 2D:



178. The architecture of Figure 2E “simulate[s] all components of the LTE network” (including the PDN (packet data network—e.g., the Internet), PCRF (policy and charging rules function), SGW (serving gateway), and HSS (home subscriber service)) in order to test the performance of “a deep packet inspection (DPI) device, e.g., a firewall, intrusion detection or prevention device (e.g., IPS or IDS), load balancer, etc.” that is monitoring the data flows on interfaces S1-U and S11. Ex-1004, ¶95, Fig. 2E:



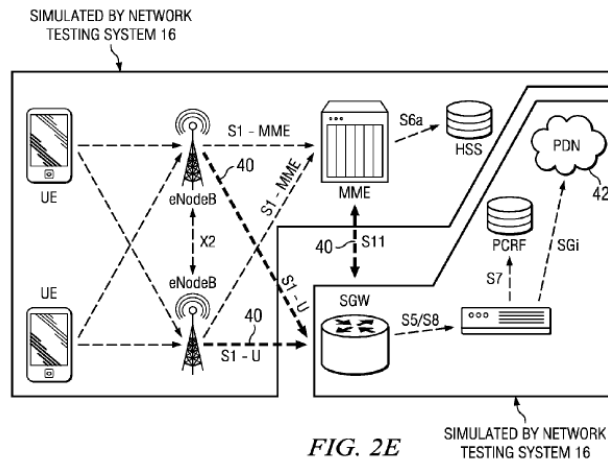
In each of these scenarios, data flows are generated and communicated in accordance with the network topology of the network architecture that is being

simulated, which was implemented using the data flow processing architecture of Stroud's platform.

179. Stroud teaches that the above-described communication and architecture are **among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards**. For example, Stroud teaches distributing workloads across hardware and software modules within a card. Ex-1004, ¶321 (“[A] single control processor may distribute workloads across two additional network processors and a total of 14 FPGAs coordinated with a single high-bandwidth Ethernet switch.”). This is part of Stroud's intended testing functionality—packets are communicated among different nodes simulating different devices on the network for processing and measuring the performance of such processing. Ex-1004, ¶¶80-81.

180. Stroud further teaches carrying out such communication across modules between different cards. Stroud's testing platform is designed in this way to provide scalable data processing capacity. Ex-1004, ¶321 (“scalability of the FPGA pipelining and interconnected FPGA/network processor architecture utilizing Ethernet as a common internal communication channel”). Indeed, Stroud even teaches distributing processing and testing architectures across cards in multiple chassis. Ex-1004, ¶343 (“The hardware devices, software, CLD configuration information, and/or firmware of each respective sub-system may

be...distributed across multiple devices of 16, as appropriate.”). Fig. 2E illustrates such an example setup where communication occurs over interfaces 21-U and S11 between two chassis (and the modules on the cards in the two chassis) that are simulating different segments of the LTE network. Ex-1004, Fig. 2E:



181. In Figures 5 and 6A through 9B, Stroud explains several data flow processing applications and provides exemplary data flow processing architectures using the modules on the blades/cards. Ex-1004, ¶178, ¶114. For example, Figure 6A depicts an architecture for a “high-speed, high-resolution network packet capture subsystem.” Ex-1004, ¶15, Fig. 6A. The controller, capture/offload CLD, and memory are configured as in Figure 6A. Ex-1004, ¶¶182-190. Stroud further explains how network packets are captured and stored in the memory under this architecture. Ex-1004, ¶¶191-192. In particular, the data flows that are captured to memory are then communicated to the controller, which stores the data to a disk. Ex-1004, ¶196, Fig. 6B. As other examples, Figure 7A depicts an architecture for

“a high-speed packet generation and measurement subsystem 22” (Ex-1004, ¶¶201-218, Figs. 7A & 7B), while Figure 8A illustrates an “application-level simulation and measurement subsystem 24” (Ex-1004, ¶¶220-250, Figs. 8A & 8B), and Fig. 9A shows an architecture for “security and exploit simulation and analysis subsystem 26” (Ex-1004, ¶¶255-266, Figs. 9A & 9B).

182. For these various data flow processing architectures, Stroud teaches that data flows are communicated among the hardware and software modules on the cards. As I explained above, the modules on Stroud’s cards carry out communication among the modules across different cards as Ethernet packets via the backplane Ethernet interface network. Ex-1004, ¶99, ¶¶113-115, ¶176, ¶377, ¶¶669-670, Figs. 3-5, 48; *supra* §V.A.3.[1a]. I also explained for claim 2, how data flows can be communicated directly between cards using the backplane serial bus. *Supra* §V.A.4 (analysis incorporated by reference).

183. The data flow processing architecture that is implemented is also among the modules on the blades/cards, as shown in the various examples and figures cited in this section. For example, the various architectures depicted in Figures 5 through 9B show the relationships between the different modules. None of these architectures would function as Stroud explains and intends, unless the modules are configured with the architecture that is defined among the modules. In short, Stroud teaches data flow processing architectures that are among the

hardware and software modules on the cards. As I explained above, Stroud's network testing platform has an architecture that is implemented among various hardware modules (e.g., routing CLD, capture/offload CLD, and through traffic generating CLD) and software modules (controller and network processors) that generate, capture, and process data flows per the architecture defined in the network testing system. *Supra* §§V.A.3.[1pre]-[1b], §V.A.3.[1d] (analyses incorporated by reference).

184. Stroud teaches the modules on the blades/cards **includ[e] the user-definable hardware modules or user-definable software modules**. As I explained above for limitation [1c], all the modules are user-definable hardware modules or user-definable software modules. *Supra* §V.A.3.[1c] (analysis incorporated by reference).

185. For at least these reasons, limitation [13c] is obvious in view of Ground 1.

[13d] communicating the data flow through the messaging interface network, comprising a network coupled to the plurality of swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.

186. For limitation [13c], I explained how Stroud teaches **communicating the data flow** among the modules on the cards. *Supra* §V.A.3.[13c] (analysis incorporated by reference). As I explained, the data flow among the modules passes **through the messaging interface network** (e.g., the Ethernet backplane interface network). *Supra* §V.A.3.[13c], §V.A.3.[1a] (analyses incorporated by reference). Limitation [13d] further overlaps with limitation [1e]; the overlapping limitations (bolded green below) are satisfied for the same reasons I explained previously for [1e]. *Supra* §V.A.3.[1e] (analysis incorporated by reference).

[13a] communicating the data flow through the messaging interface network, comprising a network coupled to the plurality of swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an	[1e] a network coupled to the one or more swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each
--	--

instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.	instance of a type of module, and each module on each board in each chassis can be addressed through the header.
--	--

187. For at least these reasons, limitation [13c] is obvious in view of Ground 1.

7. Dependent Claim 14

14.	The computer-readable media of claim 13, wherein the method further comprises: allocating at least one processor of a processor card having a plurality of processors, from among the plurality of swappable pods or cards, to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, or executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.
-----	---

188. Ground 1 teaches claim 13, including the preamble. *Supra* §§V.A.3.[13pre]-[13d]. Stroud teaches the additional limitations.

189. Claim 14 shares language with claim 3; this overlapping language (bolded green, red, and blue below) is satisfied for the same reasons I explained previously. *Supra* §V.A.5 (claim 3 analysis incorporated here by reference).

14. The computer-readable media of claim 13, wherein the method further comprises:	3. The platform of claim 1, wherein the one or more swappable pods or cards comprises at least one processor
--	--

allocating at least one processor of a processor card, from among the plurality of swappable pods or cards, to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, or executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.	card having a at least one processor allocatable for offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, and executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.
--	---

190. Stroud teaches allocating a processor (e.g., controller 106) to all four of tasks recited in claim 3. Ex-1004, ¶¶356-359, ¶154, ¶346, Fig. 6A; *supra* §V.A.5. Therefore, Stroud also teaches the limitations of claim 14.

191. For at least these reasons, claim 14 is obvious in view of Ground 1.

B. Ground 2: Claims 7 and 9-10 Are Rendered Obvious by Stroud in View of Cragon

192. It is my opinion that a POSITA would have found the Challenged Claims 7 and 9-10 obvious based on the teachings of Stroud and Cragon. Each of Stroud's cards includes multiple types of memory, including DRAM, SRAM, flash memory, hard disk, and solid-state drives, among others. Ex-1004, ¶127, ¶¶169-170. Cragon provides additional teachings on common memory access modes. Cragon explains textbook approaches for using different types of memory in computers. Ground 2 applies Cragon's teachings of common memory modes to

the various types of memory on Stroud's cards allowing for flexible and selectable arrangement. Ex-1006, 000048, 000071-73, 000219-220, 000496-497, 000255-265, 000413-415, 000536-548. The combination of Ground 2 teaches all limitations of Claims 7 and 9-10.

193. The motivation to combine and the reasonable expectation of success are explained in more detail below. *Infra* §V.B.2.[7d] (motivation to combine subsection).

1. Cragon, "Memory Systems and Pipelined Processors" ("Cragon") (Ex-1006)

194. I have reviewed the Cragon reference. I understand that Cragon was not cited or considered during prosecution of the '753 Patent, based primarily on the fact that Cragon is not cited on the face of the patent. I also have not seen the reference discussed in the prosecution history.

195. Cragon is a textbook that was published in 1996. Ex-1022, ¶¶10-12. Based on the adopted priority date of July 15, 2018 of the '753 patent, I understand Cragon is prior art to the '753 patent under at least post-AIA §102(a)(1).

196. A declaration from librarian June Munford confirms that Cragon is a printed prior art publication that was publicly available since at least December 18, 1997. Ex-1022, ¶¶11-12. The copy was available in libraries, where it was indexed and searchable. *Id.*, ¶¶11-12, 000022-23. This is confirmed by the copy held in the Library of Congress. Ex-1037, 000001, 0000007 (date stamp of "Oct

30 1995”). I have compared Exhibit 1006 with the excerpts in the copy of Cragon provided by librarian June Munford obtained from the paper copy held by Penn State University Library, and excerpts in the copy obtained from the Library of Congress. The three exhibits are consistent with each other and the pages that are present in Exhibits 1022 and 1037 are identical with the corresponding pages of Exhibit 1006 in content, although there are differences in pagination. *Compare* Ex-1006, 000001-1210, *with* Ex-1022, 000024-297; *compare* Ex-1006, 000007-30, 000073-74, 000201-228, 000255-264, 000413-418, 000491-498, *with* Ex-1037, 000006-13, 000014, 000015-28, 000029-33, 000034-36, 000037-40.

197. Cragon describes common ways in which different memory types were arranged in computer systems (Ex-1006, 000048, 000071-73), including as multi-level cache (Ex-1006, 000219-220), interleaved memory (Ex-1006, 000496-497), and other common modes of operation (e.g., Ex-1006, 000255-265, 000413-415, 000536-548).

198. Cragon is analogous art to the ’753 patent because it is in the same field of endeavor as the ’753 patent, which is data processing and streaming architectures including user-defined hardware and software modules. For example, Cragon’s disclosure relates to memory architectures that improve the performance of data processing systems. Ex-1006, 000040. Cragon is reasonably pertinent to the problem addressed by the ’753 patent (Ex-1001, 1:6-44), because it provides

various solutions for configuring memory—a flexible data processing architecture would use not one, but a variety of such memory architectures.

2. Independent Claim 7

199. It is my opinion that Ground 2 renders the limitations of Claim 7 obvious.

[7pre]. A method for data flow processing in a platform, comprising:

200. The preamble to claim 7 shares language with the preamble to claim 1; this overlapping language (bolded green below) is satisfied for the same reasons I explained previously for [1pre]. *Supra* §V.A.3.[1pre] (analysis incorporated by reference).

[7pre]. A method for data flow processing in a platform , comprising:	[1pre]. A platform for data flow processing , comprising:
--	--

201. Stroud further applies its teachings to “a method” for data flow processing in a platform. As Stroud explains, its “disclosure relates to systems and *methods* for testing communications networks, services, and devices, e.g., testing the traffic-handling performance and/or security of the network, network accessible devices, cloud services, and data center services.” Ex-1004, ¶1. Stroud teaches a platform for carrying out a method of “hardware- and software-based testing and simulation.” Ex-1004, ¶¶85-87. As I explained above, Stroud teaches a platform for data flow processing; therefore, the methods used by the platform

are also used for data flow processing; that analysis is incorporated here. *Supra* §V.A.3.[1pre].

202. Stroud's method further comprises the steps explained below. For at least these reasons, [7pre] is obvious in view of Ground 2.

[7a] configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network, to implement a data flow processing architecture;

203. Limitation [7a] overlaps with limitations [1a] and [1d]; the overlapping limitations (bolded green and blue below) are satisfied for the same reasons I explained previously for [1a] and [1d]. *Supra* §§V.A.3.[1a], [1d] (analysis incorporated by reference).

[7a] configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network, to implement a data flow processing architecture	[1a] one or more swappable pods or cards in one or more chassis, coupled through a messaging interface network; [1d] the plurality of swappable pods or cards being user-configurable to implement data flow processing architectures; and
--	---

204. I explained above how Stroud teaches one or more swappable pods or cards in one or more chassis (e.g., blades/cards in chassis), coupled through a messaging interface network (e.g., backplane interface network). Ex-1004, ¶¶98-99, ¶113-115, ¶365, ¶376, Figs. 3, 54, 55A-55B, 56A-56B, 57A-57B; *supra*

§V.A.3.[1a]. I also explained how Stroud teaches the cards being user-configurable to implement data flow processing architectures. Ex-1004, ¶¶106-112, ¶117; *supra* §V.A.3.[1d]. Because Stroud teaches configuring the plurality of hardware modules (e.g., CLDs) and software modules (e.g., controller and network processors) on the cards to implement data flow processing architectures (Ex-1004, ¶¶80-81, ¶¶63-64, ¶¶689-700, ¶¶106-107, ¶158, ¶¶170-173, Figs. 51-52; *supra* §V.A.3.[1c]), Stroud teaches configuring those pods/cards to implement a data flow processing architecture.

205. For at least these reasons, limitation [7a] is obvious in view of Ground 2.

[7b] configuring user-definable hardware modules or user-definable software modules in each of one or more of the plurality of swappable pods or cards;

206. Limitation [7b] overlaps with limitations [1b], [1c] and [1d]; the overlapping limitations (bolded red, green and blue below) are satisfied for the same reasons I explained previously for limitations [1b], [1c], and [1d]. *Supra* §§V.A.3.[1b], [1c], [1d] (analysis incorporated by reference).

[7b] configuring user-definable hardware modules or user-definable software modules in each of one or more of the plurality of swappable pods or cards	[1b] each of the one or more swappable pods or cards having one or more hardware modules or one or more software modules;
---	--

	[1c] one or more of the plurality of swappable pods or cards having a portion for user-definable hardware modules or user-definable software modules; [1d] the plurality of swappable pods or cards being user-configurable to implement data flow processing architectures; and
--	--

207. I explained above how each of Stroud’s cards includes user-definable hardware modules (e.g., CLDs) and user-definable software modules (e.g., controller and network processors), and how the cards were configured by users to implement data flow processing architectures. *Supra* §§V.A.3.[1b]-[1d]. As I explained above for limitations [1c] and [1d], Stroud teaches configuring hardware modules via the controller (Ex-1004, ¶¶63-64, ¶¶689-700, ¶¶106-107, Figs. 51-52) and software modules by changing software (Ex-1004, ¶158, ¶¶170-173). The analysis for limitations [1c] and [1d] apply here and are incorporated by reference.

208. Stroud teaches configuring the modules **in each of** the cards. My previous analysis explained how Stroud’s teachings applied to each card. Additionally, Stroud teaches “[e]ach architecture” of “[e]ach card” provides the configurable hardware or software modules to provide the configurable

functionality of the network testing system to test the performance and security of a target network. Ex-1004, ¶¶99-102, ¶¶80-81.

209. For at least these reasons, limitation [7b] is obvious in view of Ground 2.

[7c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user-definable software modules; and
--

210. Limitation [7c] recites identical language to limitation [13c]; therefore it is satisfied for the same reasons I explained previously for [13c]. *Supra* §V.A.6.[13c] (analysis incorporated by reference).

[7c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user-definable software modules; and	[13c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user definable software modules; and
--	---

211. For at least these reasons, limitation [7c] is obvious in view of Ground 2.

[7d] selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.

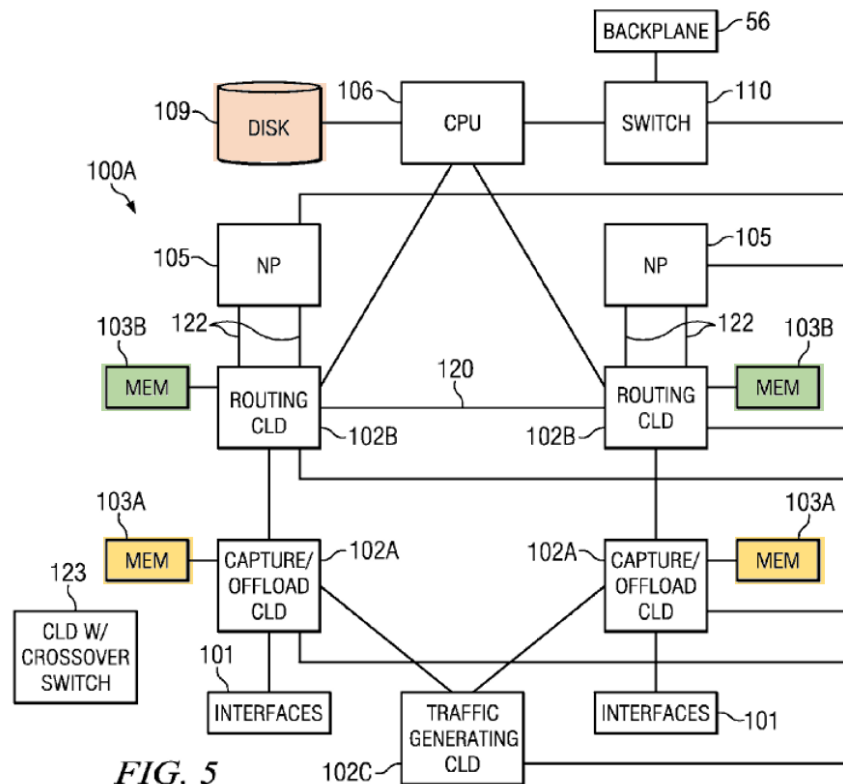
212. Ground 2 teaches limitation [7d].

213. Ground 2 teaches **selecting a memory superposition in at least one of the plurality of swappable pods or cards**. As used by the '753 patent, memory superposition refers to “two or more types of memory that can be made available for a data flow or other data processing operation.” *Supra* §IV.A. Stroud teaches this. As explained below, Stroud teaches two or more types of memory on at least one of its cards. Because Stroud uses its cards for data-flow processing, the memory on those cards is available for data-flow processing—after all, that is the functionality and purpose of Stroud’s cards. *Supra* §V.A.3.[1pre] (analysis incorporated by reference).

214. Additionally, the memory on Stroud’s cards would be available for a data processing operation because the memory is connected to processors, such as a CPU, network processors, or CLDs configured to process data, and the purpose of using memory is to make it available for data-processing operations. *See* Ex-1004, ¶120, ¶127, ¶133, ¶143, ¶186, ¶¶190-198, ¶¶306-312, Figs. 6A-9A, 14A-B.

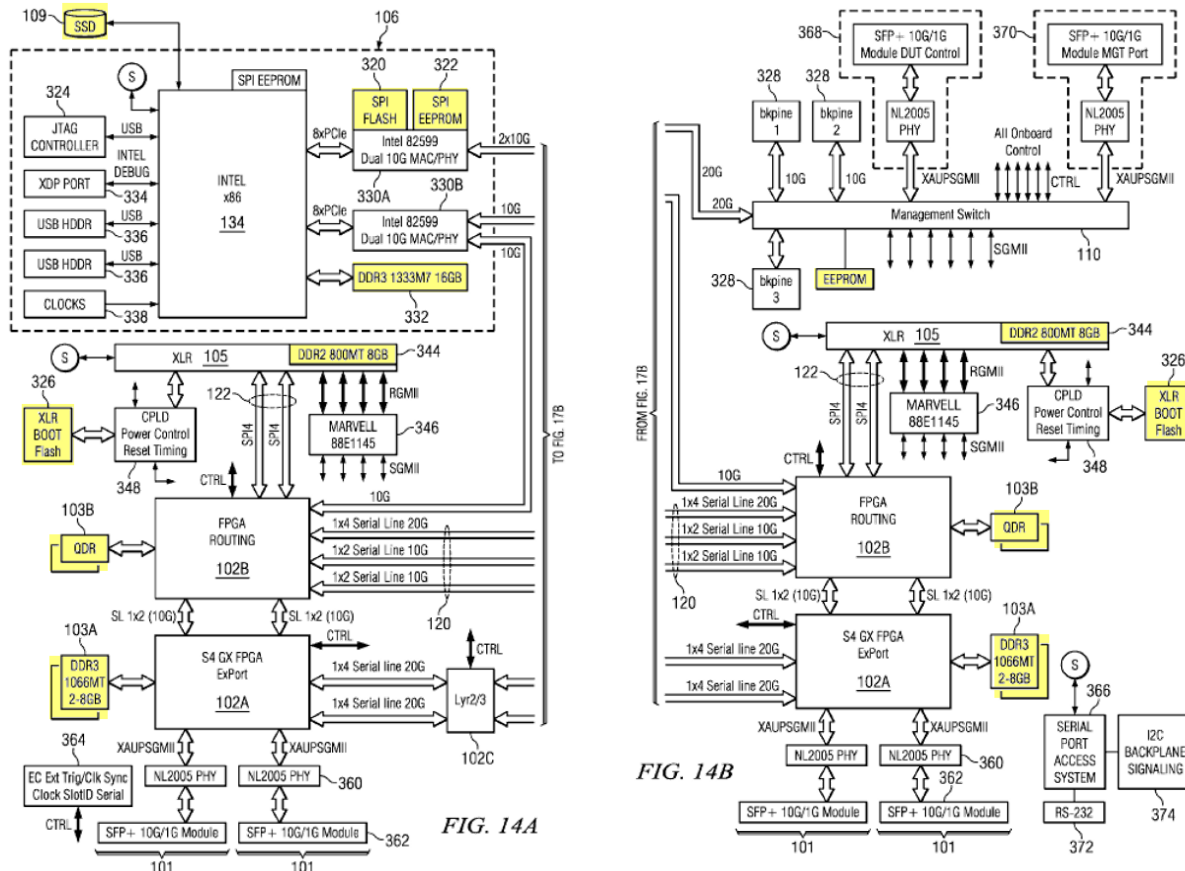
As explained further below, Ground 2 teaches how memory superposition is selected.

215. Ground 2 teaches **at least one of the plurality of swappable pods or cards having a plurality of types of memory**. For example, Stroud teaches the cards having multiple types of memory, for example memory 103A “may comprise any suitable type of memory device, e.g., DRAM, SRAM, or Flash memory, hard drive, or any other memory device with sufficient bandwidth” (Ex-1004, ¶127), while “[e]ach memory device 103B may comprise any suitable memory device” for example “high-speed, low-latency QDR (quad data rate) SRAM” (*id.*, ¶169).



216. Stroud teaches card configurations with a plurality of memory types.

E.g., Ex-1004, Figs. 14A-14B:



Id., Figs. 6A-9A. Additionally, Stroud further teaches cards with a plurality of memory types (e.g., DDR3 DIMMS, DDR2 DIMMS, and QDR SRAM) in relation to the processors and CLDs on the card for Figures 55A and 55B. Ex-1004, ¶717, Figs. 55A-55B; *see also* Figs. 56A-59B.

217. Ground 2 teaches or suggests **a selectable arrangement of the plurality of types of memory** by applying Cragon's teachings to the various types of memory on Stroud's swappable cards. Cragon teaches textbook arrangements,

explained below, for different types of memory in computers, and Cragon teaches that the arrangement of memory—for example, the types of memory assigned to different address ranges—can be selected based on the parameters of a workload. Ex-1006, 000413-414, 000536-548.

218. Applied to Stroud, which teaches a plurality of types of memory on each card, the combination therefore teaches a selectable arrangement of those memories, depending on the memory needs of a particular application. Since Stroud aims to provide a flexible platform with user-definable hardware and software modules that can be configured to perform a wide variety of applications (*see, e.g.*, Ex-1004, ¶321, ¶¶106-113, ¶343, ¶373; *supra* §V.A.3.[1b], §V.B.2.[7b]), it would have been obvious to select between common memory arrangements taught in textbooks such as Cragon to suit the particular application of a card. For example, Stroud's hardware modules include CLDs (e.g., FPGAs) with hardware circuitry the user can alter into custom functionality, including the ways in which the CLD uses and interacts with memory. Ex-1004, ¶¶106-112, ¶¶117-120, ¶¶311-313, ¶¶689-700, , Figs. 51-52. Different functions have different memory requirements, and Cragon teaches several conventional memory arrangements that were used for different purposes, e.g., depending on capacity, bandwidth, service rate, response time, and cost requirements. Ex-1006, 000536-548. Therefore, the combination teaches or suggests a selectable arrangement of the plurality of types

of memory on Stroud's card based on the needs of the application running on the card, particularly in view of the flexible, reconfigurable nature of CLDs. *See id.*

219. Ground 2 teaches the selectable arrangement including the particular arrangements listed in limitation [7d], addressed in sequence below.

220. **Multilevel cache.** Ground 2 teaches arranging memories as a multilevel cache. For example, Cragon teaches a multilevel cache with two levels (L1 and L2). Ex-1006, 000219 (“The idea at work with a multilevel cache is that a small and fast first-level cache is used and that the second-level cache can (1) reduce the transport time to the first-level cache and (2) reduce the bus loading.”), 000219-220 (“a simple two-level unified cache system”), Fig. 2.27 at 000220:

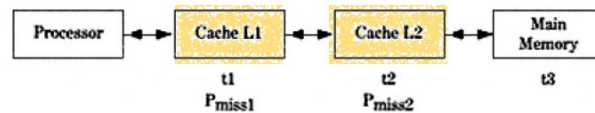


Figure 2.27
Two-level cache.

221. As Cragon explains, L1 cache comprises small and fast memory close to the processor core. Ex-1006, 000219. Stroud further teaches SRAM on the card. Ex-1004, ¶127, ¶169. The second level cache is larger and slower than L1; therefore, it would have been obvious to use other, slower types of memory on the card. Ex-1004, ¶127. Multilevel cache was known in the art as an effective way to accelerate access to recently- and frequently-used data, leveraging the speed of small, fast caches and the larger storage of lower-level caches. *See* Ex-1006,

000067, 000073. This had been widely used in the art for decades and was a textbook memory arrangement by the 1990s, as evident from Cragon. Therefore, a POSITA would have found it obvious and been motivated to use multiple types of memory (on-chip or on-card) in a multilevel cache.

222. **Interleaved memory.** Ground 2 teaches the plurality of memory types selectively arranged as interleaved memory. For example, Cragon teaches “high-order *interleaving*” in which different types of memory are alternately accessed to achieve an average access bandwidth. Ex-1006, 000496-497. Here, Cragon interleaves (alternates) access to different types of memory: as Cragon explains, “[h]igh-order interleaving...permits the easy *intermixing* of different memory technology such as a combination of RAM and ROM....” *Id.*, 000497. When Cragon’s teachings are applied to Stroud, it would have been obvious to arrange the multiple types of memory (on-chip or on-card) as high-order interleaved memory. *See id.* This was a textbook technique known since the 1990s for blending the characteristics of different memory types, for example blending size and speed characteristics. *See id.* Additionally, high-order interleaving allowed “for easy memory expansion.” Ex-1006, 000497.

223. Cragon teaches that “[i]nterleaved memories...distribut[e] the read or write accesses over a number of memory modules, which is a form of parallelism.” Ex-1006, 000491. Therefore, for modules in high-order interleaving that are

different types of memory arranged to be alternately accessed, this results in an access bandwidth that averages that of the different memory types. *See id.* For example, using simple numbers to illustrate the concept taught by Cragon, accessing DRAM at 10 MB/s and then flash memory at 2 MB/s would result in an average access bandwidth of 6 MB/s, with the access times being equal.

Therefore, when Cragon's teachings are applied to Stroud with high-order interleaving of multiple types of on-chip and on-card memory (e.g., the plurality of types of memory, including SRAM, on-chip DRAM, external DRAM, and disk storage, as explained above), the resulting accesses, when averaged out over multiple accesses, would achieve an average access bandwidth because the access bandwidth for individual accesses would vary by module but the read and write accesses would be distributed across the memory types. Ex-1006, 000491.

224. As Cragon explains, the module and word addresses for interleaved memory are calculated "by dividing the memory address by the address interleaving degree n ." Ex-1006, 000495. For high-order interleaving, "the quotient selects the module and the remainder selects the word within the module." Ex-1006, 000496-497.

225. To the extent PO argues for a plain and ordinary meaning of "interleaved memory," Cragon literally teaches "[i]nterleaved memories." Ex-1006, 000491. In other words, if PO argues different types of memory are not

required, they are also not precluded. *See also* Ex-1006, 000491-496, Fig. 5.1 at 000495.

226. **Swapped memory over a specified address range.** Ground 2 teaches the plurality of types of memory types selectively arranged as swapped memory over a specified address range. For example, Cragon teaches “swapping partitions in and out” over a specified address range. Ex-1006, 000257. To accomplish this, Cragon teaches techniques for “virtual memory” (Ex-1006, 000255, 000264-265), which provides a “virtual address space” where “the physical highspeed memory is much smaller than the virtual address space.” Ex-1006, 000051. Data for certain address ranges are moved from main memory (e.g., DRAM) to non-volatile storage (e.g., disk) when not in use, and then “moved from disk to the main memory” when needed. Ex-1006, 000256. Cragon teaches assigning a specific address range to a partition, with partitions swapped in and out between primary and secondary (e.g., non-volatile) memory. Ex-1006, 000257:

Given that real memory is large enough to hold a number of partitions, *each partition is allocated its own address space in real memory.* The monitor then manages these partitions, assigning space, adjusting addresses, and swapping partitions in and out. As jobs are completed, space in memory is vacated and the monitor attempts to load another partition.

227. Cragon further describes the technique of “paged allocation” which generalizes the technique of partitioned allocation. Ex-1006, 000262. Cragon explains that the address space is split into regularly sized pages, for example, “4 Kbytes.” *Id.* The partitions and the address space of the real memory are likewise subdivided into equally sized pages, allowing for individual pages to be swapped in and out. Ex-1006, 00265 (“Paging is the process o[f] moving pages back and forth between real and auxiliary memory.”); 000352 (“page swap”). That means each page is swapped in and out for a specific address range at 4 kilobyte (or other size) intervals. Ex-1006, 00265 (“Page is an allocation of fixed-length aggregations of addressable units.”). The address ranges in this example would be 0-4KB, 4-8KB, 8-12KB, and so on. Therefore, Cragon teaches swapping memory with specified address ranges.

228. When Cragon’s teachings are applied to Stroud, the combination teaches a plurality of memory types (on-chip or on-card) as swapped memory over a specified address range. For example, a POSITA would have found it obvious to use DRAM as the main physical memory and disk as non-volatile storage for the swap file (*see* Ex-1004, ¶308, ¶195, ¶196, ¶269, Figs. 5-10, 14A-B, 20, 23, 30, 49). This aligns with the textbook arrangement for swapped memory.

229. Alternatively, a POSITA would also have found it obvious and been motivated to use other memories on Stroud’s card for swapped memory, including

other DRAM or SRAM (Ex-1004, ¶127, ¶169), with the disk as the non-volatile storage for a swap file as Cragon teaches. Swap files and virtual memory were commonly used to expand the amount of memory that a processor could utilize. These techniques were widespread and taught in textbooks by the 1990s. Ex-1006, 000255-257, 000264-265.

230. Memory allocated to specified address ranges according to parameterized task needs. Ground 2 teaches the plurality of memory types selectively arranged as memory allocated to specified address ranges according to parameterized task needs. First, Stroud teaches assigning specified address ranges to various types of memory based on their intended use (e.g., configuration memory, internal working memory, external memory). Ex-1004, ¶352. Second, Cragon teaches memory allocated to specified address ranges in memory space for each task, so that “[o]ne task should not be able to interfere with another task.” Ex-1006, 000413. “Bounds registers” are used to demarcate the “upper address limit” and the “lower address limit” of memory allocated to a task. Ex-1006, 000413-414. “For a system that will have a number of tasks resident in memory at one time, there may be a number of bounds register pairs, one for each task.” Ex-1006, 000414. In this configuration, the memory space is allocated “within a contiguous address space” between the pointers to the upper and lower address

limits. *Id.* Thus, Cragon teaches allocating memory to a specified range of addresses in the contiguous space between the bounds register pairs.

231. Cragon also teaches how this task-based allocation is implemented using paged memory systems. For paged memory, “the segment length field of a segment table operates in the fashion of a bounds register.” Ex-1006, 000414-415. That is, the address range is specified with pages, in multiples of the page size, and thus coincide with the address range of the specified pages. I explained above how pages, when using paged memory, correspond to specified address ranges. Cragon further teaches specifying the memory space allocated to a task with a “distinct address space” that is marked by “the task name in the most significant bits.” Ex-1006, 000416-417 (“This protection is accomplished by making the task name a part of the virtual address.”).

232. Additionally, Cragon teaches allocating memory to a task, and therefore the specified address range for the task, according to parameterized task needs. Cragon teaches several parameters for task needs, including for example, disk storage used for interleaving or swapping, to meet “bandwidth and storage *requirements based upon workloads.*” Ex-1006, 000536. Cragon teaches five commonly-used parameters for task needs: “[c]apacity,” “[b]andwidth,” “[s]ervice rate,” “[r]esponse time,” and “cost.” Ex-1006, 000536. Cragon explains differences in capacity needs for tasks in mainframe and single-user computers.

Ex-1006, 000536-539. With regards to bandwidth, Cragon teaches higher bandwidth requirements relative to the rate in which instructions are processed for “interactive processing” (typically on the order of “4 to 8 times”) in comparison to “scientific processing.” Ex-1006, 000540-541. Cragon provides empirical data on differing bandwidth needs. Ex-1006, 000541 (discussing “0.73 bits per instruction” for mixed use and “0.32 bytes per instruction” for an atmospheric simulation model). Additionally, Cragon teaches “service rate” as the “amount of work completed in a specific interval of time.” Ex-1006, 000548. In practice, the service rate is determined in part as a “combination of latency, bandwidth, and scheduling.” Ex-1006, 000548. Processing video data to be displayed on-screen requires regular access to memory to keep up with the video frame rate and display refresh rate. On the other hand, background processes that store and retrieve long term data for occasional processing have lower requirements for bandwidth, service rate, and response time. A POSITA would have known of these task parameters, as they were basic aspects of computer memory and taught in textbooks by the 1990s. Cragon teaches that these parameters were equally important for other types of memory and used to organize the memory hierarchy. Ex-1006, 000048-63 (“Starting with the registers in the processor, the storage capacity in each level increases and the speed and the cost per bit decrease.”). The varying performance between “main memory, and bulk storage” was known since

the 1940's. Ex-1006, 000048. For example, Cragon suggests the following parameters for memory assigned to different types of use cases. Ex-1006, 000054:

Level	Access Time	Access Time Ratio
Cache	50 ns	1
Main Memory	200 ns	4
Disk	20 ms	100,000

See also Ex-1006, 000041-63.

233. In summary, Stroud teaches several different types of memory on a card, explained above, while Cragon teaches allocating memory that is suitable to a particular task based on common parameters for task needs. Cragon further teaches allocating memory for a task to a specified address range. Therefore, the combination teaches this limitation.

234. **Motivation to Combine.** Stroud seeks to provide a platform architecture that is scalable—with capacity that can easily be expanded by adding more cards or chassis—and flexible—with software-programmable network processors (for custom software) and user-definable hardware CLDs (for custom hardware). Ex-1004, ¶382 (looking to “improve the scalability and reliability of the system”), ¶321 (“This embodiment illustrates the scalability of the FPGA pipelining and interconnected FPGA/network processor architecture utilizing Ethernet as a common internal communication channel.”), ¶399 (“The generalized architecture characteristics of the embodiments of the present disclosure enable

allows flexible internal routing of received network messages”), ¶493, ¶128, ¶144. For additional flexibility, each of Stroud’s cards includes a controller CPU. *Id.*, ¶688. Therefore, a POSITA would have been motivated to provide flexible memory arrangements to support Stroud’s flexible architecture.

235. Cragon furthers Stroud’s stated goals of scalability and flexibility by teaching different ways for memory to be utilized. Cragon is a textbook that describes common memory modes that were used in computers. Cragon’s memory arrangements had been widely used since the 1990s, as evident from the fact that Cragon is a textbook published in the 1990s. *See, e.g.*, Ex-1006, 000219-225, 000255-431, 000491-597. These memory arrangements, described earlier in this section, were taught in undergraduate courses, often in the context of computer architecture and memory hierarchies; as Cragon explains, various types of memory such as “main memory, and bulk storage on magnetic wire recorders” were part of the memory hierarchy by the 1990s. Ex-1006, 000048. A POSITA would have been motivated to use each of Cragon’s memory arrangements for its known benefits. Given that Stroud teaches a flexible, scalable platform, a POSITA would have been motivated to apply those teachings to the use of common memory modes and to select between them to suit a particular application, as explained above.

236. This would have been a combination of prior art elements according to known methods to yield predictable results—namely, the use of Cragon’s basic memory arrangements with Stroud. Stroud teaches cards with a variety of different memory types. Ex-1004, ¶127 (“any suitable type of memory device, e.g., DRAM, SRAM, or Flash memory, hard drive, or any other memory device with sufficient bandwidth”), ¶169 (“each memory device 103B may comprise high-speed, low-latency QDR (quad data rate) SRAM memory”). These different types of memory were arranged in different ways to meet the particular applications implemented on Stroud’s cards. Ex-1004, ¶717, Figs. 55A & 55B. Stroud seeks to maximize and optimize memory performance (*see, e.g.*, Ex-1004, ¶127 (“data may be written at near maximum-theoretical rates to maintain an accurate representation of all packets that arrived on the network, within the limits of the amount of available memory.”)), and a POSITA looking to implement Stroud’s network simulation platform with different memory types would naturally have looked for references that provide the details for using different types of memory together. Cragon teaches these details. Ex-1006, 000066.

237. Additionally, a POSITA would have been motivated to improve Stroud’s memory systems as recognized in the art. Ex-1028, ¶2 (“Finding ways to improve system memory is therefore a motivation of computing system engineers”). The close fit between Stroud’s memory teachings and Cragon’s

memory teachings would have drawn a POSITA to look to Cragon. Stroud discusses the allocation of different address ranges to different types of memory. Ex-1004, ¶352 (“Some CLDs have a flat memory space where a range of addresses corresponds to CLD configuration data, another range of addresses corresponds to internal CLD working memory, and yet another range of addresses corresponds to external memory connected to the CLD such as quad data rate static random access memory (QDR) or double data rate synchronous dynamic access memory (DDR).”). Cragon provides the finer implementation details such as the use of translation tables and bound registers to track the specified memory ranges. Ex-1006, 000413-415. In short, Cragon teaches the “how” to do what Stroud asks for in terms of memory configuration. And a POSITA would have appreciated the benefits from Cragon’s teachings of selecting between different memory arrangements based on application needs. Therefore, a POSITA would have been motivated to apply such techniques to improve the system’s memory.

238. Furthermore, the combination would not have changed the principle of operation for either reference because it simply applies Cragon’s teachings to uses memory in the common and conventional ways in which memory had been used in computers since the 1990s. Stroud’s cards and chassis would still have been used for a network testing platform, as Stroud teaches. Ex-1004, ¶¶85-86.

Cragon's teachings would likewise be used as Cragon teaches—which had long been conventional.

239. **Reasonable Expectation of Success.** A POSITA would have had a reasonable expectation of success in applying Cragon's teachings to Stroud. The addition of Cragon's teachings simply select between conventional memory modes, which had been known and used for decades. *See, e.g.*, Ex-1006, 000219-225, 000255-431, 000491-597. The combination provides routine details that a POSITA would have looked for in textbooks such as Cragon. Moreover, Stroud teaches CLDs, such as FPGAs, on its cards, and a POSITA would have been more than capable of implementing CLD/FPGA designs to use the memory on Stroud's cards according to the textbook arrangements taught by Cragon. *Supra* §III (explaining the level of ordinary skill in the art).

240. A POSITA would have been more than capable of using memory in these basic ways. Cragon's teachings reflect common curriculum material for computer architecture courses and memory hierarchies. A POSITA would have been able to implement Cragon's memory teachings based on undergraduate education and also work experience, given the prevalence and basic nature of Cragon's teachings. Indeed, the same techniques taught by Cragon were being employed in systems with various memory types. *See, e.g.*, Ex-1032, Abstract, Figs. 1A, 1B, 2; Ex-1033, Abstract. Implementing Cragon's memory teachings

would have been straightforward and well within the level of ordinary skill. *Supra* §III.

3. Dependent Claim 9

9. The method of claim 7, wherein the communicating the data flow comprises: bypassing the messaging interface network and communicating through a direct connection among two or more pods, cards or components.

241. Ground 2 teaches claim 7, including the preamble. *Supra* §§V.B.2.[7pre]-[7e]. Claim 9 shares language with claim 2; this overlapping language (bolded green and blue below) is satisfied for the same reasons I explained previously. *Supra* §V.A.4 (claim 2 analysis incorporated by reference).

9. The method of claim 7, wherein the communicating the data flow comprises: bypassing the messaging interface network and communicating through a direct connection among two or more pods, cards or components.	2. The platform of claim 1, further comprising: at least a subset of the one or more swappable pods or cards having an option for direct connection to one or more pods, cards or components, bypassing the module messaging interface network.
--	---

242. Ground 2 further teaches communicating data flows through a direct connection between Stroud’s cards or components. For example, Stroud teaches several direct connections between cards, including direct serial connections. *Supra* §V.A.4. The purpose of having direct serial connections between Stroud’s

cards is to allow the cards (and the network processors and CLDs on the cards) to communicate between each other. The serial connections that I explained above for claim 2—especially the serial bus which provides high-speed MLVDS connections—provide alternative paths to communicate the data flows of the network traffic being generated and processed for simulation. Therefore, Stroud teaches claim 9 for the same reasons I explained above for claim 2. *Supra* §V.A.4; Ex-1004, ¶101, ¶320, ¶¶669-671, Abstract, ¶¶6-8, Fig. 46 & 14B.

243. Additionally, Stroud teaches this limitation even under Patent Owner's apparent interpretation of the claims, which allege front-panel Ethernet connections to be direct connection in the claims. *Supra* §V.A.4 (claim 2 analysis incorporated by reference); Ex-1041, 000007-8. Stroud teaches front-panel Ethernet connections, which satisfy claim 9 for reasons I explained previously. *See id.*; Ex-1004, ¶¶114-116, ¶714, ¶717, Figs. 5, 6A, 7A, 8A, 9A, 14A, 14B, 31, 54, 55A-59B. It would have been obvious to send data flows through the front panel Ethernet, in addition to or instead of the back, because Stroud explains how the front panels are for testing external networks, and Stroud teaches some data flows used for this purpose. *See* Ex-1004, ¶¶114-116, ¶¶121-125, ¶¶128-132, ¶145, Figs. 2C-2E, 6A-9B; *supra* §V.B.2.[7c] (explaining data flows).

244. For at least these reasons, claim 9 is obvious in view of Ground 2.

4. Dependent Claim 10

**10. The method of claim 7, further comprising:
allocating at least one processor of a processor card, from among the plurality of swappable pods or cards, to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, or executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.**

245. Ground 2 teaches claim 7, including the preamble. *Supra*

§§V.B.2.[7pre]-[7e]. Claim 10 shares language with claim 3; this overlapping language (bolded green, red, and blue below) is satisfied for the same reasons I explained previously. *Supra* §V.A.5 (claim 3 analysis incorporated here by reference).

10. The method of claim 7, further comprising: allocating at least one processor of a processor card, from among the plurality of swappable pods or cards, to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, or executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.	3. The platform of claim 1, wherein the one or more swappable pods or cards comprises at least one processor card having a at least one processor allocatable for offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, and executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.
---	---

246. Stroud teaches allocating a processor (e.g., controller 106) to all four of the tasks recited in claim 3. Ex-1004, ¶¶356-359, ¶154, ¶346, Fig. 6A; *supra* §V.A.5. Therefore, Stroud also teaches the limitations of claim 10.

247. For at least these reasons, claim 10 is obvious in view of Ground 2.

C. Ground 3: Claims 7 and 9-10 Are Rendered Obvious by Stroud in View of Cragon and Saulsbury

248. It is my opinion that a POSITA would have found the Challenged Claims 7 and 9-10 obvious based on the teachings of Stroud, Cragon and Saulsbury. Ground 2 teaches each limitation of claim 7, as I explained above. To the extent Patent Owner argues that it would somehow not have been obvious to select an appropriate memory arrangement for Stroud's cards, Saulsbury provides additional teachings for selecting between memory arrangements.

249. Each of Stroud's cards includes multiple types of memory, including DRAM, SRAM, flash memory, hard disk, and solid-state drives, among others. Ex-1004, ¶127, ¶¶169-170. Cragon teaches common modes for using different memory types. Saulsbury provides additional teachings for selecting between memory modes, for example using the memories as a hierarchal multi-level cache in one mode or directly-accessible memory in another mode. Ex-1007, Abstract. Since Stroud provides a flexible platform with programmable software modules and hardware modules, Ground 3 provides a flexible memory arrangement to suit the needs of the platform by combining Stroud with Saulsbury's teachings.

250. The motivation to combine is explained below. *Infra* §V.C.2.[7d]
(Motivation subsection).

**1. U.S. Patent Application Publication No. 2002/0087821
("Saulsbury") (Ex-1007)**

251. I have reviewed the Saulsbury reference. I understand that Saulsbury was not cited or considered during the prosecution of the '753 patent, based primarily on the fact that Saulsbury is not cited on the face of the patent. I also have not seen the reference discussed in the prosecution history.

252. Saulsbury was published July 4, 2002. Ex-1007, (43) Pub. Date. Therefore, I understand Saulsbury is prior art under at least AIA 35 U.S.C. §102(a)(1) because it was published before the effective filing date for the '753 patent of July 15, 2018.

253. Saulsbury teaches using memory in selectable modes of operation. For example, Saulsbury teaches a "computer processing architecture with on-chip DRAM usable as physical memory or cache memory." Ex-1007, (54) Title. Saulsbury "control[s] the mode" of the DRAM to operate as either directly accessible physical memory or cache memory. *Id.*, Abstract, ¶¶67-68 (mode controlled with a "control input bit 32"), Fig. 1:

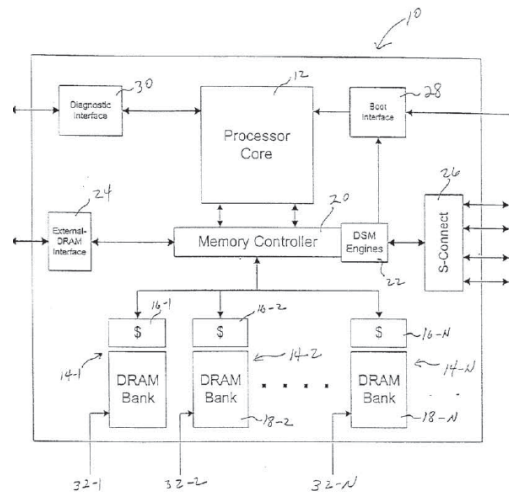


Fig. 1

254. Saulsbury is analogous art to the '753 patent because it is in the same field as the '753 patent, which is data processing and streaming architectures including user-defined hardware and software modules. Saulsbury is directed to a memory architecture for use in a “scalable computer processing architecture.” Ex-1007, ¶8, ¶20, ¶¶22-26. In particular, Saulsbury supports the flexible selection of memory operation modes. Ex-1007, Abstract, ¶3, ¶5-6, ¶¶67-70, Figs. 1 & 5. Saulsbury is reasonably pertinent to the problem addressed by the '753 patent (Ex-1001, 1:6-44), because it provides a solution for flexibility in memory configuration: Saulsbury presents memory architectures with selective operation of memory in different configurations. Ex-1007, Abstract, ¶3, ¶5-6, ¶8, ¶¶67-70, Figs. 1 & 5.

2. Independent Claim 7

[7pre]-[7c] ...

255. These limitations are taught by Stroud and Cragon for reasons explained above. *Supra* §§V.B.2.[7pre]-[7c] (analyses incorporated by reference).

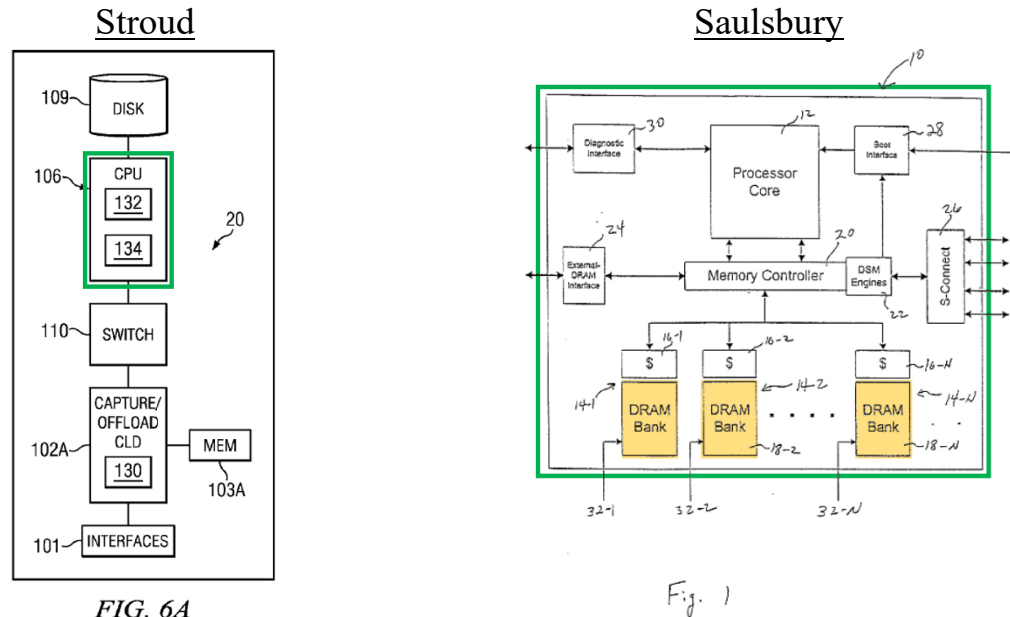
[7d] selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.

256. The combination of Stroud and Cragon teaches limitation [7d], as I explained for Ground 2. That analysis applies here. *Supra* §V.B.2.[7d] (analysis incorporated by reference). Saulsbury provides additional teachings, as explained below.

257. Ground 3 teaches **at least one of the plurality of swappable pods or cards having a plurality of types of memory**. As I explained above for Ground 2, Stroud teaches that each of its cards has multiple types of memory. *Supra* §V.B.2.[7d]; Ex-1004, ¶127, ¶169, Figs. 5, 14A-14B, ¶717, Figs. 55A-59B.

258. The combination of Stroud and Saulsbury further teaches a plurality of types of memory. Stroud teaches each card having a CPU and memory, including for example, a controller CPU connected to a “disk,” such as a solid state disk. Ex-1004, ¶127, ¶170, Figs. 5, 6A, 7A, 8A, 9A, 14A-B. Saulsbury teaches

adding on-chip DRAM to a CPU. Ex-1007, ¶¶65, ¶¶2-3, ¶¶5-8, Fig. 1. When applied to Stroud, this combination results in a plurality of types of memory on the card (e.g., disk, on-chip DRAM, and other capture/buffer memories). The motivation to combine is explained below.



259. Ground 3 teaches **a selectable arrangement of the plurality of types of memory**. Saulsbury provides additional teachings for selecting a memory superposition in cards having a multiple types of memory with a selectable arrangement of those memories. Saulsbury teaches selecting between different memory arrangements; as an example, Saulsbury “*switch[es] the memory mode between physical memory and cache memory, depending on the particular application of the processor and memory chip.*” Ex-1007, ¶3, Fig. 1. Here, the memory is used either as a cache or directly-addressable physical memory for

DRAM, but in both cases, the memory is part of a hierarchy including cache within the processor core, DRAM, and non-volatile memory. *Id.*, ¶68, ¶65, Fig. 3. The arrangement is selected using a control bit: in one state, the DRAM operates as a cache memory; in another state, the DRAM operates as a physical memory. *Id.*, ¶68.

260. Ground 3 applies Saulsbury's teachings to Stroud's swappable cards, which have a plurality of types of memory on each card. The combination therefore teaches a selectable arrangement of those memories, where the mode of operation is selectable, using control bits, depending on memory needs at any particular time. Ex-1007, ¶3, ¶¶65-68. Since Stroud aims to provide a flexible platform (*see, e.g.*, Ex-1004, ¶321, ¶¶106-113, ¶343, ¶373), it would have been obvious to apply Saulsbury's teachings to select between common memory arrangements, for example as taught in textbooks such as Cragon, as explained above for Ground 2. Therefore, the combination selects between different arrangements of the plurality of memories in the memory hierarchy of the card. *See id.*

261. Additionally, Saulsbury teaches techniques for adding on-chip DRAM to a CPU and selecting an arrangement where that DRAM operates in multi-level cache or as physical memory. Ex-1007, ¶3, ¶¶67-68, Figs. 1 & 3. Applied to the controller CPU on Stroud's card, the combination teaches a selectable arrangement

of the on-chip DRAM, disk, and other memory on the card in various memory arrangements discussed below.

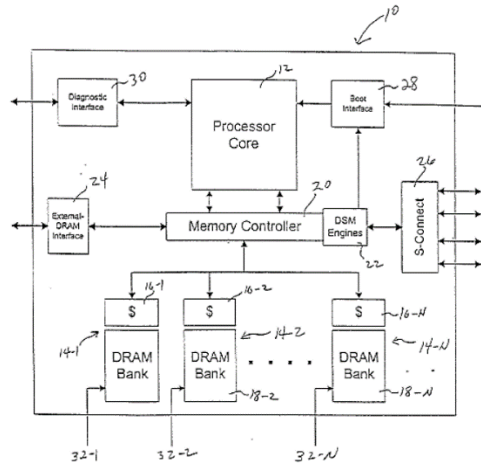


Fig. 1

262. **Multilevel cache.** I explained above how Cragon teaches multilevel cache. The combination with Saulsbury further teaches the plurality of memory types selectively arranged as multilevel cache. For example, Saulsbury teaches using its on-chip DRAM as “cache memory.” Ex-1007, ¶¶2-3, ¶¶5-8, Fig. 1.

Saulsbury also teaches SRAM-based instruction and data caches within the processor core, which was common in processors by the 2000s. Ex-1007, ¶3, ¶26, ¶¶36-37, ¶¶45-47, Fig. 3. Therefore, Saulsbury teaches multi-level cache, e.g., L1 cache in the processor core and on-chip DRAM as a second level of cache (L2).

Id. The cache stores recently-accessed data from other memories, including external DRAM (Ex-1007, Fig. 1 (“External-DRAM”); Ex-1004, Figs. 14A-14B (“DDR3”)) or disk (Ex-1004, ¶127, ¶170, Figs. 5-9A, 14A-B).

263. Moreover, Saulsbury's multilevel cache is consistent with Cragon's teachings. As Cragon explains, L1 cache comprises small and fast memory close to the processor core. Ex-1006, 000219. Saulsbury further teaches this with instruction and data caches in the processor core, which were generally SRAM. Ex-1007, ¶26, ¶¶36-37, ¶¶45-47, Fig. 3. The second level cache is larger and slower than L1; therefore, it would have been obvious to use, e.g., Saulsbury's on-chip DRAM or other larger SRAM or DRAM memories on the card. Ex-1007, ¶3, ¶5, ¶8, Fig. 1; Ex-1004, ¶127. Therefore, a POSITA would have found it obvious and been motivated to use multiple types of memory (on-chip or on-card) in a multilevel cache.

264. **Interleaved memory.** I explained above how Cragon teaches the plurality of memory types on Stroud's card selectively arranged as interleaved memory. Saulsbury's on-chip DRAM (Ex-1007, ¶¶2-3, ¶¶5-8, Fig. 1) provides another memory type for Stroud's cards, to be used as interleaved memory in accordance with Cragon's teachings.

265. **Swapped memory over a specified address range.** I explained above how Cragon teaches the plurality of types of memory types arranged as swapped memory over a specified address range. Saulsbury's on-chip DRAM adds an additional type of memory for the plurality of memory types (on-chip or on-card) that Ground 3 teaches as swapped memory over a specified address range.

For example, Saulsbury teaches on-chip DRAM as direct physical memory; therefore, a POSITA would have found it obvious to use on-chip DRAM as the main physical memory (Ex-1007, ¶¶2-3, ¶¶7-8, ¶¶67-68, Fig. 1) and the disk that is attached to the CPU as non-volatile storage for the swap file (*see* Ex-1004, ¶308, ¶195, ¶196, ¶269, Figs. 5-10, 14A-B, 20, 23, 30, 49). This aligns with Cragon's teachings of arrangements for swapped memory, with DRAM as main memory and a disk for the swap file. Ex-1006, 000256.

266. Memory allocated to specified address ranges according to parameterized task needs. I explained above how, for the memory on Stroud's cards, Cragon teaches the plurality of memory types selectively arranged as memory allocated to specified address ranges according to parameterized task needs. Ex-1004, ¶352; Ex-1006, 000413-414, 000536-548, 000048, 000054. Saulsbury's on-chip DRAM provides another memory type for Stroud's cards, to be allocated to a specified address range, in accordance with Cragon's teachings.

267. For example, Saulsbury teaches adding on-chip DRAM to a CPU and selectively operating memory in different modes. Ex-1007, ¶3, ¶¶65-68, Fig. 1. It would have been obvious to apply that teaching to the controller CPU on Stroud's card. As an example, Stroud Figures 14A-B teach a card configuration with an Intel CPU connected to a solid-state drive (flash memory) and DDR3 DRAM. The

application of Saulsbury's teachings to Stroud adds additional DRAM into the CPU itself:

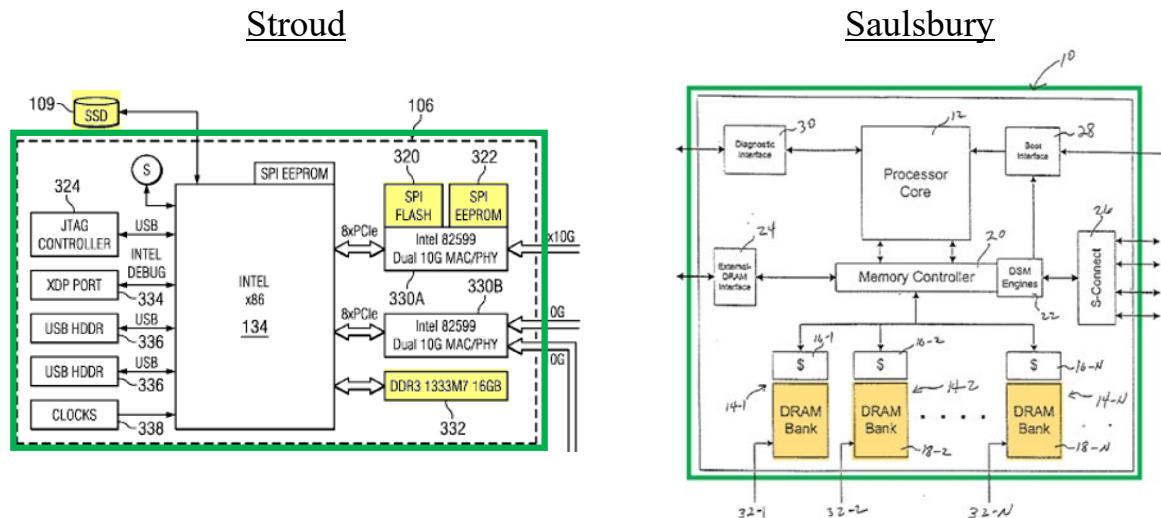


Fig. 1

268. As explained above, Cragon teaches allocating memory to specified address ranges for particular tasks according to parameterized task needs for those tasks (e.g., capacity, bandwidth, service rate, response time, and cost). Ex-1006, 000536-541. For example, on-chip DRAM would be allocated to a range for tasks that require quick response time but small capacity because it has low latency, being on the same chip as the processor. *See* Ex-1006, ¶65. Conversely, external DDR3 DRAM would be allocated to a range for tasks that require larger capacity and can tolerate slower response time. Disks (e.g., hard disks or flash) are allocated to ranges for tasks that require even larger capacity, less frequent access, and non-volatile storage. *See* Ex-1006, 000536-541. Therefore, the combination teaches this limitation.

269. **Motivation to Combine.** Stroud seeks to provide a platform architecture that is scalable and flexible, with software-programmable network processors (for custom software) and user-definable hardware CLDs (for custom hardware) performing a wide variety of applications. Ex-1004, ¶382, ¶321, ¶399, ¶493, ¶128, ¶144. For additional flexibility, each of Stroud’s cards includes a controller CPU. *Id.*, ¶688. Therefore, a POSITA would have been motivated to provide flexible memory arrangements based on the needs of particular applications, including the common memory modes taught by Cragon. I explained the motivation to combine Stroud and Cragon above for Ground 2. *Supra* §V.B.2.[7d] (Motivation to Combine subsection incorporated here).

270. Saulsbury teaches two techniques that further Stroud’s stated goals of scalability and flexibility: (1) the addition of on-chip DRAM to a CPU and (2) techniques for selecting the arrangement of a memory hierarchy. As Saulsbury explains, its teachings are directed to “scalable computer processing architecture[s]” (Ex-1007, ¶8, ¶16) and provide flexible memory arrangements “depending on the particular application.” *Id.*, ¶3. Therefore, a POSITA would have been motivated to apply Saulsbury’s teachings to Stroud’s cards. This would have been a combination of prior art elements according to known methods to yield the predictable result of a CPU with on-chip DRAM and selectable modes, as Saulsbury teaches.

271. The combination would also use Saulsbury's known techniques to improve similar devices in Stroud in the same way. Indeed, Stroud provides motivation to combine with known CPU architectures by explaining that "[c]ontroller 106 may compare any suitable controller" including for example "a general purpose CPU with multiple cores." Ex-1004, ¶154. "For example, controller 106 may be an off-the-shelf processor[.]" *Id.* "[C]ontroller 106 may be a general purpose central processing unit (CPU) such as an Intel x86 compatible part." *Id.* ¶104. Stroud applies its teachings to known a wide swath of general-purpose CISC and RISC CPUs: "[a]s used herein, the terms processor and CPU mean general purpose computing devices with fixed instruction sets or microinstruction sets such as x86 processors (e.g., the INTEL XEON family and the AMD OPTERON family, as examples only), POWERPC processors, and other well-known processor families." *Id.*, ¶110. Saulsbury teaches a general-purpose CPU and originated from Sun Microsystems, a well-known company for RISC CPUs. *See* Ex-1007. Therefore, a POSITA would have been motivated to use Saulsbury's CPU teachings for the controller on Stroud's card or to apply Saulsbury's teachings by adding on-chip DRAM to the CPU.

272. The application of Saulsbury to Stroud offers many benefits. Saulsbury's on-chip DRAM allows "low performance applications" to have "simple, physical memory" that is close to the processor core, while "high

performance applications” gain access to additional “cache memory, and more particularly, second-level cache memory.” Ex-1007, ¶3. Moreover, having a selectable memory arrangement would ensure that memory is maximized for particular applications at any given time, allowing the performance of the memory, and functions of the chip that depend on the memory, to be optimized. *See id.* Saulsbury’s teachings improve on fixed memory arrangements by allowing for flexible deployment. The added flexibility of Saulsbury’s teachings fits with the flexible nature of Stroud’s architecture, which uses a variety of CLDs and network processors that can be programmed for a variety of applications with different memory requirements.

273. The prior art provides express teaching, suggestion, and motivation to combine. Numerous prior art references teach on-chip DRAM and selectable memory arrangements. *See, e.g.,* Ex-1031, Abstract, Figs. 3, 4B, 5; Ex-1026, 1:48-58; Ex-1032, Abstract, Fig. 8; Ex-1027, [54] Title, Abstract, Fig. 4; Ex-1028, (54) Title, ¶5, Fig. 2; Ex-1029, Abstract, Fig. 6; Ex-1030, Abstract, ¶8. The prior art teaches numerous benefits. For example, the flexible arrangement of memory provides increased performance at a lower cost. *See* Ex-1029, 1:23-33; Ex-1032, ¶21-22, ¶49. Selecting to cache between different types of memory, as in the combination of Stroud and Saulsbury, was suggested in the art. Ex-1030, ¶35 (“Caching data...can be carried out effectively...between different storage

systems, thereby making it possible to realize higher performance.”). The prior art teaches the use of on-chip memory and suggests using selectable memory arrangements when “more memory may be required than is available on the chip.” Ex-1031, ¶6. The technique was also used to remedy disadvantages of fixed cache systems that duplicated copies of the same data in different memory, leading to data inconsistencies. Ex-1031, ¶¶7-8. Selectable memory arrangement also contributed to the longevity of non-volatile memory. Ex-1033, Abstract (“accommodate disparate access granularities, reduce the requisite number of memory transactions, and minimize the flow of data external to nonvolatile memory components.”). Selectable arrangements were also used to selectively “combine[] the nonvolatility and reduced per-bit price of flash memory with the speed and durability of DRAM.” Ex-1033, 2:6-16. Memory was placed on-chip to increase access speed in processors and ASICs. Ex-1031, ¶36, ¶11, Fig. 1; *e.g.*, Ex-1034, 1:13-32; Ex-1032, Abstract, Fig. 3. A POSITA would have recognized such benefits and would therefore have been motivated to apply Saulsbury’s techniques to Stroud’s various types of memory.

274. Additionally, a POSITA would have been motivated to improve Stroud’s memory systems as recognized in the art. Ex-1028, ¶2 (“Finding ways to improve system memory is therefore a motivation of computing system engineers”). A POSITA would have appreciated the many benefits from

Saulsbury's teachings that I explained in the preceding paragraph, and would therefore have been motivated to apply such techniques to improve the system's memory.

275. Furthermore, the combination would not have changed the principle of operation for any of these reference because it simply applies Saulsbury's teachings for on-chip DRAM and a selectable arrangement of memory modes to Stroud and Cragon. Stroud's cards and chassis would still have been used for a network testing platform, as Stroud teaches. Ex-1004, ¶¶85-86. Cragon's teachings would have been used to arrange memory in the way that was taught by Cragon. Ex-1006, 000048, 000071-73, 000219-220, 000496-497, 000255-265, 000413-415, 000536-548. And Saulsbury's on-chip DRAM would still have applied to a CPU, while Saulsbury's selection of memory modes would still have been used to control the operation and arrangement of memory in a memory hierarchy. Ex-1007, Abstract, ¶8.

276. **Reasonable Expectation of Success.** I explained above how a POSITA would have had a reasonable expectation of success in combining the teachings of Stroud and Cragon. *Supra* §V.B.2.[7d] (Reasonable Expectation of Success subsection; analysis incorporated by reference). A POSITA would have had a reasonable expectation of success in further combining the teachings of Saulsbury. As explained above, Stroud teaches flexible configurations for its cards

that seek to encompass a wide range of possible CPUs as the card controller, explaining that it can utilize any suitable CPU as a controller, ranging from CISC (x86) to RISC (e.g., PowerPC) architectures. Ex-1004, ¶154, ¶110. Cragon teaches various memory arrangements that were commonly used with CPUs. Ex-1006, 000040-43, 000496-497, 000255-265, 000413-415, 000536-548. Saulsbury teaches a general purpose processor and applies its teachings to CPUs. *See* Ex-1007, ¶¶16-22, Figs. 3 & 5.

277. Additionally, the combination would not have changed the principle of operation of any of the references, as explained above. The combination simply adds Saulsbury's on-chip DRAM and techniques for controlling the memory mode. Ex-1007, ¶68, ¶8. That technique—using control bits—had been the conventional way for controlling settings and modes in a computer for decades. And on-chip DRAM was also known in the art. Ex-1028, ¶22, ¶14, Fig. 7; Ex-1034, 2:3-13. A POSITA would have been confident in the success of applying Saulsbury's technique to Stroud and Cragon because the use of on-chip DRAM and similar selectable memory arrangements were known and taught in the art. Ex-1031, Abstract, Figs. 3, 4B, 5; Ex-1026, 1:48-58; Ex-1032, Abstract, Fig. 8; Ex-1027, [54] Title, Abstract, Fig. 4; Ex-1028, (54) Title, ¶5, Fig. 2; Ex-1029, Abstract, Fig. 6; Ex-1030, Abstract, ¶8.

278. A POSITA would have been more than capable in making the combination given the defined level of education and work experience in processing and streaming architectures including user-defined hardware and software modules. The design and use of CPUs, memory types and hierarchies, and DRAM was part of undergraduate EE/CE education, and work experience in processing architectures would have provided experience to the use of different memory arrangements and configurations, as well as to the settings and controls for computer systems. Education and work experience in user-defined hardware and software modules would have encompassed the types of PLDs, FPGAs, and network processors used in Stroud's cards. *Supra* §III.

3. Dependent Claims 9-10

279. Above, I explained how claims 9-10 were obvious based on Stroud and Cragon. *Supra* §V.B.3-4 (analyses for claims 9-10 incorporated by reference). That same analysis applies here. Saulsbury provides additional teachings with respect to claim 7 but does not otherwise alter the analysis.

D. Ground 4: Claims 4, 8, and 15 Are Rendered Obvious by the Combination of Stroud in View of DaSilva, Cragon, and Saulsbury

280. It is my opinion that a POSITA would have found the Challenged Claims 4, 8, and 15 obvious based on the teachings of Stroud, DaSilva, Cragon and Saulsbury. Dependent claim 4 combines the memory arrangement limitations of

claim 7 into the platform of claim 1, which was directed to details of a packet header used for network communications. In other words, claims 1 and 7 were directed to separate aspects of a platform, and dependent claim 4 merely imports the limitation of claim 7 to claim 1.⁸

281. Accordingly, Ground 4 combines the teachings from Ground 1 (which addressed the packet header of claim 1) with Ground 3 (which addressed the memory limitations of claim 7). Both combinations were based on Stroud. Ground 1 explained the motivation to combine Stroud's packets with DaSilva's chassis ID teachings. Ground 3 explained the motivation to combine Stroud's platform with the selectable memory arrangements taught by Saulsbury and Cragon. Those same motivations apply here. *Supra* §V.A.3.[1e] (Motivation subsection for Ground 1), §V.C.2.[7d] (Motivation subsection for Ground 3). Since the combinations were directed at separate aspects of Stroud's platform—

⁸ Claims 4, 8, and 15 result in similar permutations of claim limitations.

Dependent claim 8 combines the packet header details of claim 1 into the platform used in the method of claim 7. Similar to claim 4, claim 15 adds the memory arrangement limitations of claim 7 to the media instructions of claim 13 that include instructions to carry out steps for the platform of claim 1.

network packet details in Ground 1 versus hardware memory arrangements in Ground 3—the combinations would not have precluded each other.

1. Dependent Claim 4

- | |
|--|
| <p>4. The platform of claim 1, further comprising: at least one of the one or more swappable pods or cards having a plurality of types of memory and memory superposition with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.</p> |
|--|

282. Ground 1 teaches claim 1, including the preamble. *Supra* §§V.A.3.[1pre]-[1e]. The combination with Saulsbury and Cragon further teaches the limitations of claim 4, as explained above for claim 7. *Supra* §V.B.2.[7d] (analysis incorporated by reference).

2. Dependent Claim 8

- | |
|--|
| <p>8. The method of claim 7, wherein the communicating the data flow comprises:
communicating the data flow through the messaging interface network, comprising a network coupled to the plurality of swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.</p> |
|--|

283. Ground 2 teaches claim 7, including the preamble. *Supra* §§V.B.2.[7pre]-[7d]. The combination with DaSilva further teaches the limitations

of claim 8, as explained above for claims 1 and 13. *Supra* §V.A.3.[1e], §V.A.6.[13d] (analyses incorporated by reference).

3. Dependent Claim 15

**15. The computer-readable media of claim 13, wherein the method further comprises:
selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.**

284. Ground 1 teaches claim 13, including the preamble. *Supra* §§V.A.6.[13pre]-[13d]. The combination with Cragon and Saulsbury further teaches the limitations of claim 15, as explained above for claim 7. *Supra* §V.B.2.[7d], §V.C.2.[7d] (analysis incorporated by reference).

VI. BACKGROUND AND QUALIFICATIONS

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

I am an independent consultant with Camelback Computer Architecture, LLC. My residence and place of business is at 7 Farm Road, Los Altos CA, 94024. I am over the age of eighteen, and I am a citizen of the United States.

I have over 45 years of academic and industrial experience in applying, designing, studying, teaching, and writing about microprocessors and computer systems. I received an Electrical Engineering Ph.D. degree in 1984 from Stanford University. I earlier received an Electrical Engineering B.S. degree from MIT in 1973 and an Electrical Engineering M.S. degree from Stanford University in 1978. I have taught classes in computer architecture at Stanford, Tel Aviv, and Arizona State Universities.

From 1976 to 1977, I worked at Burroughs Corporation, where I designed peripheral interface controllers, including those for serial data communications based on Intel 8080 microprocessor components. From 1980 to 1989, I was the lead architect for the design of three high-performance microprocessors at Zilog and National Semiconductor. Later, at Intel, I was the lead architect of the Pentium® Processor from 1989 to 1992 and of the 815 chipset from 1999 to 2000, both of which became the most widely used PC components of their time. The 815 chipset comprised two components: (1) a memory controller hub (MCH) that included a graphics controller and memory controller with interfaces to the CPU, 133 MHz SDRAM system memory modules, an optional, external graphics controller and (2) an I/O controller hub (ICH) that included various I/O controllers (e.g., network, hard drive, USB) for system peripheral devices. Additionally, I served as co-manager for the Itanium processor design from 1993-1997.

I am a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE), and served as the chair of the IEEE Technical Committee on Microprocessors and Microcomputers from 1999 to 2000. I was the keynote speaker at the first Cool Chips conference, dedicated to the study of low-power microprocessors and systems. I have given invited lectures at several universities, and published ten papers in various professional journals and conference proceedings. My paper entitled “Architecture of the Pentium Processor,” was selected as best paper in IEEE Micro for 1993. I am a named inventor on over 30 U.S. patents that pertain to microprocessors, computer systems, and related technology.

286. I have reviewed the '753 Patent, and I am familiar with the patent's subject matter, which is within the scope of my education and professional experience. For example, during development of the Pentium Processor I am familiar with design verification using a large emulation system of FPGAs. Based at least on my background in academia and industry, I am familiar with the issues and technology relating to processors, chipsets, memory, and peripheral devices for computer systems, including those for scalable, multiprocessor systems. I have personally analyzed, developed, and tested such computer components and systems.

A. Compensation

287. For my efforts in connection with the preparation of this declaration I have been compensated at my standard rate of \$695 per hour 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

288. 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

289. 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

290. 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 '753 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 '753 Patent and to take into account new information that becomes available to me.

291. 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 February 11, 2025.

By:

A handwritten signature in black ink, appearing to read "Donald Alpert", written in a cursive style.

Donald Alpert

APPENDIX 1: CURRICULUM VITAE OF DONALD ALPERT



Donald Alpert. Ph.D.
Camelback Computer Architecture, LLC
2310 Homestead Road
Suite C1 #411
Los Altos CA 94024
(650) 386-5301

Donald Alpert, Ph.D., is an independent consultant, specializing in microprocessors and computer systems. For more than a decade in his role as president of Camelback Computer Architecture LLC, Dr. Alpert has provided clients with expert services for patent litigation, technology assessment, and product development. He has over 40 years of wide-ranging academic and industry experience that enable him to provide expert consulting services in the following areas:

- Computers, Computer Systems
- Computer architecture
- Microprocessors
- Semiconductors
- PC Systems
- Embedded Systems
- Design Validation and Debug
- Performance Evaluation
- Cache Memory
- Instruction Sets
- Media Processing
- Power Management

Patent Analysis: Dr. Alpert has served as an expert on dozens of patent litigation cases, handling both infringement and invalidity issues. Dr. Alpert has worked as both a consulting expert and testifying expert assisting attorneys representing major high-tech corporations. He has substantial experience testifying in depositions and at trials before U.S. Federal Courts and the International Trade Commission. Dr. Alpert also consults to assess patents for acquisition and for pre-litigation review. See [here](#) for a summary of testifying experience.

Product Development: Dr. Alpert is a widely recognized innovator with over 30 patents. Dr. Alpert has worked at some of the most ground-breaking technology firms, including Zilog, National Semiconductor, and Intel, where he was the principal architect of the Pentium® Processor. See [here](#) for more information and example projects.

Employment Experience

Camelback Computer Architecture, LLC, Sacramento, CA

Consultant

2/02-Present

Computer architecture consulting services to assess technology, develop products, and protect IP.

Intel Corporation, Chandler, Arizona

Principal Engineer

12/99-1/02

Led architecture team of 10 responsible for definition and performance evaluation of Itanium™ Processor Family microprocessors.

Principal Engineer

5/98-12/99

Led architecture team of 4 responsible for technical strategy and product definition of PC graphics processors and chipsets.

Stanford University, Stanford, California

Consulting Assistant Professor

9/97-4/98

Responsible for teaching undergraduate and graduate classes in computer architecture and participating in processor-design research projects.

Intel Corporation, Santa Clara, California

Engineering Manager

10/93-6/97

Responsible for microarchitecture definition and architecture validation of the Itanium™ microprocessor, which was the first implementation of Intel's 64-bit extension architecture. Led a team of 60 engineers.

Architecture Manager

1/89-10/93

Responsible for the Pentium® Processor technical definition. Worked with customers and Intel marketing to identify product requirements and with the design engineering team to evaluate design options. Led a team of seven engineers responsible for creating technical specifications and for modeling and simulating performance using traces from DOS, Windows, and UNIX execution. After completing the Pentium Processor, led an interdisciplinary team of 30 engineers to define a new 64-bit instruction set architecture.

National Semiconductor Corporation, Herzlia, Israel

Architecture Manager

5/85-1/89

Responsible for defining new microprocessors in the Series 32000 family, including the NS32532 and "Swordfish" high-performance microprocessors. Led a team of five engineers working in the following areas: modeling and simulating high-performance microprocessors, evaluating novel computer architectures, and writing technical specifications for new products.

Zilog, Incorporated, Campbell, California

Component Architect

6/80-4/85

Worked on the Z80,000 CPU, a 32-bit microprocessor that integrated an instruction pipeline, cache, and memory management unit. Responsible for specifying the non-privileged instruction set, evaluating performance, and writing the reference manual for the microprocessor.

Burroughs Corporation, Pasadena, California

Associate Engineer

7/76-7/77

Designed computer peripheral interface controllers using MSI-TTL and 8080 microprocessors.

Alpex Computer Corporation, Danbury, Connecticut

8/75-6/76

Consulting engineer working on design of video games.

Action, Washington, DC

6/73-6/75

Peace Corps volunteer teaching mathematics and physics in a secondary school in Ghana, West Africa.

Donald Alpert Camelback Computer Architecture, LLC (650) 386-5301

Education

Stanford University 9/77-6/84
Graduate study in computer design. National Science Foundation Graduate Fellowship.
Awarded MSEE in 1978 and PhDEE in 1984.

M.I.T. 9/69-6/73
Member of Tau Beta Pi and Eta Kappa Nu. Awarded BSEE in 1973.

Professional Activities

Member of Microprocessor Report editorial board (2002-present)

Anthem College Commencement Address (18-Dec-2009)

Panel session about history of microprocessor development at National Semiconductor Corporation, Computer History Museum (2008).

[\[http://archive.computerhistory.org/resources/access/text/Oral_History/102658246.05.01.acc.pdf\]](http://archive.computerhistory.org/resources/access/text/Oral_History/102658246.05.01.acc.pdf)

IEEE Technical Committee on Microprocessors and Microcomputers, Chair (1999-2000) and Vice-Chair (2001-present).

Lectures about the Itanium™ Processor Family at U.C. Irvine (1999, 2000).

Cool Chips I, Keynote speaker (Tokyo, 1998)

IEEE Hot Chips Symposium Program Co-chair (1994), member of program committee (1993, 1995).

Lectures about the Pentium® Processor at Stanford University, University of Illinois, and Cornell (1993-1994).

Appeared in Stanford UVC Educational Video production about the Pentium Processor (1993).

IEEE International Symposium on Computer Architecture member of program committee (1996) and Session Chair for Cache Design (1989).

Teaching Experience

Faculty Associate at Arizona State University teaching undergraduate class on assembly language programming (Spring/2001) and computer architecture (Spring/2002).

Consulting Associate Professor at Stanford University teaching undergraduate and graduate classes in computer architecture (Fall/1997, Winter/1998).

Consulting Professor at Tel Aviv University teaching graduate class in computer architecture (Spring/1987)

Peace Corps volunteer in a secondary school in Ghana, West Africa teaching mathematics and physics (1973-1975).

Don Alpert, "How Microprocessors Upset the Computer Industry," Microprocessor Report, December 11, 2006.

Don Alpert, "Evolving PC Chip-Set Topologies," Microprocessor Report, April 17, 2006.

Don Alpert, "Will Microprocessors Become Simpler?" Microprocessor Report, November 17, 2003.

Don Alpert, "Itanium Processor Status Report," Microprocessor Report, July 28, 2003.

Don Alpert, "Scalable MicroSupercomputers," Microprocessor Report, March 17, 2003.

Don Alpert, "ColdFire Goes Fully Superscalar," Microprocessor Report, October 28, 2002.

Donald Alpert and Alan Jay Smith, "Guest Editor Introduction," IEEE Micro special issue on Hot Chips VI, April 1995.

Donald Alpert and Dror Avnon, "Architecture of the Pentium Processor," IEEE Micro, June 1993, pp. 11-21. Selected best paper in IEEE Micro for 1993.

D. Alpert, A. Averbuch, and O. Danielli, "Performance Comparison of Load/Store and Symmetric Instruction Set Architectures," Proceedings, 18th International Symposium on Computer Architecture, June 1990, pp. 172-181.

Benjamin Maytal, Sorin Iacobovici, Donald Alpert, et al, "Design Considerations for a General-Purpose Microprocessor," Computer, January 1989, pp. 66-76.

Donald Alpert and Michael Flynn, "Performance Trade-offs for Microprocessor Cache Memories," IEEE Micro, August 1988, pp. 44-54.

D. Alpert, J. Levy, and B. Maytal, "Architecture of the NS32532 Microprocessor," Proceedings, 1987 IEEE International Conference on Computer Design, October 1987, pp. 168-172. Awarded best presenter at Design and Test sessions.

D. Alpert, D. Biran, L. Epstein, et al, "Trends in VLSI Microprocessor Design," Proceedings, First Annual Conference on Computer Technology, Systems and Applications (CompEuro '87), May 1987, pp. 564-567.

Y Sidi, D. Alpert, D. Biran, et al, "Design Considerations of an Advanced 32-Bit Microprocessor," Proceedings, 15th Conference of Electrical and Electronic Engineers in Israel, April 1987.

D. Alpert, "Trends in VLSI Microprocessor Design," Proceedings, 15th Conference of Electrical and Electronic Engineers in Israel, April 1987.

Donald Alpert, Michael J. Flynn, and Scott Wakefield, "Directly Executed Languages for VLSI Processor Design," Proceedings, 1983 IEEE International Conference on Computer Design, October 1983, pp. 609-612.

Donald Alpert, "Powerful 32-Bit Micro Includes Memory Management," Computer Design, October 1983, pp. 213-220.

Don Alpert, Dean Carberry, Mike Yamamura, et al, "32-Bit Processor Chip Integrates Major System Functions," Electronics, July 14, 1983, pp. 113-119.

Patents

4,802,085, Apparatus and method for detecting and handling memory-mapped I/O by a pipelined microprocessor

5,249,286, Selectively locking memory locations within a microprocessor's on-chip cache

5,263,153, Monitoring control flow in a microprocessor

5,416,913, Method and apparatus for dependency checking in a multi-pipelined microprocessor

5,442,756, Branch prediction and resolution apparatus for a superscalar computer processor

5,475,824, Microprocessor with apparatus for parallel execution of instructions

5,479,652, Microprocessor with an external command mode for diagnosis and debugging

5,481,751, Apparatus and method for storing partially-decoded instructions in the instruction cache of a CPU having multiple execution units

5,559,986, Interleaved cache for multiple accesses per clock cycle in a microprocessor

5,606,676, Branch prediction and resolution apparatus for a superscalar computer processor

5,617,554, Physical address size selection and page size selection in an address translator

5,621,886, Method and apparatus for providing efficient software debugging

5,638,525, Processor capable of executing programs that contain RISC and CISC instructions

5,657,253, Apparatus for monitoring the performance of a microprocessor

5,659,679, Method and apparatus for providing breakpoints on taken jumps and for providing software profiling in a computer system

5,669,011, Partially decoded instruction cache

5,671,435, Technique for software to identify features implemented in a processor

5,675,825, Apparatus and method for identifying a computer microprocessor

5,692,167, Method for verifying the correct processing of pipelined instructions including branch instructions and self-modifying code in a microprocessor

5,729,724, Adaptive 128-bit floating point load and store operations for quadruple precision compatibility

5,740,413, Method and apparatus for providing address breakpoints, branch breakpoints, and single stepping

5,764,959, Adaptive 128-bit floating point load and store instructions for quad-precision compatibility

5,774,686, Method and apparatus for providing two system architectures in a processor

5,790,834, Apparatus and method using an ID instruction to identify a computer microprocessor

5,802,605, Physical address selection and page size selection in an address translator

5,958,037, Apparatus and method for identifying features and the origin of a computer microprocessor

5,991,874, Conditional move using a compare instruction generating a condition field

6,052,801, Method and apparatus for providing breakpoints on a selectable address range

6,219,774, Address translation with/bypassing intermediate segmentation translation to accommodate two different instruction set architecture

6,385,718, Computer system and method for executing interrupt instructions in operating modes

6,408,386, Method and apparatus for providing event handling functionality in a computer system

6,584,558, Article for providing event handling functionality in a processor supporting different instruction sets

7,010,671, Computer system and method for executing interrupt instructions in two operating modes

7,389,403, Adaptive computing ensemble microprocessor architecture

7,802,073, Virtual core management

APPENDIX 2: MATERIALS CONSIDERED IN THE PREPARATION OF THIS DECLARATION

Exhibit No.	Description
1001	U.S. Patent No. 10,873,753 (the '753 patent)
1002	Prosecution History of the '753 patent
1004	U.S. Patent Application Publication No. 2013/0346667A1 ("Stroud")
1005	U.S. Patent No. 9,148,389 ("DaSilva")
1006	Harvey G. Cragon, <i>Memory Systems and Pipelined Processors</i> ("Cragon")
1007	U.S. Patent Application Publication No. 2002/0087821 ("Saulsbury")
1008	U.S. Patent No. 8,015,386 ("Kulkarni")
1009	Reserved
1010	Kuon et al., "FPGA Architecture: Survey and Challenges" ("Kuon")
1011	Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology" ("Trimberger")
1012	Reserved
1013	Reserved
1014	Frey, "The IBM Data Center Family" ("Frey")
1015	Desai et al., "BladeCenter system overview" ("Desai")
1016	Hughes et al., "BladeCenter processor blades, I/O expansion adapters, and units" ("Hughes")
1017	Caldeira et al., "IBM Power Systems S814 and S824 Technical Overview and Introduction" ("Caldeira")
1018	Steffen, "Nallatech Software Tools on the Maxwell FPGA Parallel Computer" ("Steffen")
1019	Nallatech, "H100 Series FPGA Application Accelerators" ("Nallatech")
1020	Reserved
1021	Exhibit C to Concurrent Ventures' and XstreamEdge's Oct. 11, 2024 Preliminary Infringement Contentions
1022	Declaration of J. Munford re Cragon
1023	IETF, RFC 6146 "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers"
1024	U.S. Patent No. RE45708 ("Frantz")
1025	U.S. Patent No. 9,270,592 ("Sites")

Exhibit No.	Description
1026	U.S. Patent No. 5,155,833 (“Cullison”)
1027	U.S. Patent No. 5,586,293 (“Baron”)
1028	U.S. Patent Application Publication No. 2018/0088853A1 (“Kotra”)
1029	U.S. Patent No. 9,959,212 (“Takeda”)
1030	U.S. Patent Application Publication No. 2013/0318196A1 (“Yamamoto”)
1031	U.S. Patent Application Publication No. 2006/0277365A1 (“Pong”)
1032	U.S. Patent Application Publication No. 2006/ 0294308A1 (“Bailey”)
1033	U.S. Patent No. 10,847,196 (“Ware”)
1034	U.S. Patent No. 6,029,224 (“Asthana”)
1035	Concurrent Ventures’ and XtreamEdge’s March 29, 2024 Complaint in WDTX 1:24-cv-335
1036	Concurrent Ventures’ and XtreamEdge’s Jan. 23, 2025 Preliminary Claim Construction Disclosures
1037	Declaration of R. Jones re Cragon
1038	Microsoft, Microsoft Computer Dictionary (5th Ed., 2002) (excerpts)
1039	Chambers Dictionary (13th Ed., 2014) (excerpts)
1040	U.S. Patent Application Publication No. 2014/0115137A1 (“Keisam”)
1041	Exhibit C to Concurrent Ventures’ and XtreamEdge’s Dec. 20, 2024 Supplemented Infringement Contentions
1042	Concurrent Ventures’ and XtreamEdge’s July 22, 2024 Response to Defendant’s Motion to Dismiss
1043	U.S. Patent No. 7,356,047 (“Mahalawat”)
1044	Chu, ENG-46158 rev 1.8 “Serial-GMII Specification”

APPENDIX 3: CHALLENGED CLAIMS

[1pre]. A platform for data flow processing, comprising:

[1a] one or more swappable pods or cards in one or more chassis, coupled through a messaging interface network;

[1b] each of the one or more swappable pods or cards having one or more hardware modules or one or more software modules;

[1c] one or more of the plurality of swappable pods or cards having a portion for user-definable hardware modules or user-definable software modules;

[1d] the plurality of swappable pods or cards being user-configurable to implement data flow processing architectures; and

[1e] a network coupled to the one or more swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.

2. The platform of claim 1, further comprising:

at least a subset of the one or more swappable pods or cards having an option for direct connection to one or more pods, cards or components, bypassing the module messaging interface network.

3. The platform of claim 1, wherein the one or more swappable pods or cards comprises at least one processor card having a at least one processor allocatable for offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, and executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.

4. The platform of claim 1, further comprising:

at least one of the one or more swappable pods or cards having a plurality of types of memory and memory superposition with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.

[7pre]. A method for data flow processing in a platform, comprising:

[7a] configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network, to implement a data flow processing architecture;

[7b] configuring user-definable hardware modules or user-definable software modules in each of one or more of the plurality of swappable pods or cards;

[7c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user-definable software modules; and

[7d] selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.

8. The method of claim 7, wherein the communicating the data flow comprises:

communicating the data flow through the messaging interface network, comprising a network coupled to the plurality of swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.

9. The method of claim 7, wherein the communicating the data flow comprises:

bypassing the messaging interface network and communicating through a direct connection among two or more pods, cards or components.

10. The method of claim 7, further comprising:

allocating at least one processor of a processor card, from among the plurality of swappable pods or cards, to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, or executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.

[13pre]. A tangible, non-transitory, computer-readable media having instructions thereupon which, when executed by a processor, cause the processor to perform a method comprising:

[13a] configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network, to implement a data flow processing architecture;

[13b] configuring user-definable hardware modules or user-definable software modules in each of one or more of the plurality of swappable pods or cards;

[13c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more

software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user-definable software modules; and

[13d] communicating the data flow through the messaging interface network, comprising a network coupled to the plurality of swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.

14. The computer-readable media of claim 13, wherein the method further comprises:

allocating at least one processor of a processor card having a plurality of processors, from among the plurality of swappable pods or cards, to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, or executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.

15. The computer-readable media of claim 13, wherein the method further comprises:

selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of

the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.

Side-by-Side List of Challenged Claims (substantively similar language bolded in red, green, and blue):

[1pre]. A platform for data flow processing , comprising:	[7pre]. A method for data flow processing in a platform , comprising:
[1a] one or more swappable pods or cards in one or more chassis, coupled through a messaging interface network;	[7a] configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network , to implement a data flow processing architecture
[1b] each of the one or more swappable pods or cards having one or more hardware modules or one or more software modules;	[7b] configuring user-definable hardware modules or user-definable software modules in each of one or more of the plurality of swappable pods or cards
[1c] one or more of the plurality of swappable pods or cards having a portion for user-definable hardware	
[1d] the plurality of swappable pods or cards being user-	[7a] configuring a plurality of swappable pods or cards in one or

configurable to implement data flow processing architectures; and	more chassis, coupled through a messaging interface network, to implement a data flow processing architecture
	[7c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user-definable software modules; and
[1e] a network coupled to the one or more swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.	

	[7d] selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.
--	--

2. The platform of claim 1, further comprising: at least a subset of the one or more swappable pods or cards having an option for direct connection to one or more pods, cards or components, bypassing the module messaging interface network.	9. The method of claim 7, wherein the communicating the data flow comprises: bypassing the messaging interface network and communicating through a direct connection among two or more pods, cards or components.
3. The platform of claim 1, wherein the one or more swappable pods or cards comprises at least one processor card having a at least one processor allocatable for offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, and executing	10. The method of claim 7, further comprising: allocating at least one processor of a processor card, from among the plurality of swappable pods or cards, to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of

in cooperation with a sandbox of one of the plurality of swappable pods or cards.	hardware modules, or executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.
---	--

4. The platform of claim 1, further comprising: at least one of the one or more swappable pods or cards having a plurality of types of memory and memory superposition with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.	[7d] selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.
---	---

[7pre]. A method for data flow processing in a platform , comprising:	[1pre]. A platform for data flow processing , comprising:
[7a] configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network, to implement a data flow processing architecture	[1a] one or more swappable pods or cards in one or more chassis, coupled through a messaging interface network; [1d] the plurality of swappable pods or cards being user- configurable to

	implement data flow processing architectures; and
[7b] configuring user-definable hardware modules or user-definable software modules in each of one or more of the plurality of swappable pods or cards	[1b] each of the one or more swappable pods or cards having one or more hardware modules or one or more software modules; [1c] one or more of the plurality of swappable pods or cards having a portion for user-definable hardware modules or user-definable software modules; [1d] the plurality of swappable pods or cards being user-configurable to implement data flow processing architectures; and
	[1e] a network coupled to the one or more swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each

	board in each chassis can be addressed through the header.
[7c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user-definable software modules; and	[13c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user definable software modules; and
[7d] selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.	15. The computer-readable media of claim 13, wherein the method further comprises: selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges

	according to parameterized task needs.
8. The method of claim 7, wherein the communicating the data flow comprises: communicating the data flow through the messaging interface network, comprising a network coupled to the plurality of swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.	[1e] a network coupled to the one or more swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.
9. The method of claim 7, wherein the communicating the data flow comprises: bypassing the messaging interface network and communicating through a	2. The platform of claim 1, further comprising: at least a subset of the one or more swappable pods or cards having an option for direct connection to one or more pods, cards or

direct connection among two or more pods, cards or components.	components, bypassing the module messaging interface network.
10. The method of claim 7, further comprising: allocating at least one processor of a processor card , from among the plurality of swappable pods or cards , to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, or executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.	3. The platform of claim 1, wherein the one or more swappable pods or cards comprises at least one processor card having a at least one processor allocatable for offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, and executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.

13 <pre>[pre]</pre> A tangible, non-transitory, computer-readable media having instructions thereupon which, when executed by a processor, cause the processor to perform a method comprising:	<pre>[1pre].</pre> A platform for data flow processing, comprising:
<pre>[13a]</pre> configuring a plurality of swappable pods or cards in one or more chassis, coupled through a messaging interface network, to implement a data flow processing architecture	<pre>[1a]</pre> one or more swappable pods or cards in one or more chassis, coupled through a messaging interface network; <pre>[1d]</pre> the plurality of swappable pods or cards being user- configurable to

	implement data flow processing architectures; and
[13b] configuring user-definable hardware modules or user-definable software modules in each of one or more of the plurality of swappable pods or cards	[1b] each of the one or more swappable pods or cards having one or more hardware modules or one or more software modules; [1c] one or more of the plurality of swappable pods or cards having a portion for user-definable hardware modules or user-definable software modules; [1d] the plurality of swappable pods or cards being user-configurable to implement data flow processing architectures; and
[13c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user definable software modules; and	[17c] communicating a data flow in accordance with the implemented data flow processing architecture among one or more hardware modules or one or more software modules in each of the plurality of swappable pods or cards, including the user-definable hardware modules or user-definable software modules; and

[13d] communicating the data flow through the messaging interface network, comprising a network coupled to the plurality of swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.	[1e] a network coupled to the one or more swappable pods or cards and supporting messaging-based communication using packets each having a header with a chassis identifier, a board identifier, a module identifier, an instance identifier, and a type identifier, so that each type of module, each instance of a type of module, and each module on each board in each chassis can be addressed through the header.
---	---

14. The computer-readable media of claim 13, wherein the method further comprises: allocating at least one processor of a processor card, from among the plurality of swappable pods or cards, to offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, or executing in cooperation with a sandbox of one of	3. The platform of claim 1, wherein the one or more swappable pods or cards comprises at least one processor card having a at least one processor allocatable for offloading hardware tasks to software, orchestrating data flows, orchestrating configurations of hardware modules, and executing in cooperation with a sandbox of one of the plurality of swappable pods or cards.
---	---

the plurality of swappable pods or cards.	
15. The computer-readable media of claim 13, wherein the method further comprises: selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.	[7d] selecting a memory superposition in at least one of the plurality of swappable pods or cards having a plurality of types of memory with selectable arrangement of the plurality of types of memory as multilevel cache, interleaved memory, swapped memory over a specified address range, and memory allocated to specified address ranges according to parameterized task needs.

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.