

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

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

v.

POLARIS POWERLED TECHNOLOGIES, LLC,
Patent Owner.

IPR2025-00517
U.S. Patent No. 8,554,968

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

TABLE OF CONTENTS

PETITIONER’S EXHIBIT LIST	5
I. INTRODUCTION	7
II. GROUNDS FOR STANDING.....	8
III. NOTE.....	8
IV. BACKGROUND	8
V. SUMMARY OF THE ’968 PATENT	8
VI. PROSECUTION HISTORY OF THE ’968 PATENT.....	16
VII. LEVEL OF ORDINARY SKILL IN THE ART.....	17
VIII. CLAIM CONSTRUCTION	17
IX. RELIEF REQUESTED AND THE REASONS FOR THE REQUESTED RELIEF	18
X. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE....	18
A. Challenged Claims	18
B. Statutory Grounds for Challenges	18
C. Ground 1: Claims 1-18 are obvious in view of Borchers.	20
1. Summary of Borchers.	20
2. The PCI Standards Represent Background Knowledge of a POSITA.....	25
3. Claim 1	25
4. Claim 2.....	47
5. Claim 3.....	48

6.	Claim 4.....	49
7.	Claim 5.....	50
8.	Claim 6.....	52
9.	Claim 7.....	54
10.	Claim 8.....	56
11.	Claim 9.....	67
12.	Claim 10.....	70
13.	Claim 11.....	71
14.	Claim 12.....	71
15.	Claim 13.....	72
16.	Claim 14.....	72
17.	Claim 15.....	73
18.	Claim 16.....	74
19.	Claim 17.....	75
20.	Claim 18.....	75
D.	Ground 2: Claims 1-18 are obvious in view of Borchers and the PCI standards.....	76
1.	Summary of the PCI Standards.....	77
2.	Reasons to combine the PCI standard with Borchers.....	77
3.	Claim Analysis for claims 1-18.....	80
E.	Ground 3: Claims 1-18 are obvious in view of Borchers, the PCIe/PCI standards, and Serebrin.....	80

1.	Summary of Serebrin	81
2.	Serebrin is Entitled to its Provisional Filing Date.	81
3.	Reasons to combine Serebrin with Borchers and the PCIe/PCI standards	83
4.	Claim Analysis for claims 1-18	87
XI.	DISCRETIONARY DENIAL IS NOT WARRANTED.....	87
A.	The Fintiv factors favor institution.	87
1.	No evidence regarding a stay	88
2.	Parallel proceeding trial date	88
3.	Investment in the parallel proceeding	88
4.	Overlapping issues with the parallel proceeding	88
5.	Petitioner is a defendant	89
6.	Other circumstances	89
B.	Advanced Bionics Test Favors Institution.	89
XII.	CONCLUSION.....	91
XIII.	MANDATORY NOTICES	92
A.	Real Party-in-Interest	92
B.	Related Matters.....	92
C.	Lead and Back-up Counsel and Service Information	92
	CERTIFICATE OF WORD COUNT	94
	CERTIFICATE OF SERVICE	95

PETITIONER’S EXHIBIT LIST

Ex.1001	U.S. Patent No. 8,554,968 to Onufryk et al.
Ex.1002	Prosecution History of U.S. 8,554,968
Ex.1003	Declaration of Paul Min under 37 C.F.R. § 1.68
Ex.1004	Curriculum Vitae of Paul Min
Ex.1005	U.S. Patent Publication No. 2010/0262979 to Borchers et al. (“Borchers”)
Ex.1006	PCI Local Bus Specification 3.0 (February 3, 2004) (“PCI”)
Ex.1007	PCI Express 2.0 Base Specification Revision 0.9 (September 11, 2006), (“PCIe”)
Ex.1008	U.S. Patent Publication No. 2011/0197003 to Serebrin et al. (“Serebrin”)
Ex.1009	U.S. Provisional Patent No. 61/301,937, (“Serebrin Provisional”), Ex.1009
Ex.1010	Excerpts from the IEEE Dictionary
Ex.1011	Complaint, <i>Polaris PowerLED Technologies, LLC v. Western Digital Corporation et al.</i> 2:24-cv-02864 (C.D. CA)
Ex.1012	District Court Trial Statistics
Ex.1013	Director Vidal Memorandum re: Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation, June 22, 2022
Ex.1014	Internet Archive – PCI Express
Ex.1015	Internet Archive – PCI
Ex.1016	U.S. Patent Publication No. 2007/0245061
Ex.1017	U.S. Patent No. 7,924,914

Ex.1018	U.S. Patent Publication No. 2010/0185719
---------	--

I. INTRODUCTION

The prior art presented in this Petition describes the same interrupt coalescing technique that is described and claimed in U.S. Patent 8,554,968 (the “’968 patent,” Ex.1001). The ’968 patent relates to communication between a memory controller and a host system. In conventional systems, the host sends read/write commands to the memory controller. *See* Ex.1001, 1:26-35. Upon completing those commands, the memory controller reports a response back to the host. *See id.* The reported responses are then placed into a response queue managed by the host. *See e.g.*, Ex.1005, [0017]. Upon placing an entry into the host’s response queue, the memory controller also sends an interrupt signal to let the host know there is a new item in that response queue for processing. Ex.1005, [0035].

The purported novelty of the ’968 patent is to hold off on sending the interrupt signal until certain conditions (e.g., by a timer or response counter) are met. *See* Ex.1001, 1:38-2:41. But this technique—referred to as interrupt coalescing—was already known as of the filing date of the ’968 patent. *See generally*, Ex.1005. The Borchers reference, describes using a “ResponseNew Counter 280” and a “LastResponse Timer 282” to determine when to send an interrupt signal to the host. Ex.1005, Fig. 2. As shown in this Petition, and supported by the declaration of Dr. Paul Min (Ex.1003), Borchers renders obvious claims 1-18 (hereinafter, the “Challenged Claims”) of the ’968 patent.

Accordingly, pursuant to 35 U.S.C. §§ 311, 314(a), and 37 C.F.R. § 42.100, Petitioner respectfully requests that the Board review and cancel as unpatentable under (pre-AIA) 35 U.S.C. § 103(a) the Challenged Claims of the '968 patent.

II. GROUNDS FOR STANDING

Petitioner certifies that the '968 patent is eligible for IPR and that Petitioner is not barred or estopped from requesting IPR challenging the patent claims. 37 C.F.R. § 42.104(a).

III. NOTE

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

IV. BACKGROUND

Dr. Min's declaration provides a background section on memory systems, command and response queues, circular queues, doorbell registers, and interrupt signals that are relevant to the '968 patent. Ex.1003, ¶¶28-41.

V. SUMMARY OF THE '968 PATENT

The '968 patent describes and claims known concepts related to a memory controller that triggers interrupt signals in a host system. The '968 patent relates to communication between a non-volatile memory controller 105 and a host processing unit 110, as shown in Fig. 1 below.

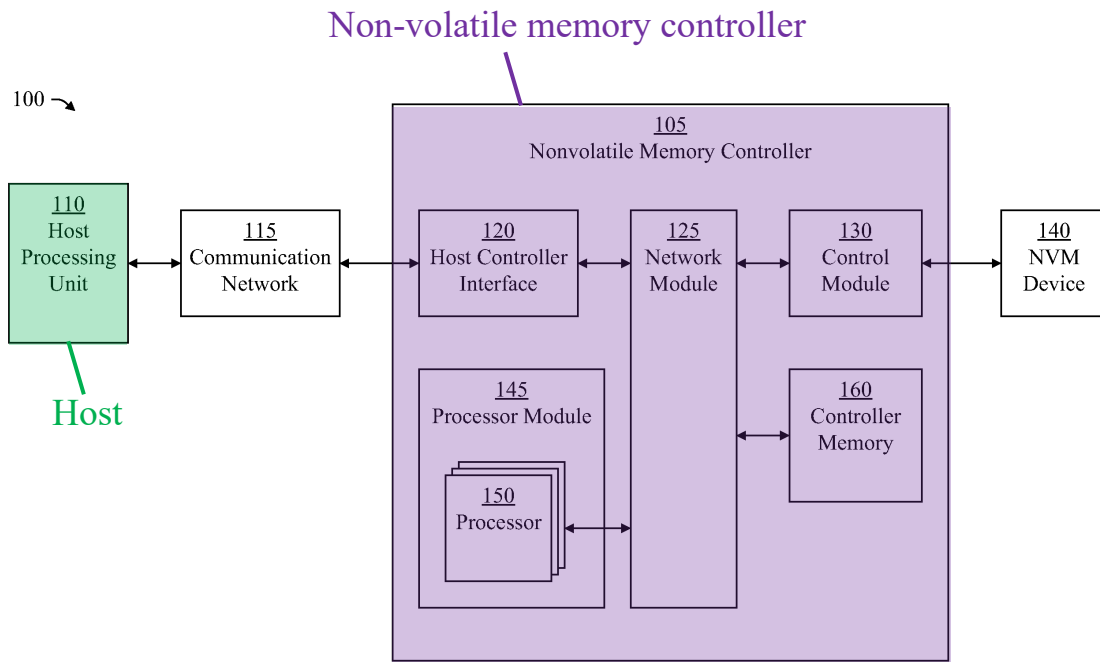


FIG. 1

Ex.1001, Fig. 1 (annotated); Ex.1003, ¶43.

To process read or write commands from the host 110 to the controller 105, the host processing unit 110 “includes submission queues 520 and completion queues 525” shown in Fig. 2. Ex.1001, 9:38-39. “The submission queues 520 store nonvolatile memory commands for controlling operation of the nonvolatile memory controller 105.” Ex.1001, 9:39-41. “The completion queues 525 store completion statuses corresponding to nonvolatile memory commands stored in the submission queues 520.” Ex.1001, 9:41-44. The completion and submission queues are shown in Fig. 5 below.

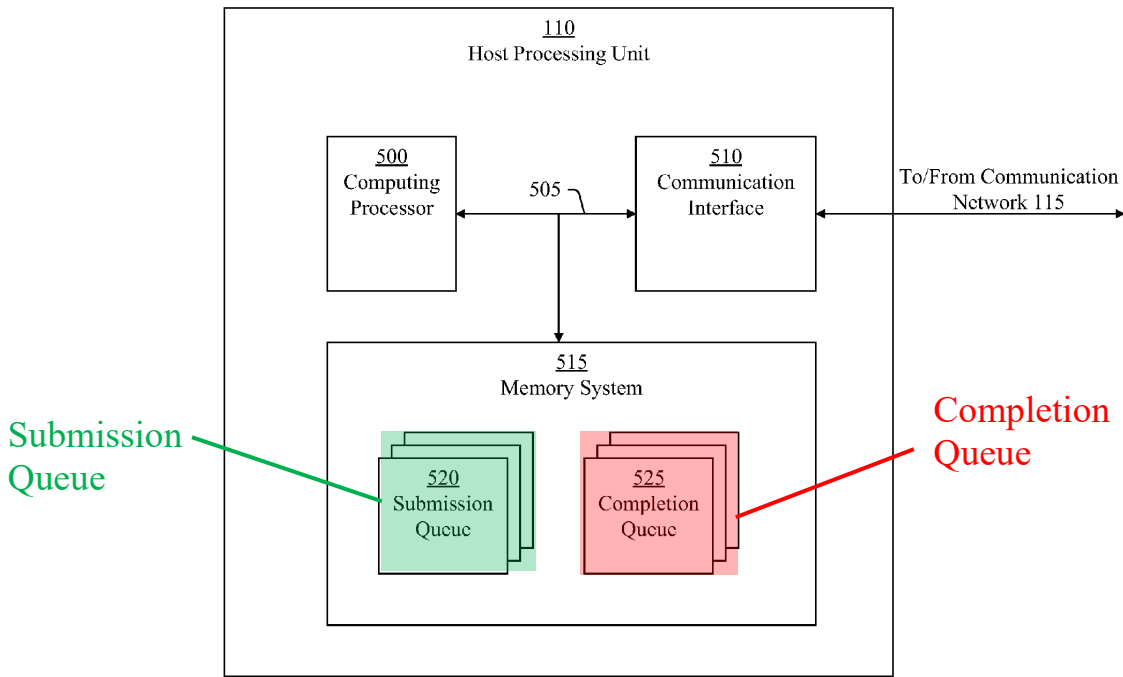
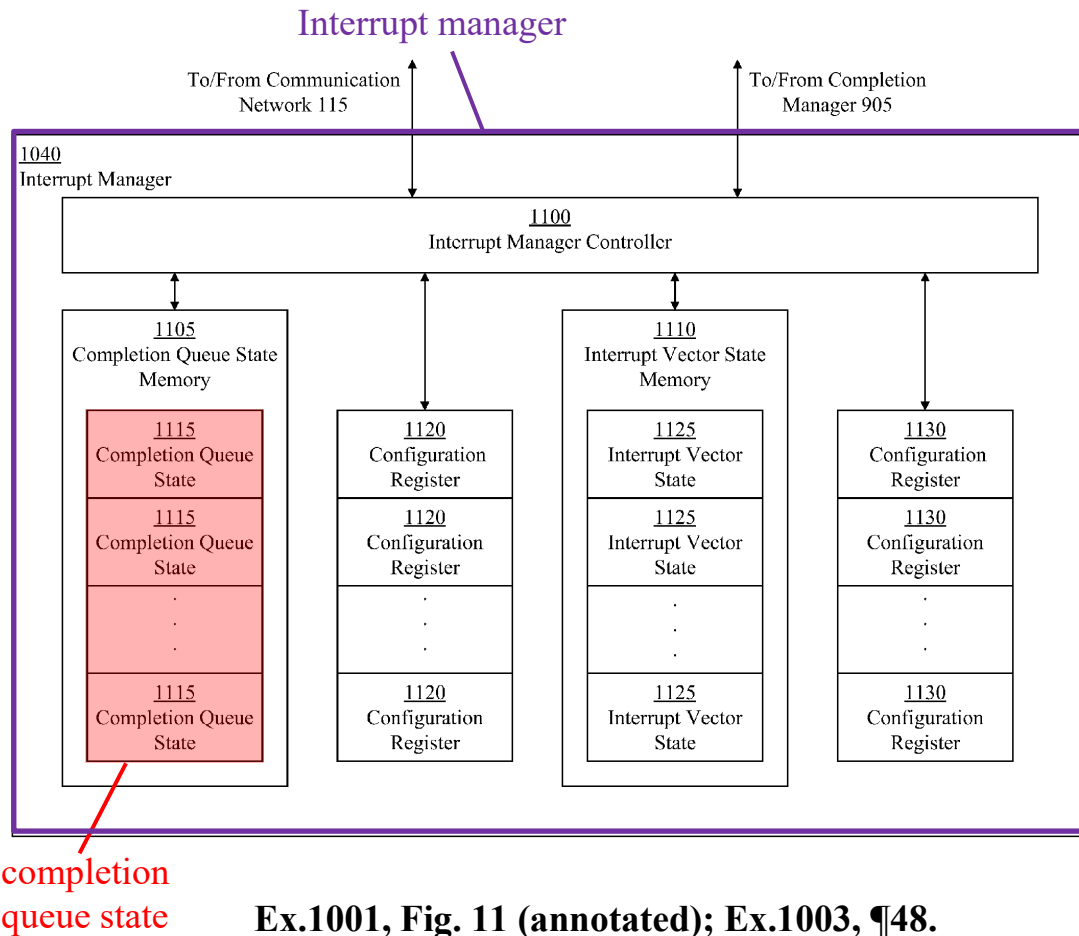


FIG. 5
Ex.1001, Fig. 5 (annotated); Ex.1003, ¶44.

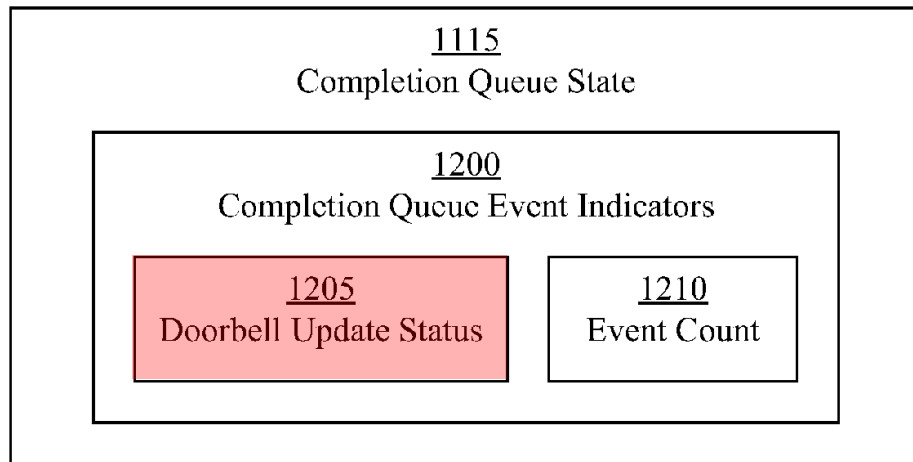
The completion queue is used to manage completion notifications from the memory controller. After the memory controller completes a command, “the host processing unit 110 receives a request packet including a completion status from the host controller ... [and] writes the completion status into the storage location 700 at the tail 710 of the completion queue 525.” Ex.1001, 10:43-48. Then, after being notified through an interrupt signal, “the host processing unit 110 reads completion statuses from storage locations 700 starting at the head 705 of the completion queue 525 to determine the status of the nonvolatile commands associated with the completion statuses that have been processed by the nonvolatile memory controller 105.” Ex.1001, 10:48-53.

To notify the host that there are items within the completion queue to be processed, the memory controller sends an interrupt signal to the host. The claims of the '968 patent are directed to this interrupt process. For example, claim 1 recites “[a] nonvolatile memory controller for alerting a host processing unit to an unprocessed completion status contained in a completion queue of the host processing unit.” The memory controller refrains from sending an interrupt until certain conditions are met.

To determine whether certain conditions are met, the memory controller receives information as to the status of the completion queue. The '968 patent calls this information a “*completion queue state*.” Claim 1 recites, for example, “*an interrupt manager configured to generate a completion queue state for indicating the occurrence of a completion queue event associated with the completion queue.*” An interrupt manager 1040 that receives completion queue states 1115 is shown in Fig. 11 below.



The completion queue state indicates the status of the completion queue. The “interrupt manager 1040 (e.g., the interrupt manager controller 1100) monitors completion queue events and generates completion queue states 1115.” ’968 patent, 15:13-15. Some examples of completion queue states are “completion queue event indicators 1200 [which] include a doorbell update status 1205 and an event count 1210.” Ex.1001, 18:1-2. The doorbell update status 1205 is shown in Fig. 12 below.

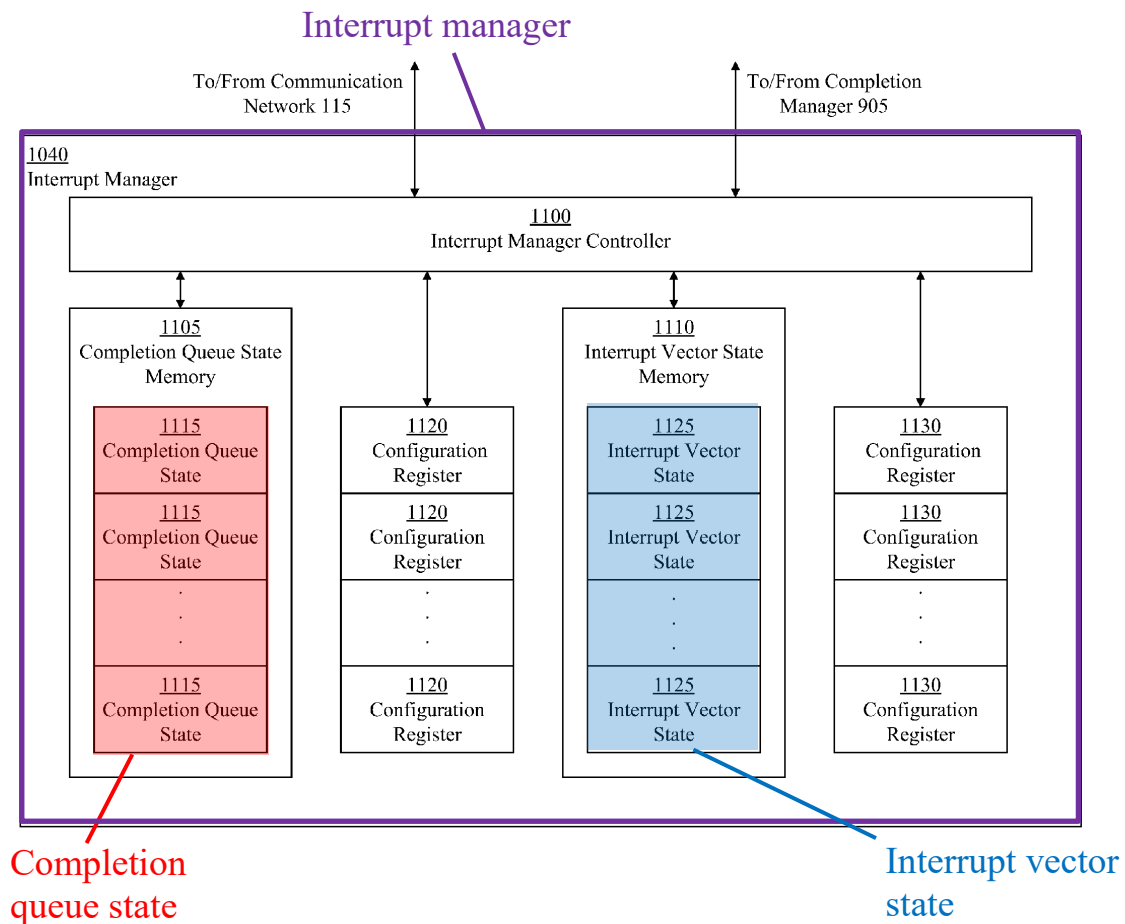


Ex.1001, Fig. 12 (annotated); Ex.1003, ¶49.

The “doorbell update status 1205 indicates whether the host processing unit 110 has performed a doorbell update event since the last time the interrupt manager 1040 cleared the completion queue state 1115.” Ex.1001, 18:3-6. A doorbell update event is when the host updates the head pointer of the completion queue: “In the doorbell update event, the host processing unit 110 updates the completion queue head pointer 1025 of the completion queue 525 associated with the completion queue state 1115.” Ex.1001, 18:6-10. The doorbell update event concept is claimed as follows: “*and wherein the completion queue state includes a doorbell update status indicating whether the host processing unit has performed a doorbell update event in which the host processing unit updates a head pointer stored in the nonvolatile memory controller for the completion queue.*”

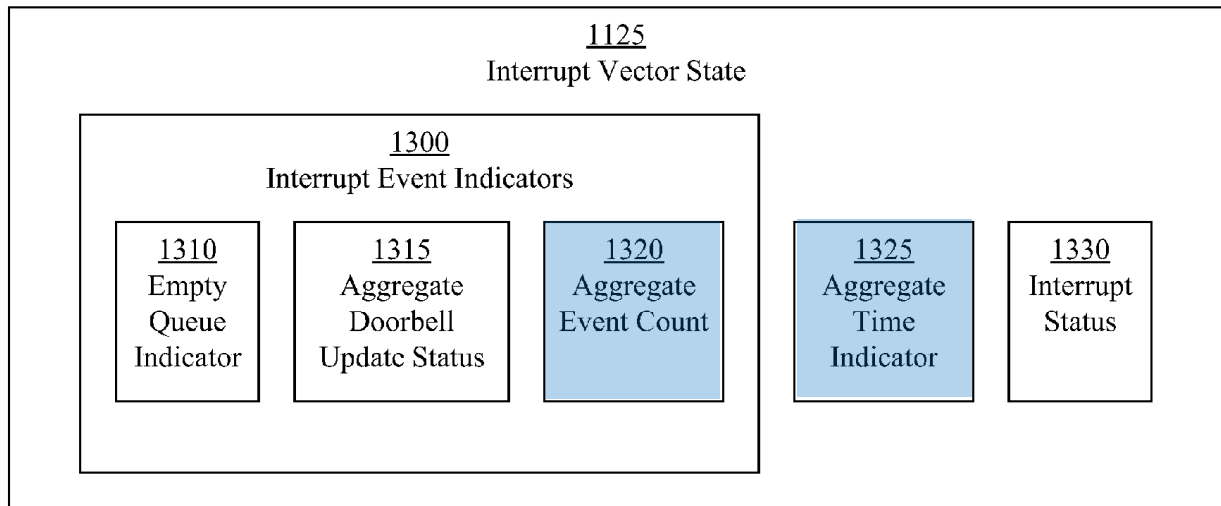
The completion queue state is used by an “*interrupt vector state*” to maintain

the set of conditions for determining whether to generate an interrupt. Ex.1001, 17:21-23. The claims of the '968 patent recite this concept as “*generate an interrupt vector state based on the completion queue state.*” The interrupt vector state is also shown in Fig. 11.



Ex.1001, Fig. 11 (annotated); Ex.1003, ¶51.

The interrupt vector state includes timers and counters, as illustrated in Fig. 13, shown below.



Ex.1001, Fig. 13 (annotated); Ex.1003, ¶52.

The aggregate event count 1320 may be used to trigger an interrupt when the number of unprocessed completion packets sent reaches a threshold. *See* Ex.1001, 19:15-33. “In this way, the interrupt manager 1040 controls a transmission rate of interrupt message packets to the host processing unit 110 based on the aggregate event count threshold.” Ex.1001, 19:28-32. As another example, “[w]hen the aggregate time indicator 1325 is enabled, the interrupt manager 1040 controls a transmission rate of interrupt message packets to the host processing unit 110 based on the aggregate time threshold.” Ex.1001, 19:49-52.

Accordingly, the interrupt vector state is used to determine that there are unprocessed completion events in the completion queue and that the conditions have been met for triggering an interrupt. Claim 1 recites this concept as “*determine the completion queue of the host processing unit contains an*

unprocessed completion status based on the interrupt vector state.” The ’968 patent explains that “[t]he interrupt manager 1040 determines based on the interrupt vector state 1125 whether any completion queue 525 associated with the interrupt vector state 1125 contains an unprocessed completion status.” Ex.1001, 16:3-6. “[T]he interrupt manager 1040 refrains from generating further interrupt message packets unless one or more conditions in the interrupt vector state 1125 are satisfied.” Ex.1001, 16:11-14.

When the conditions are satisfied, “[t]he interrupt manager 1040 transmits the interrupt message packet to the host processing unit 110 through the communication network 115.” Ex.1001, 16:18-20. This concept is recited in claim 1 as “*generate an interrupt message packet for triggering an interrupt in the host processing unit to alert the host processing unit of the unprocessed completion status in the completion queue.*” Thus, by waiting until certain conditions have been met before sending more interrupt signals, the interrupt manager 1040 performs a technique known as interrupt coalescing.

However, the concept of interrupt coalescing, as well as the other concepts recited in the claims of the ’968 patent, were already known as of the time of filing. Ex.1003, ¶¶42-56.

VI. PROSECUTION HISTORY OF THE ’968 PATENT

The ’968 patent was filed on March 21, 2011 and claims priority to a

provisional application filed August 16, 2010. In a first Office Action, the USPTO rejected several claims based on 25 U.S.C. 112 as being indefinite. Ex.1002, 115-118. There were no prior art rejections. In response, the Applicant amended the claims to address the Office’s 112 rejection. Ex.1002, 136-145.

However, for the reasons explained below, the prior art teaches the interrupt coalescing techniques recited in the ’968 patent. Ex.1003, ¶¶57-63.

VII. LEVEL OF ORDINARY SKILL IN THE ART

A person of ordinary skill in the art (“POSITA”) in the field of the ’968 patent, as of its earliest possible filing date of August 16, 2010, would have been someone knowledgeable and familiar with memory systems. A POSITA would have had a bachelor’s degree in electrical engineering, or an equivalent, and approximately two years of professional working in the field of memory systems. Lack of professional experience can be remedied by additional education, and vice versa. Ex.1003, ¶¶21-23.

VIII. CLAIM CONSTRUCTION

Claim terms in IPR are construed according to their “ordinary and customary meaning” to those of skill in the art. 37 C.F.R. § 42.100(b). Petitioner submits that, for the purposes of this proceeding and the Grounds presented herein, no claim term requires express construction. *Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co.*, 868 F.3d 1013, 1017 (Fed. Cir. 2017). Ex.1003, ¶64.

IX. RELIEF REQUESTED AND THE REASONS FOR THE REQUESTED RELIEF

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

X. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE

A. Challenged Claims

Petitioner challenges claims 1-18.

B. Statutory Grounds for Challenges

Grounds	Claims	Basis
#1	1-18	Obvious under 35 U.S.C. § 103 (Pre-AIA) over Borchers
#2	1-18	Obvious under 35 U.S.C. § 103 (Pre-AIA) over Borchers and the PCI Standards
#3	1-18	Obvious under 35 U.S.C. § 103 (Pre-AIA) over Borchers, the PCI Standards, and Serebrin

U.S. Patent Publication No. 2010/0262979 to Borchers et al. (“Borchers”) was filed on April 8, 2010 and published on October 14, 2010. Borchers is thus prior art to the ’968 patent under at least (pre-AIA) 35 U.S.C. 102(e).

The PCI standards include the PCI Local Bus Specification Revision 3.0 (Ex.1006) dated February 3, 2004. The PCI standards also include the PCI Express 2.0 Based Specification Revision 0.9 (Ex.1007) dated September 11, 2006. Indeed, the background section of the ’968 patent acknowledges that PCI and PCIe

standards were known. *See* Ex.1001, 1:15-25. Moreover, the '968 patent explains that interrupt signals may comport with the "PCI Specification 3.0" and the "PCIe specification." Ex.1001, 16:61-67. Ex.1003, ¶204.

Dr. Min explains that POSITAs were aware of and relied on the PCI and PCIe standards documents (Ex.1006 and 1007). Ex.1003, ¶205. POSITAs knew that the standards documents could be obtained from the PCI website (www.pcisig.com). *See* Ex.1014; Ex.1015; Ex.1003, ¶206. Additionally, various patents that predate the '968 patent refer to the PCI local bus specification 3.0 and the PCI express specification 2.0. *See* Ex.1016; Ex.1017; Ex.1018; Ex.1003, ¶204.

The PCI standards (Ex.1006 and Ex.1007) are thus printed publications that were publicly available and known to POSITAs before the filing date of the '968 patent. Ex.1003, ¶¶201-06. Accordingly, the PCI standards are prior art to the '968 patent under (pre-AIA) 35 U.S.C. 102(b).

U.S. Patent Publication No. 2011/0197003 to Serebrin et al. ("Serebrin," Ex.1008) was filed on December 6, 2010. Serebrin claims priority to provisional application 61/301,937 (Ex.1009) filed February 5, 2010. The Serebrin provisional application is nearly identical to the specification of the Serebrin publication and provides support for at least one claim. *See Dynamic Drinkware, LLC v. Nat'l Graphics, Inc.*, 800 F.3d 1375, 1380 (Fed. Cir. 2015). As such, Serebrin is entitled to its provisional filing date and is prior art to the '968 patent under pre-AIA 35

U.S.C. 102(e).

C. Ground 1: Claims 1-18 are obvious in view of Borchers.

1. Summary of Borchers.

Borchers, like the '968 patent, describes techniques for interrupt coalescing. Borchers describes communication between a host 106 and a controller board 102 of a data storage device 100, as shown in Fig. 1 below.

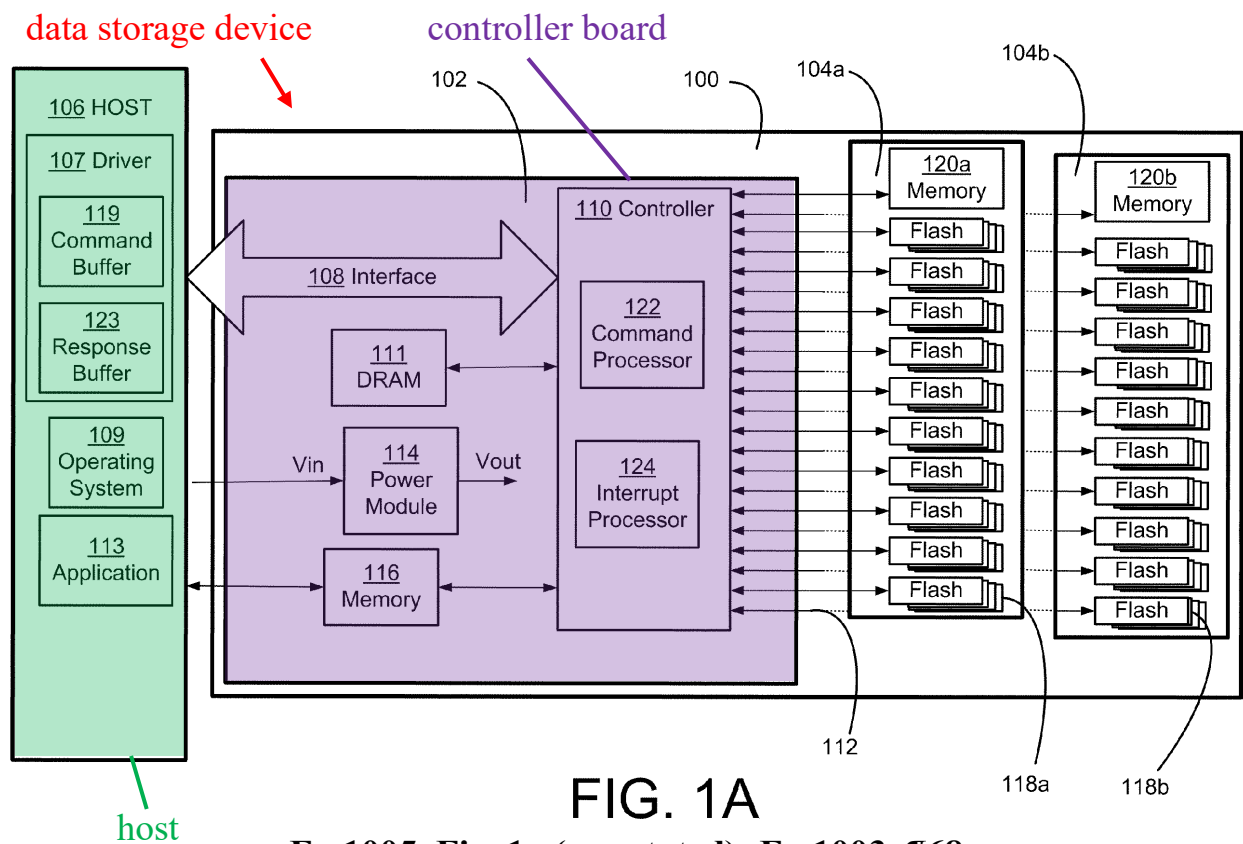
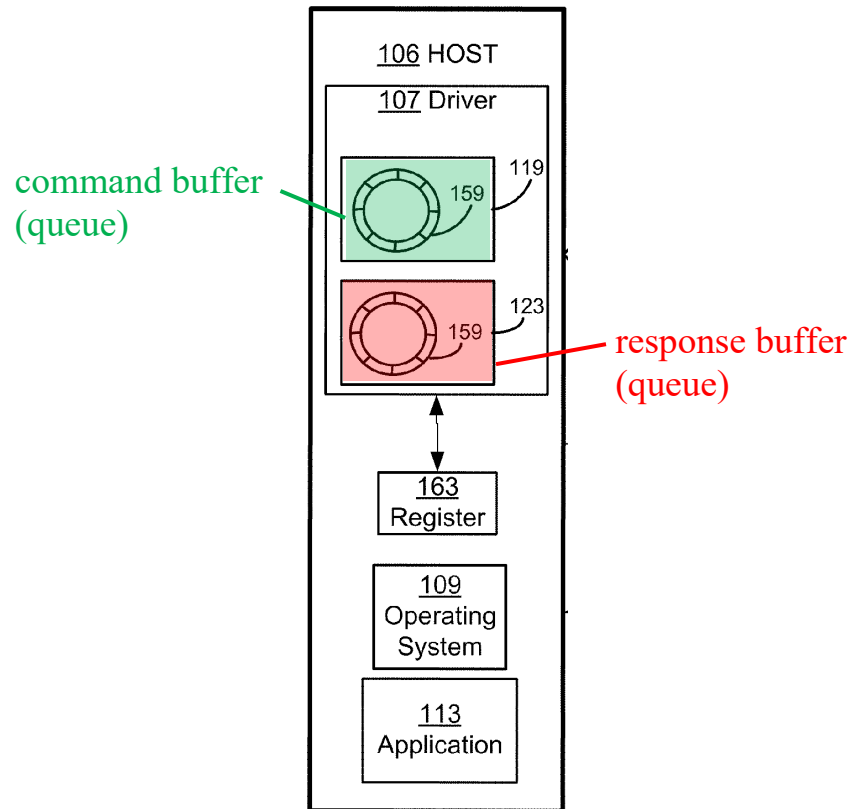


FIG. 1A
Ex.1005, Fig. 1a (annotated); Ex.1003, ¶68.

Like the '968 patent, the host includes two queues: a command buffer (queue) 119 and a response buffer (queue) 123. The response buffer 123 “is populated with responses by the data storage device for retrieval by the host

device.” Ex.1005, [0004]. “The response buffer 123 also can be configured as a circular queue.” Ex.1005, [0030].



Ex.1005, Fig. 1c (partial, annotated); Ex.1003, ¶69.

Borchers explains that “[e]ach of the circular queues 159 of the command buffer 119 and the response buffer 123 include a head pointer and a tail pointer.” Ex.1005, [0030]. “Commands and responses may be inserted into the circular queue 159 at the tail pointer and removed at the head pointer.” Ex.1005, [0030]. For example, when the host seeks to read or write to memory, “[t]he host 106 may write a command to the command buffer 119 and update the command tail

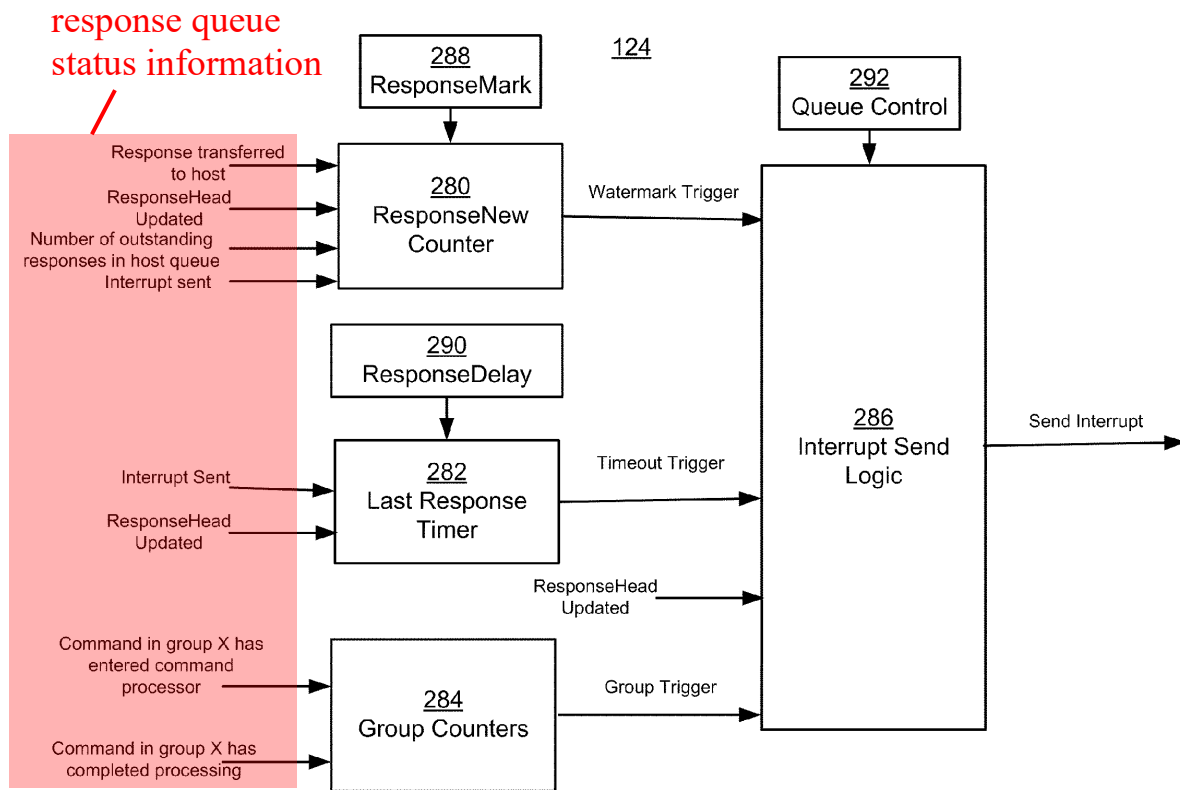
pointer.” Ex.1005, [0034]. Then, when the memory controller is ready to process those commands, it will “read the command out of the command buffer 119 [and] update the head pointer ... to indicate to the host 106 that the command processor 122 has received the command.” Ex.1005, [0034].

After the memory controller has processed a command, it sends a response command back to the host. It does so by writing “a response to the host” at the tail end of the response queue. Ex.1005, [0035]. The memory controller then “updates the response tail pointer ... to indicate that the command is finished.” Ex.1005, [0035].

In some examples, “[t]he interrupt processor 124 is configured to signal the host 106 with an interrupt when new responses are available in the response buffer 123.” Ex.1005, [0035]. However, in some cases, the controller may refrain from sending an interrupt until certain conditions have been met (i.e., interrupt coalescing). In particular, “the host 106, through its driver 107, may control when the interrupt processor 124 should generate interrupts.” Ex.1005, [0036].

To determine whether to send interrupts, the memory controller receives information that indicates the status of the response queue. This response queue status information may include “when a packet is transferred to the host 106, when the ResponseHead is updated, the number of outstanding responses in the host response buffer 123 and when an interrupt has been sent.” Ex.1005, [0060].

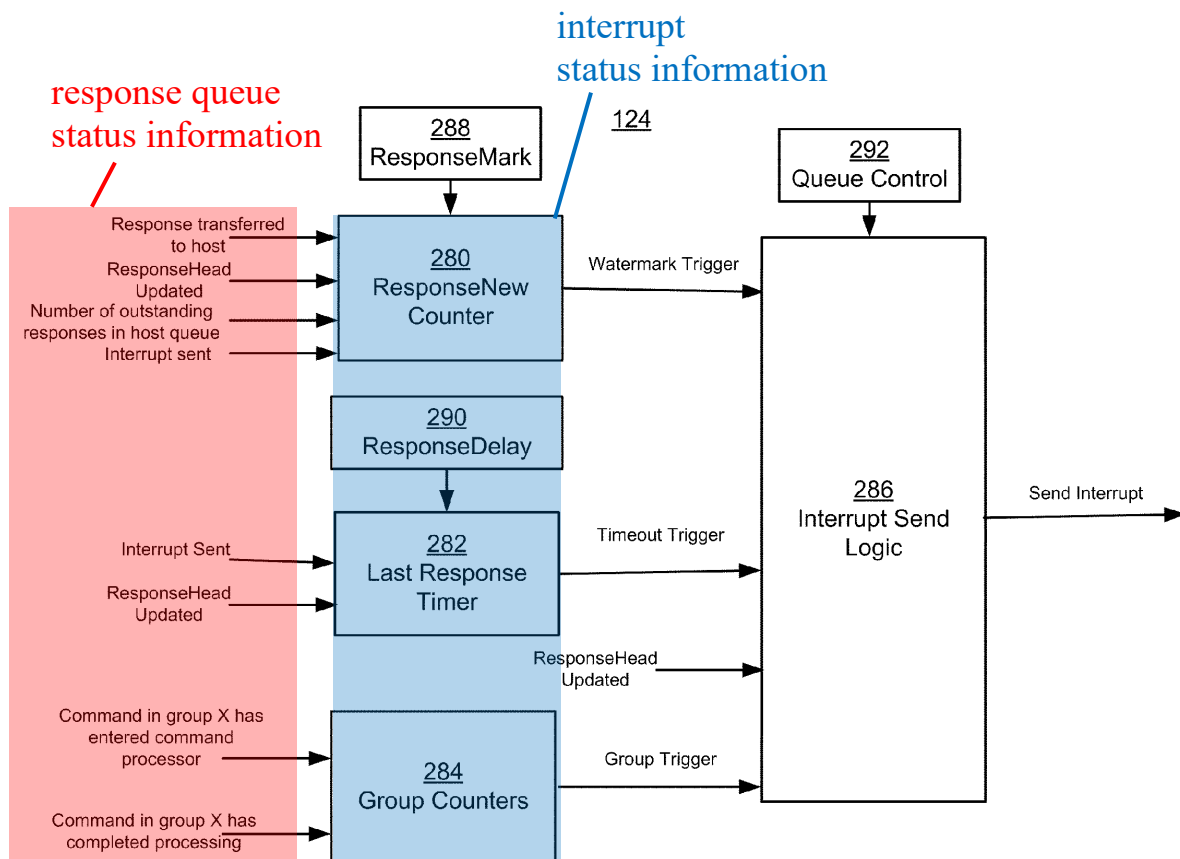
Determining whether the ResponseHead (head pointer of the response queue) has been updated is referred to as a doorbell status update. *See e.g.*, Ex.1006, 308.



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶73.

The response queue status information can then be used to determine the status of interrupt mechanisms, which keep track of the conditions used to trigger an interrupt. For example, Borchers explains that the response queue status information can be used to update a ResponseNew Counter 280. “The ResponseNew Counter 280 is configured to track the number of responses transferred to the host 106 that the host has yet to process.” Ex.1005, [0060].

“When the counter 280 reaches or exceed the watermark level set by the host 106, i.e., the ResponseMark 288, then a watermark trigger is generated and sent to the interrupt send logic 286.” Ex.1005, [0060]. Borchers also describes other interrupt mechanisms such as a timer 282 or a group counter 284. As illustrated in Fig. 2 below, these interrupt mechanisms maintain status information (i.e., count of responses or time since last response).



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶74.

Accordingly, as will be explained in detail in the analysis below, Borchers describes the same interrupt coalescing technique described in the '968 patent.

Borchers uses the state of the response queue (“*completion queue state*”) to update interrupt status information (“*interrupt vector state*”). That interrupt status information keeps track of the conditions for triggering an interrupt signal. When those conditions are met (i.e., a watermark or expired timer), then the interrupt signal is sent. Ex.1003, ¶¶68-76.

2. The PCI Standards Represent Background Knowledge of a POSITA.

The PCI Standards represent the background knowledge of a POSITA because their subject matter is critical for designing memory systems that are compliant with industry standards. Given that the PCIe standard is “an enhancement to the PCI architecture” which was “the most widely accepted and implemented expansion standard in the world” as of 2004 (Ex.1007, 21; Ex.1006, 15), POSITAs in the field of memory systems would have been familiar with their teachings and would have sought to design their memory systems in accordance with such standards. Ex.1003, ¶77.

3. Claim 1

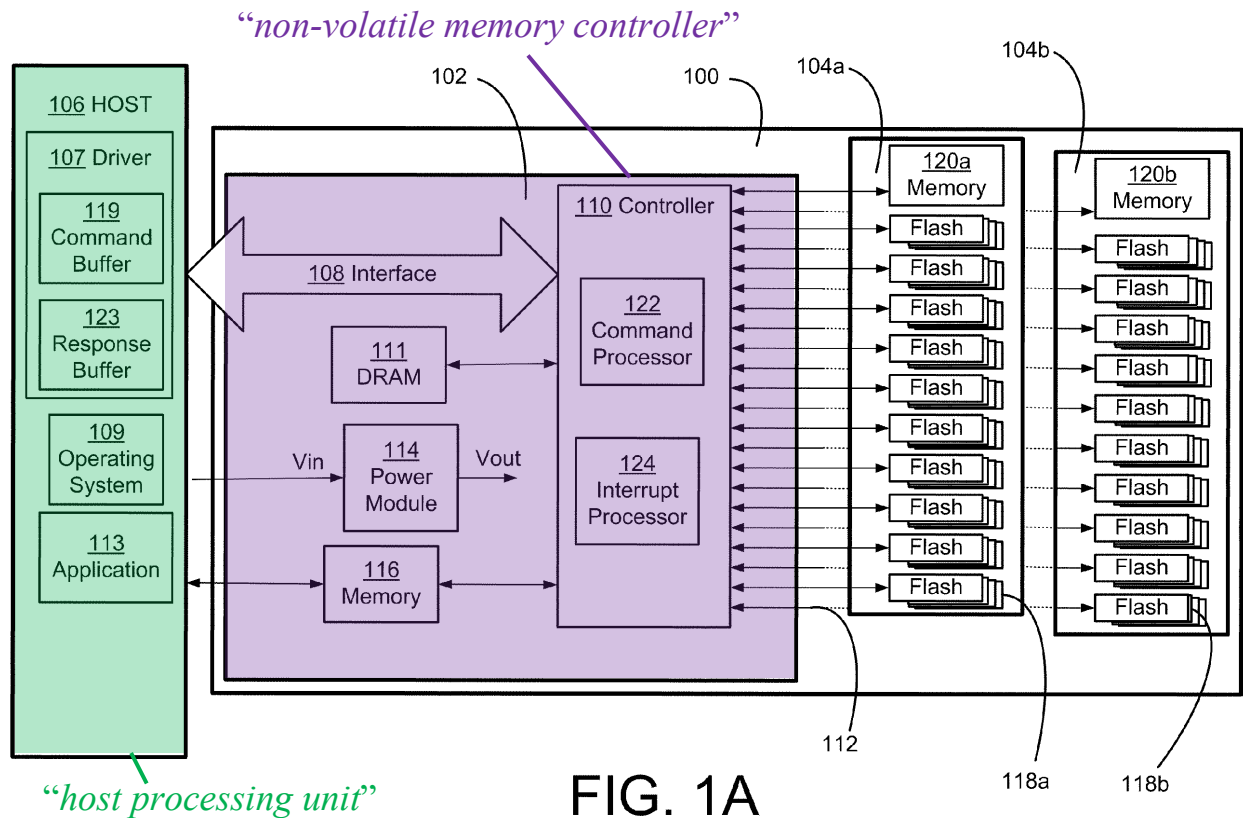
The present Ground describes how the challenged claims are obvious over a single reference, Borchers, with additional reference to the PCIe Standard, which is explicitly called out in Borchers. Grounds 2 and 3 present alternative challenges for the claims based on Borchers in view of the PCIe Standard (Ground 2), and

further in view of Serebrin (Ground 3).

[1.0] *A nonvolatile memory controller for alerting a host processing unit to an unprocessed completion status contained in a completion queue of the host processing unit, the nonvolatile memory controller comprising:*

The language that comes after the phrase “*A nonvolatile memory controller for ...*” of the preamble [1.0] describes a purpose, or intended use, of the controller. To the extent that the preamble is deemed limiting, Borchers teaches the entire preamble.

First, Borchers describes a controller board 102 (“*non-volatile memory controller*”) for a data storage device 100 that stores data for a host 106 (“*host processing unit*”). “FIG. 1A illustrates a block diagram of a data storage device 100 and a host 106. The data storage device 100 may include a controller board 102 and one or more memory boards 104a and 104b.” Ex.1005, [0018]. “The controller board 102 may include a controller 110, a DRAM 111, multiple channels 112, a power module 114, and a memory module 116.” Ex.1005, [0019]. Fig. 1A is shown below.



Borchers, Fig. 1a (annotated); Ex.1003, ¶79.

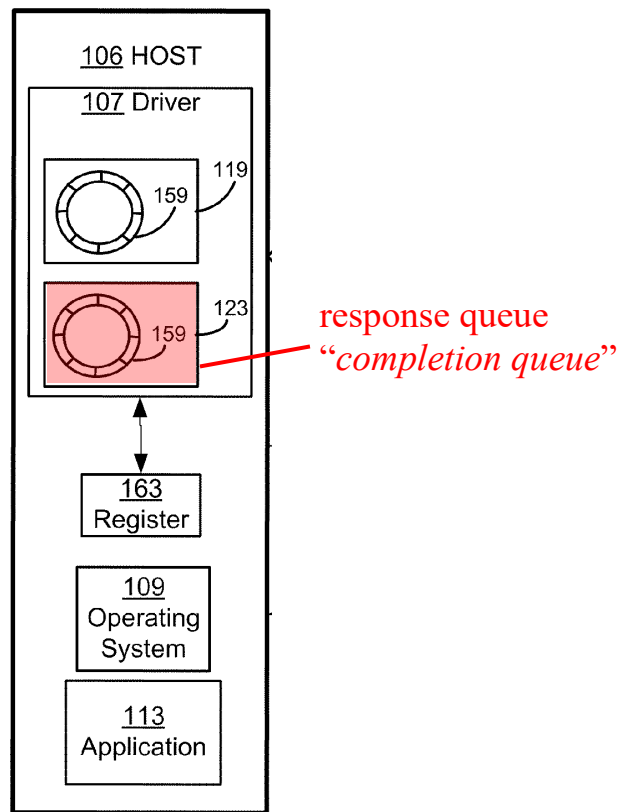
Borchers’ controller board 102 is a non-volatile memory controller because it is responsible for controlling flash memory. Ex.1005, [0019]. Fig. 1A illustrates the memory devices 104a as flash memory and explains that “the memory boards 104a and 104b may include multiple flash memory chips 118a and 118b on each of the memory boards.” Ex.1005, [0019]; *see also* Ex.1005, [0125]. Flash memory is described as a non-volatile type of memory. Ex.1005, [0128] (“**forms of non-volatile memory**, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and **flash memory devices**.”). Accordingly, because the controller board 102 serves as a controller for non-volatile flash memory, it is a

“*nonvolatile memory controller*” as claimed.

Furthermore, Borchers’ host 106 is a “*host processing unit*” because it processes instructions from an application. “The application 113 may be operably coupled to the operating system 109 and/or to the driver 107. The application 113 may generate the commands and communicate the commands to the operating system 109.” Ex.1005, [0020].

Second, the host 106 (“*host processing unit*”) includes a response buffer 123¹ (“*completion queue*”). “For example, a command buffer 119 may be used for commands and a response buffer 123 may be used for responses or results to the commands.” Ex.1005, [0027]. The response queue 123 may be “configured as a circular queue.” Ex.1005, [0030]. Fig. 1C (partial) is shown below.

¹ In, Borchers, “the terms **buffer** and **queue** are used interchangeably.” Borchers, [0027].



Ex.1005, Fig. 1c (partial, annotated); Ex.1003, ¶83.

The response queue 123, like the claimed completion queue, receives completion responses from the memory controller board 102. “For example, a command buffer 119 may be used for commands and **a response buffer 123 may be used for responses or results to the commands.**” Ex.1005, [0027]. “When the command processor 122 **completes** the command, **the command processor 122 writes a response to the host** using a DMA operation **and updates the response tail pointer** with another DMA operation to indicate that the command is finished.” Ex.1005, [0035].

Borchers explains that the response queue uses the producer-consumer

model. *See* Ex.1005, [0030]. “The host 106 may be ... the consumer of the response buffer 123. The data storage device 100 may be the producer of the response buffer 123.” Ex.1005, [0030]. In other words, the “data storage device 100 may write the command head pointer and the response tail pointer and may read the command tail pointer and the response head pointer.” Ex.1005, [0030].

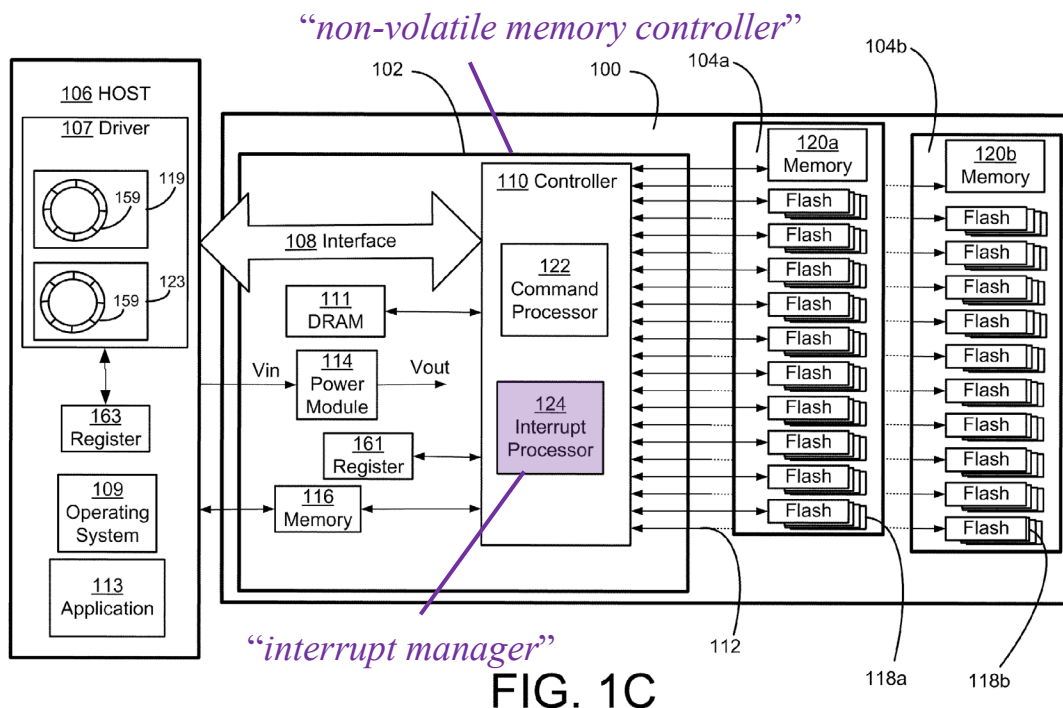
Accordingly, because the response queue stores the notifications of completed commands, it is a “*completion queue*” as claimed. Borchers’ response queue performs a similar function as does the completion queue described in the ’968 patent. *Compare* Ex.1005, [0030]-[0035] *with* Ex.1001, 10:43-53.

Third, the non-volatile memory controller is configured to alert the host processing unit when there are new (i.e., unprocessed) events in the response queue (“*alerting a host processing unit to an unprocessed completion status contained in a completion queue*”). Borchers explains that the “interrupt processor 124 is configured to signal the host 106 with an interrupt when new responses are available in the response buffer 123.” Ex.1005, [0035].

Thus, because Borchers describes a non-volatile memory controller that alerts a host when there are new items in a response queue, Borchers renders this claim element obvious, to the extent the preamble is limiting. Ex.1003, ¶¶78-87.

[1.1] *an interrupt manager configured to generate a completion queue state for indicating the occurrence of a completion queue event associated with the completion queue,*

First, Borchers’ memory controller board 102 includes an interrupt processor 124 (“*interrupt manager*”). “The controller board 102 may include a controller 110.” Ex.1005, [0019]. “The controller 110 may include a command processor 122 and **an interrupt processor 124.**” Ex.1005, [0019].

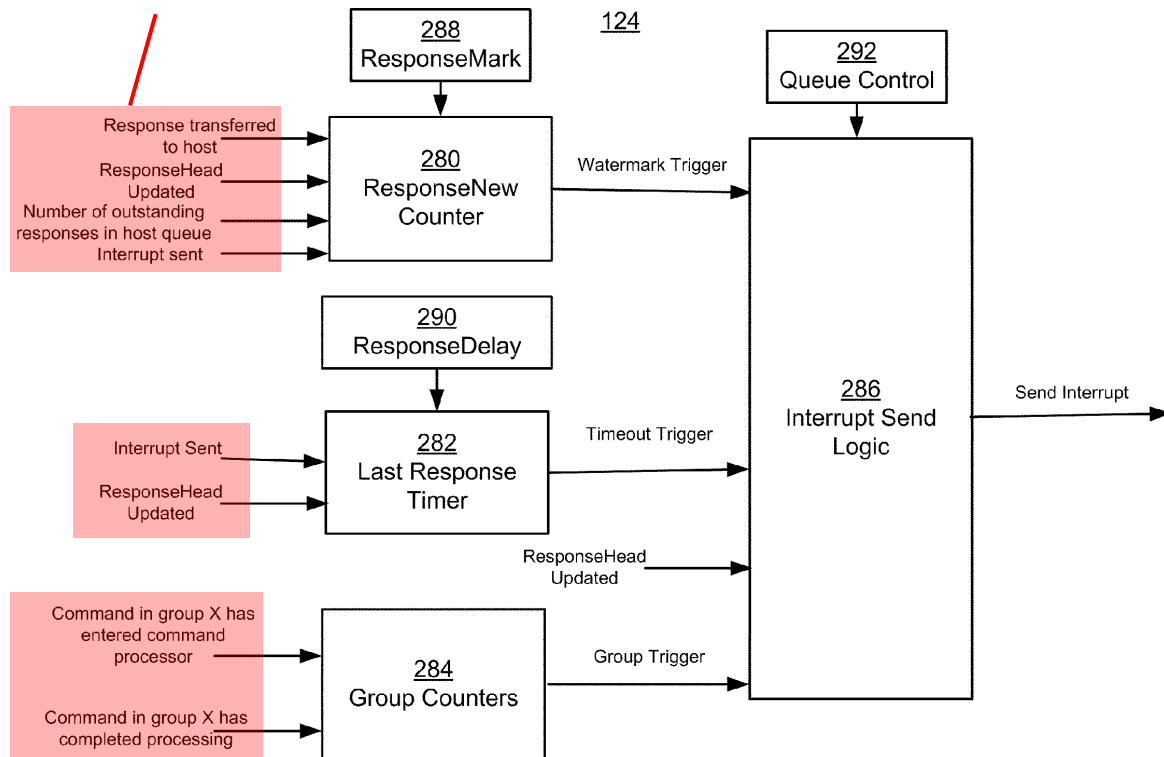


Ex.1005, Fig. 1C (annotated); Ex.1003, ¶88.

Second, Borchers’ interrupt processor 124 (shown in detail in Fig. 2) receives information from the host system as to the status of the response queue (“*completion queue*”). To “*indicate[] the occurrence of a completion queue event associated with the completion queue,*” the response queue status information includes an updated response queue head pointer, and the number of outstanding

items in the response queue: “The ResponseNew counter 280 receives as inputs information including when a packet is transferred to the host 106, when the ResponseHead is updated, the number of outstanding responses in the host response buffer 123 and when an interrupt has been sent.” Ex.1005, [0060].

“completion queue state”



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶89.

Accordingly, because the updated head pointer and updated number of outstanding responses indicates the status of the response queue (*“completion queue”*) that information is a *“completion queue state”* as claimed. In other words, the updated head pointer and updated number of outstanding responses *“indicat[e]*

the occurrence of a completion queue event associated with the completion queue.

The status information received and relied upon by the interrupt controller 124 “*indicat[es] the occurrence of a completion queue event associated with the completion queue*” as claimed. For example, an update to the head pointer of the response queue is a “*completion queue event associated with the completion queue*” because it is an event that directly affects and relates to the response queue (“*completion queue*”).

Further, the information received by the interrupt processor (i.e., “when the ResponseHead is updated” or “the number of outstanding responses in the host response buffer”) represents a “*completion queue state*” because it represents the *status* of the completion queue.

This status information is “*generate[d]*” by the interrupt processor when it is received and/or stored by Borchers’ interrupt processor 124 as illustrated in Fig. 2. To the extent the claims are interpreted such that the *completion queue state* is to be stored, a POSITA would understand that Borchers’ status information would be stored in some type of memory such as a buffer, register, flag, or other form of storage.² In other words, a POSITA would have recognized that Borchers’

² Ex.1003, ¶93. Note that Ground 3 presents an alternative obviousness challenge for this point.

response queue status information (“*completion queue state*”) is not merely transitory signals. Rather, a POSITA would have recognized that the response queue status information would be stored somewhere by the interrupt processor until such information was needed or ready to be processed. Ex.1003, ¶93. Indeed, it was known for memory controllers and interrupt managers to use such types of storage. *See e.g.*, Ex.1008, Figs. 2 & 5, [0057], [0090]-[0091]; Ex.1009, Figs. 2 & 5, [0051], [0083], [0084] (describing how an interrupt manager uses various forms of memory such as cache or registers to store state information). And a POSITA would have found it obvious for the response queue status information to be generated/stored by the interrupt processor given that the ResponseHead being updated is on the controller side: “The command tail pointer and the response head pointer may be stored in registers of the data storage device.” Ex.1005, [0031]; Ex.1003, ¶93. Accordingly, because Borchers’ interrupt processor receives and uses the response queue status information, the interrupt processor “*generate[s]*” a “*completion queue state*” as claimed.

Thus, because Borchers’ interrupt manager receives and uses information as to the status of the response queue, Borchers renders this limitation obvious. Ex.1003, ¶¶88-94.

[1.2] *generate an interrupt vector state based on the completion queue state,*

First, Borchers’ interrupt processor generates an “*interrupt vector state*” by

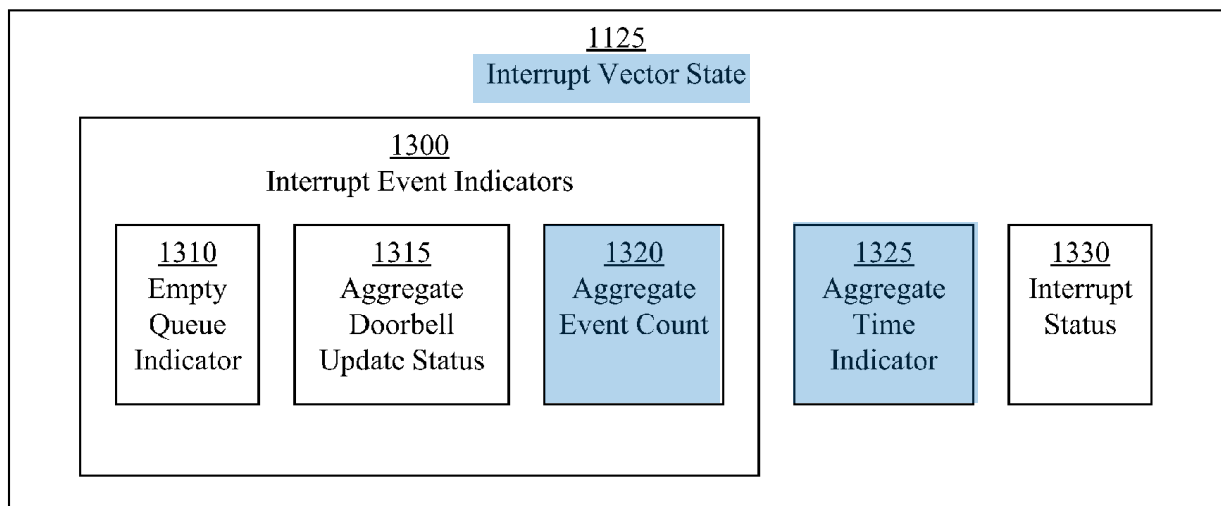
maintaining a set of conditions that determine whether to send an interrupt signal. These set of conditions are maintained by various interrupt mechanisms including “a ResponseNew counter 280, a last response timer 282, group counters 284 and interrupt send logic 286.” Ex.1005, [0059].

The ResponseNew Counter 280, for example, “is configured to track the number of responses transferred to the host 106 that the host has yet to process.” Ex.1005, [0060]. “When the counter 280 reaches or exceed the watermark level set by the host 106, i.e., the ResponseMark 288, then a watermark trigger is generated and sent to the interrupt send logic 286.” Ex.1005, [0060]. If the count is less than the threshold, an interrupt is not sent. Ex.1003, ¶96.

The last response timer 282, for example, is “configured to keep track of time since the last interrupt.” Ex.1005, [0062]. “In one exemplary implementation, the ResponseDelay 290 timeout may be the number of microseconds since the last interrupt, or since the last time that the host 106 removed new responses from the response buffer 123, before an interrupt is generated.” Ex.1005, [0062].

The group counters 284 track the number of unprocessed responses for a number of response queues so that an interrupt can be sent for the whole group. *See* Ex.1005, [0066]. **“In this manner, an interrupt is not generated for each of the individual commands but only for the group of commands.”** Ex.1005, [0066].

Accordingly, because the interrupt processor generates counters and timers (by incrementing the counters and tracking time) for determining whether to send an interrupt signal, Borchers' interrupt processor is configured to generate an “*interrupt vector state*” as claimed. Notably, the interrupt mechanisms of Borchers, which include timers and counters, are similar to how the '968 patent describes the interrupt vector states.



Ex.1001, Fig. 13 (annotated); Ex.1003, ¶99.

As can be seen from Fig. 13 of the '968 patent, the interrupt vector state includes items such as an aggregate event count 1320 or an aggregate time indicator 1325. The aggregate event count 1320 may be used to trigger an interrupt when the number of unprocessed completion packets sent reaches a threshold. See Ex.1001, 19:15-33 (“In this way, the interrupt manager 1040 controls a transmission rate of interrupt message packets to the host processing unit 110

based on the aggregate event count threshold.”). As another example, “[w]hen the aggregate time indicator 1325 is enabled, the interrupt manager 1040 controls a transmission rate of interrupt message packets to the host processing unit 110 based on the aggregate time threshold.” Ex.1001, 19:49-52.

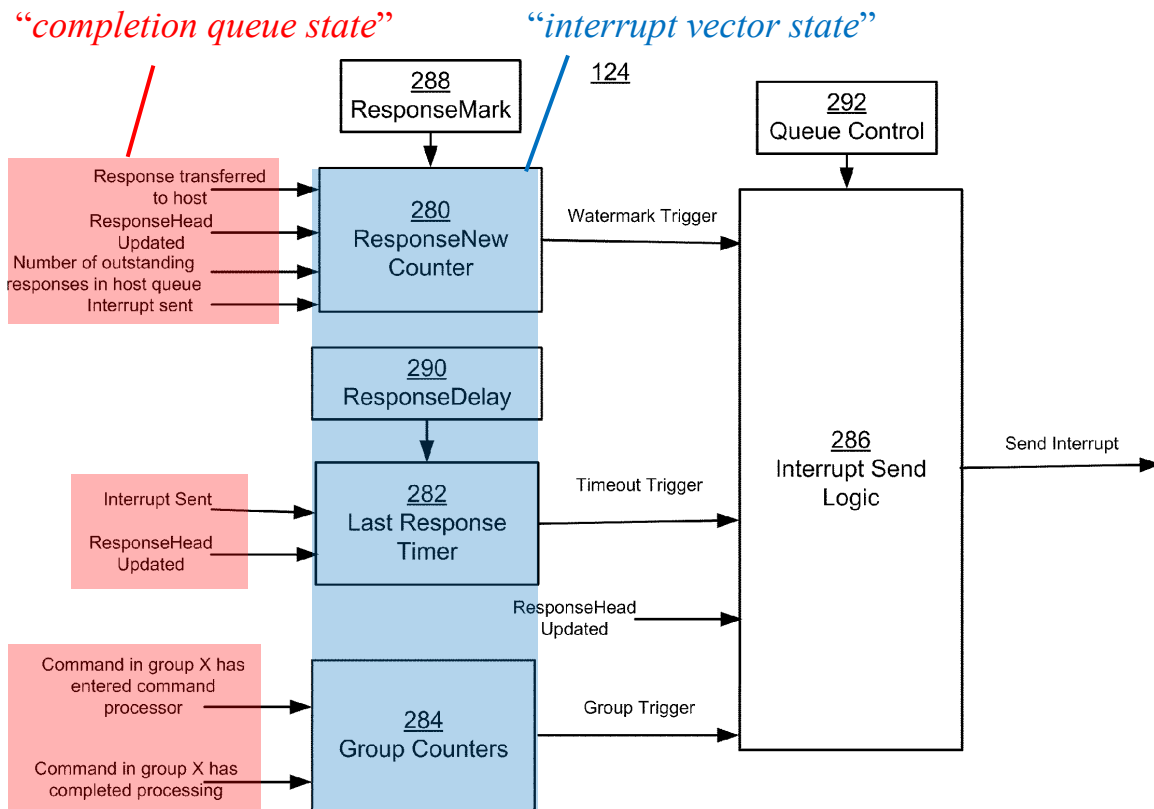
Thus, Borchers’ ResponseNew Counter 280 and last response timer 282 are analogous to the aggregate event count 1320 and aggregate time indicator 1325 which are examples of interrupt vector states described in the ’968 patent.

Second, Borchers’ interrupt mechanisms (e.g., ResponseNew Counter 280, last response timer 282, and group counters) teach an “*interrupt vector state*”. Specifically, Borchers teaches an embodiment in which its memory interface “include[s] a PCIe interface.” Ex.1005, [0110]. It was known that the PCIe standard was required to support Message Signaled Interrupts (MSI), which use interrupt vectors. *See* Ex.1007, 341 (“PCI Express requires support of MSI or MSI-X or both.”) “Message Signaled Interrupts (MSI)” includes “the message address and message data (from here on referred to as the “vector”).” Ex.1006, 231.

A POSITA would have thus recognized based on background knowledge that Borchers’ interrupt signals—as part of a PCIe interface—would be MSI signals that use interrupt vectors. Accordingly, the interrupt mechanisms that maintain the set of conditions for determining whether to send an interrupt signal teach an “*interrupt vector state*” as claimed. Again, Borchers’ interrupt mechanism

are analogous to the interrupt vector state described in the '968 patent. *See* Ex.1001, 19:15-52.

Third, Borchers' interrupt mechanisms such as timers and counters (“*interrupt vector state*”) are based on the response queue status information (“*completion queue state*”) because they receive the completion queue state information as inputs. For example, the “ResponseNew counter 280 receives as inputs information including when a packet is transferred to the host 106, when the ResponseHead is updated, the number of outstanding responses in the host response buffer 123 and when an interrupt has been sent.” Ex.1005, [0060].



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶104.

Accordingly, because the interrupt mechanisms (e.g., timers and counters) (“*interrupt vector state*”) receive information (“*completion queue state*”) as to the status of the response queue (“*completion queue*”), Borchers’ interrupt mechanisms are generated “*based on*” that information. The interrupt mechanism information is *generated* when the timers or counters are updated. In some examples, the values of the timers or counters may be stored in a register. Indeed, a POSITA would have found it obvious for Borchers’ interrupt mechanism information to be stored in some type of memory such as a buffer, register, flag, or other form of storage. Ex.1003, ¶105. In other words, a POSITA would have

recognized that the timers and counters of the interrupt mechanisms (“*interrupt vector state*”) are not merely transitory signals. Indeed, it was known for memory controllers and interrupt managers to use such registers or caches for storage. *See e.g.*, Ex.1008, Figs. 2 & 5, [0057], [0090]-[0091]; Ex.1009, Figs. 2 & 5, [0051], [0083], [0084]] (describing how an interrupt manager uses various forms of memory such as cache or registers to store state information). Accordingly, because Borchers’ interrupt processor receives and uses the response queue status information to update the interrupt mechanism timers and counters, Borchers’ interrupt processor “*generate[s]*” an “*interrupt vector state*” as claimed.

Thus, because Borchers’ interrupt processor uses counters and timers to maintain the set of conditions used to determine whether to send an interrupt, and the counters or timers are updated with information as to the status of the response queue, Borchers renders this limitation obvious. Ex.1003, ¶¶95-107.

[1.3] *determine the completion queue of the host processing unit contains an unprocessed completion status based on the interrupt vector state, and*

First, as described above at [1.0], Borchers’ host unit includes a response queue (“*completion queue of the host processing unit*”).

Second, Borchers’ ResponseNew Counter 280 (“*interrupt vector state*”) “is configured to track the number of responses transferred to the host 106 that the host has yet to process” (“*unprocessed completion status*”). Ex.1005, [0060].

“Each time a response is transferred to the response buffer 123 the counter is incremented.” Ex.1005, [0060]. Accordingly, the ResponseNew counter 280 indicates the number of unprocessed events in the response queue (“*completion queue*”). “If the count of new responses in the response buffer 123 is equal to or greater than the ResponseMark, then an interrupt is generated by the interrupt processor 124 and the count is zeroed.” Ex.1005, [0037]. Accordingly, the interrupt processor determines there are unprocessed completion statuses in the response queue (“*the completion queue of the host processing unit*”) “*based on*” the receipt of the number of outstanding responses in the response queue by the ResponseNew counter (part of the “*interrupt vector state*”).

Thus, because the ResponseNew Counter 280 (an “*interrupt vector state*”) tracks the number of responses the host has yet to process, Borchers renders this limitation obvious. Ex.1003, ¶¶108-12.

[1.4] generate an interrupt message packet for triggering an interrupt in the host processing unit to alert the host processing unit of the unprocessed completion status in the completion queue, and

First, as explained above at [1.0], Borchers’ system sends an interrupt to alert the host processing unit of the unprocessed completion status in the response queue (“*completion queue*”). “The interrupt processor 124 is configured to signal the host 106 with an interrupt when new responses are available in the response buffer 123.” Ex.1005, [0035]. “The interrupt processor 124 is configured to signal

the host 106 with an interrupt when new responses are available in the response buffer 123.” The interrupt signal received by the host then causes the host to process the interrupt, thereby “*triggering the interrupt in the host processing unit.*” See e.g., Ex.1005, [0050] (“an interrupt may be received by an interrupt handler 225a-225n”).

Second, a POSITA would have found it obvious for Borchers’ interrupt signal to be an “interrupt message packet” consistent with PCI/PCIe standards. Ex.1003, ¶114. Borchers teaches an embodiment in which its memory interface “include[s] a PCIe interface” ([0110]), and as noted previously, the PCIe standard “requires” Message Signaled Interrupts (MSI) (*see* Ex.1006, 231, 249).

Accordingly, it would have been obvious to a POSITA that when Borchers describes sending interrupt signals, those signals would be embodied as MSI or MSI-X interrupts consistent with the PCI/PCIe standards. Borchers’ interrupt signals—embodied as MSI or MSI-x interrupts—are therefore “*interrupt message packets.*” Such MSI or MSI-x interrupt signals are within the scope of the claims. Indeed, the ’968 patent similarly describes the interrupt message packets as being MSI or MSI-x interrupts: “In some embodiments, the hardware interrupt generated in the host processing unit 110 is a message signal interrupt (MSI) or an extended message signal interrupts (MSI-X) as defined in the PCI Specification 3.0.” Ex.1001, 16:61-64.

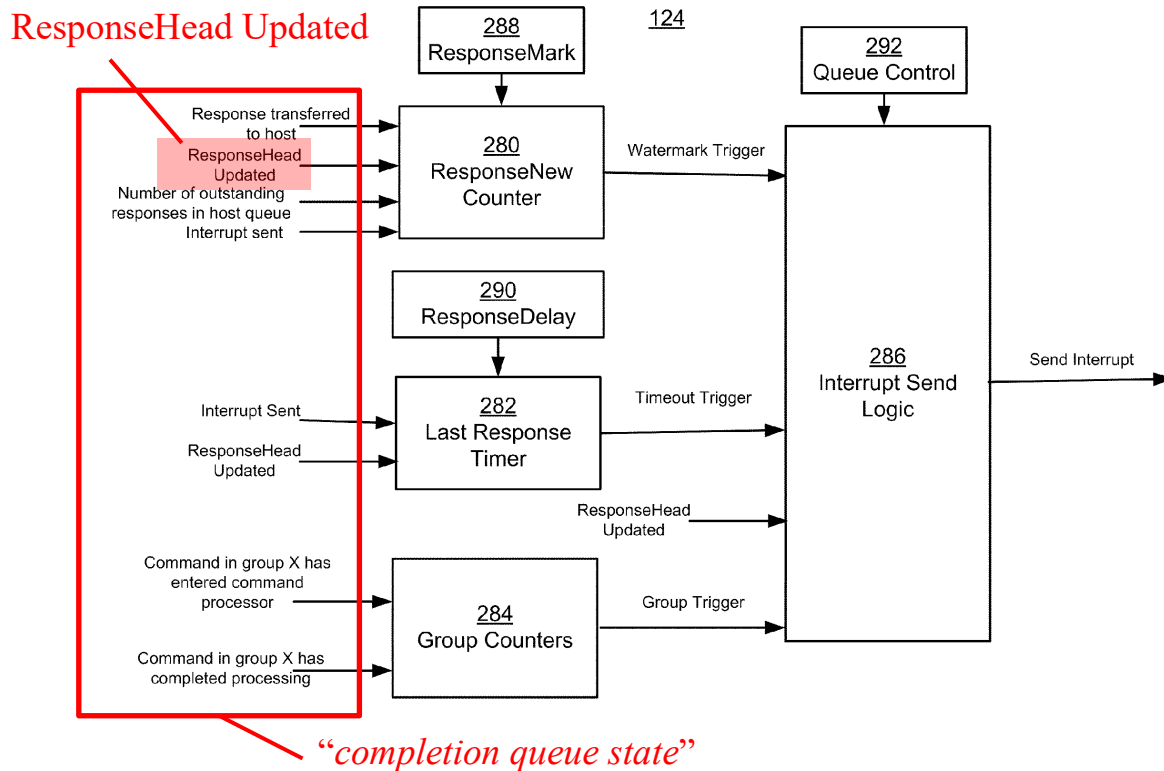
Thus, because Borchers' interrupt manager sends an interrupt signal to alert the host that there are unprocessed events in the response queue, Borchers renders this limitation obvious. Ex.1003, ¶¶113-16.

[1.5] wherein the completion queue state includes a doorbell update status indicating whether the host processing unit has performed a doorbell update event in which the host processing unit updates a head pointer stored in the nonvolatile memory controller for the completion queue.

First, as mentioned above at [1.1], Borchers' interrupt processor receives response queue status information ("*completion queue state*") from the host.

Second, Borchers' response queue status information ("*completion queue state*") indicates whether the host processing unit has updated the response head pointer ("*the host processing unit updates a head pointer*"). As explained above at [1.1], Borchers' interrupt processor 124 receives as information from the host system "when the ResponseHead is updated." Ex.1005, [0060].

During operation, "[t]he host 106 is configured to read the responses from the response buffer 123 and **update the head pointer** in the MMIO space of the data storage device to indicate that the host has received the response." Ex.1005, [0035].



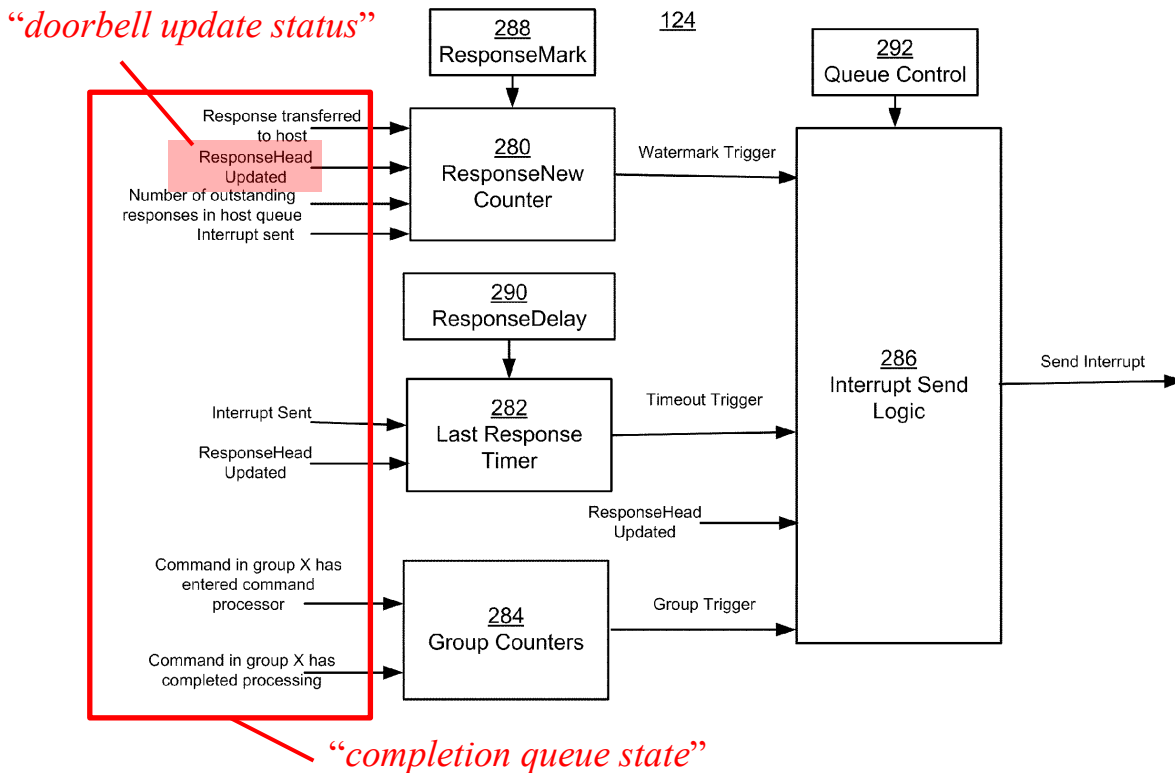
Ex.1005, Fig. 2 (annotated); Ex.1003, ¶119.

Third, the indication that the host has updated the response head pointer is a “doorbell update status” as claimed. Such a mechanism that indicates when the response head pointer has been updated is often referred to as a “doorbell” update status. Ex.1003, ¶120. Using a response queue that is produced by the memory controller and consumed by the host is referred to as the producer/consumer model. Ex.1003, ¶120. Indeed, Borchers explains that the “data storage device 100 may be the ... producer of the response buffer 123” and the “host 106 may be the ... consumer of the response buffer 123.” Ex.1005, [0030].

The PCI standard explains that in this Producer-Consumer model, a

notification of the head pointer being updated is a “doorbell” event. Ex.1006, 308; Ex.1003, ¶121. Thus, the host (consumer of the response queue) may “signal the Producer using an interrupt or **another doorbell register.**” Ex.1006, 308.

A POSITA would have thus recognized that the register that stores the head pointer is a doorbell update register. Ex.1003, ¶122. Accordingly, the notification that the head pointer of the response queue has been updated is a “*doorbell update status*” as claimed. Indeed, the claim language itself defines what a doorbell update status does—an indication “*in which the host processing unit updates a head pointer ... for the completion queue.*” Borchers does precisely that by sending a notification that the head pointer of the response queue (“*completion queue*”) has been updated. Accordingly, Borchers’ notification that the head pointer of the response queue is a “*doorbell update status*” as claimed.



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶122.

Fourth, Borchers’ head pointer is stored in a register on the storage device controller (“head pointer stored in the nonvolatile memory controller for the completion queue”). Borchers’ “data storage device can include ... a fourth register configured to store responsehead pointer values.” Ex.1005, [0005]; *see also* Ex.1005, [0030] (“[A]nd values of the head pointer of the response buffer can be stored in a register on the data storage device 100.”). Borchers further explains that the register that stores the head pointer may be within the controller: “The command tail pointer and the response head pointer may be stored in registers of the data storage device 100 in memory mapped I/O space within the

controller 110.” Ex.1005, [0031].

Thus, because Borchers’ interrupt processor maintains information about the response queue, including that the response head pointer has been updated, and the head pointer is stored in a doorbell register within the memory controller, Borchers renders this limitation obvious. Ex.1003, ¶¶117-25.

4. Claim 2

[2.1] The nonvolatile memory controller of claim 1, wherein the interrupt is a hardware interrupt.

Borchers controller 110, which includes the interrupt processor 124, may be “implemented in hardware.” Ex.1005, [0113]. Thus, the interrupts sent by the interrupt processor are hardware interrupts.

Additionally, the specification of the ’968 patent refers to MSI and MSI-x interrupts as hardware interrupts: “In some embodiments, **the hardware interrupt** generated in the host processing unit 110 **is a message signal interrupt (MSI) or an extended message signal interrupts (MSI-X)** as defined in the PCI Specification 3.0.” Ex.1001, 16:61-64. For the reasons explained above at [1.4], a POSITA would have found it obvious for Borchers’ PCIe interrupts to be MSI or MSI-x signals associated with the PCI express standard.

Thus, because Borchers’ memory controller and interrupt processor may be implemented in hardware, and may be MSI or MSI-x interrupt signals, Borchers

renders this limitation obvious. Ex.1003, ¶¶126-28.

5. Claim 3

[3.1] *The nonvolatile memory controller of claim 1, wherein the interrupt message packet includes an interrupt vector associated with the completion queue for triggering the interrupt in the host processing unit.*

First, as explained above at [1.4], a POSITA would have found it obvious—based on background knowledge—for Borchers’ interrupt signals to be MSI or MSI-x interrupt signals. Such interrupt signals include both a message address and message data, which are referred to as an interrupt vector. *See* Ex.1006, 231 (“System software initializes the message address and message data (from here on referred to as the ‘vector’)”). Accordingly, Borchers’ interrupt signals include an “*interrupt vector.*”

Second, Borchers’ interrupt signals trigger an interrupt in the host processor. “The interrupt processor 124 is configured to signal the host 106 with an interrupt when new responses are available in the response buffer 123.” Ex.1005, [0035]. The interrupt signal received by the host then causes the host to process the interrupt, thereby “*triggering the interrupt in the host processing unit.*” *See e.g.*, Ex.1005, [0050] (“an interrupt may be received by an interrupt handler 225a-225n”).

Thus, because it would have been obvious for Borchers’ interrupt signals, which trigger interrupts in the host, to include vectors consistent with PCI

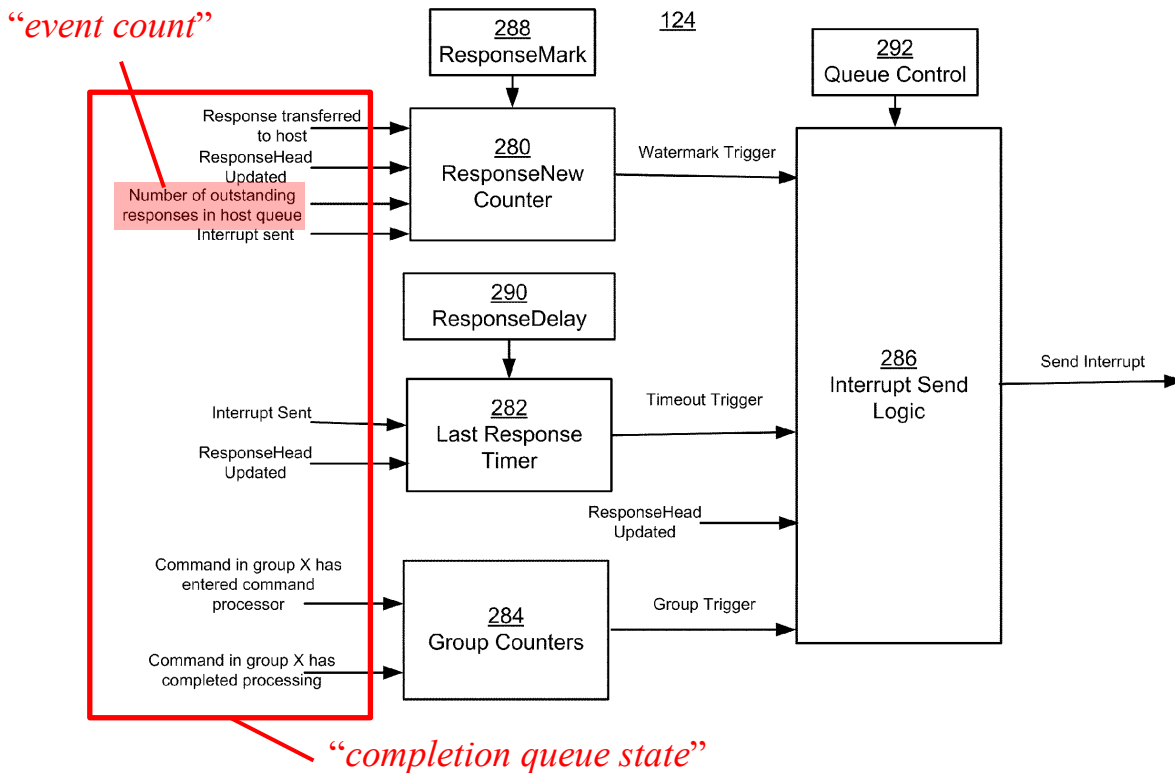
standards, Borchers renders this limitation obvious. Ex.1003, ¶¶129-32.

6. Claim 4

[4.1] *The nonvolatile memory controller of claim 3, wherein completion queue state includes an event count indicating a number of completion message packets received by the host controller interface for the completion queue associated with the completion queue state.*

First, as explained above at [1.1], Borchers' interrupt processor generates response queue status information ("*completion queue state*").

Second, that information includes the number of outstanding responses in the response queue ("*event count*"). "The ResponseNew counter 280 receives as inputs information including ... **the number of outstanding responses** in the host response buffer 123 and when an interrupt has been sent." Ex.1005, [0060].



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶134.

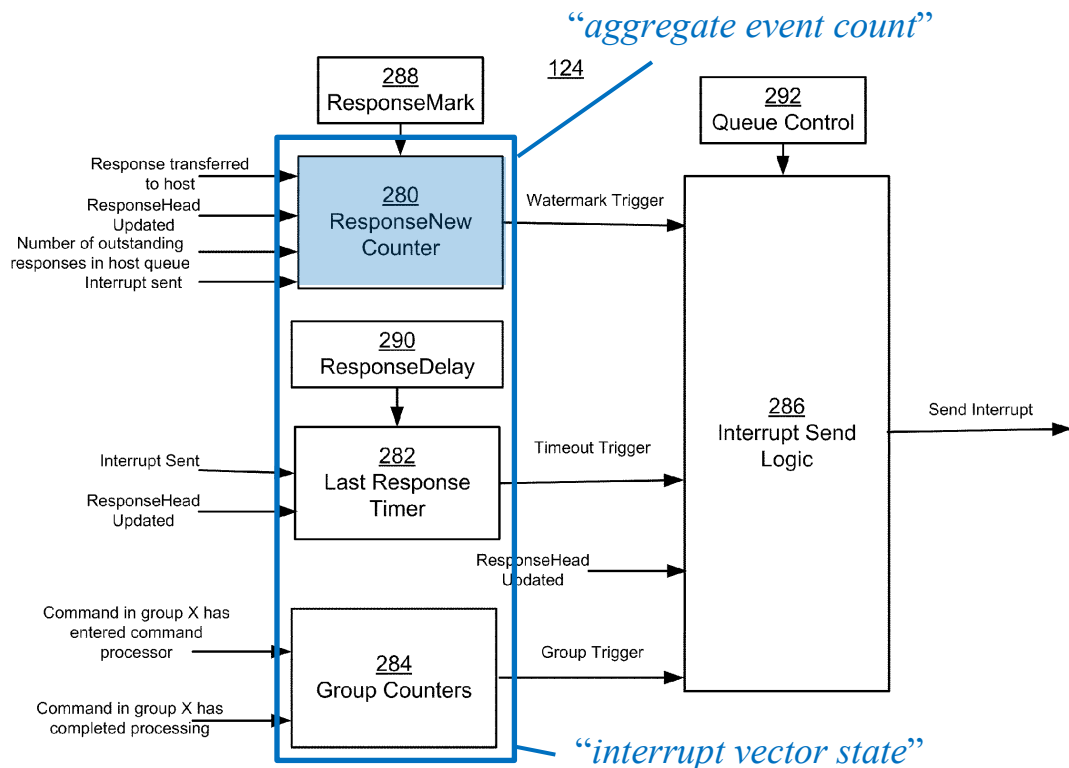
Thus, because Borchers maintains status information including the number of outstanding responses in the host's response queue, Borchers renders this limitation obvious. Ex.1003, ¶¶133-35.

7. Claim 5

[5.1] The nonvolatile memory controller of claim 4, wherein the interrupt vector state includes an aggregate event count for aggregating event counts of the completion queue state over a plurality of update operations, and wherein the interrupt manager is inhibited from generating an interrupt message packet if the aggregate event count is below an aggregate event count threshold.

First, Borchers' interrupt mechanisms ("interrupt vector state") include a ResponseNew Counter 280 ("aggregate event count"). See Ex.1005, [0060]. The

ResponseNew Counter receives as an input various pieces of information as to the status of the response queue (“*completion queue state*”). One such piece of information is when an event is sent to the host. “The ResponseNew Counter 280 is configured to track the number of responses transferred to the host 106 that the host has yet to process. Each time a response is transferred to the response buffer 123 the counter is incremented.” Ex.1005, [0060]. Accordingly, because the ResponseNew Counter 280 is incremented over multiple events, the ResponseNew counter 280 is for “*aggregating event counts of the completion queue state over a plurality of update operations.*”



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶137.

Second, Borchers' interrupt processor 124 does not send an interrupt signal until the ResponseNew Counter 280 ("*aggregate event count*") reaches a watermark level ("*the interrupt manager is inhibited from generating an interrupt message packet if the aggregate event count is below an aggregate event count threshold*"). "In one exemplary implementation, the host 106, through its driver 107, may control when the interrupt processor 124 should generate interrupts." Ex.1005, [0036] "For instance, the host 106 through the driver 107 may configure the interrupt processor 124 to use a water mark interrupt mechanism." Ex.1005, [0036]. "If the count of new responses in the response buffer 123 is equal to or greater than the ResponseMark [*("threshold")*], then an interrupt is generated by the interrupt processor 124 and the count is zeroed." Ex.1005, [0037].

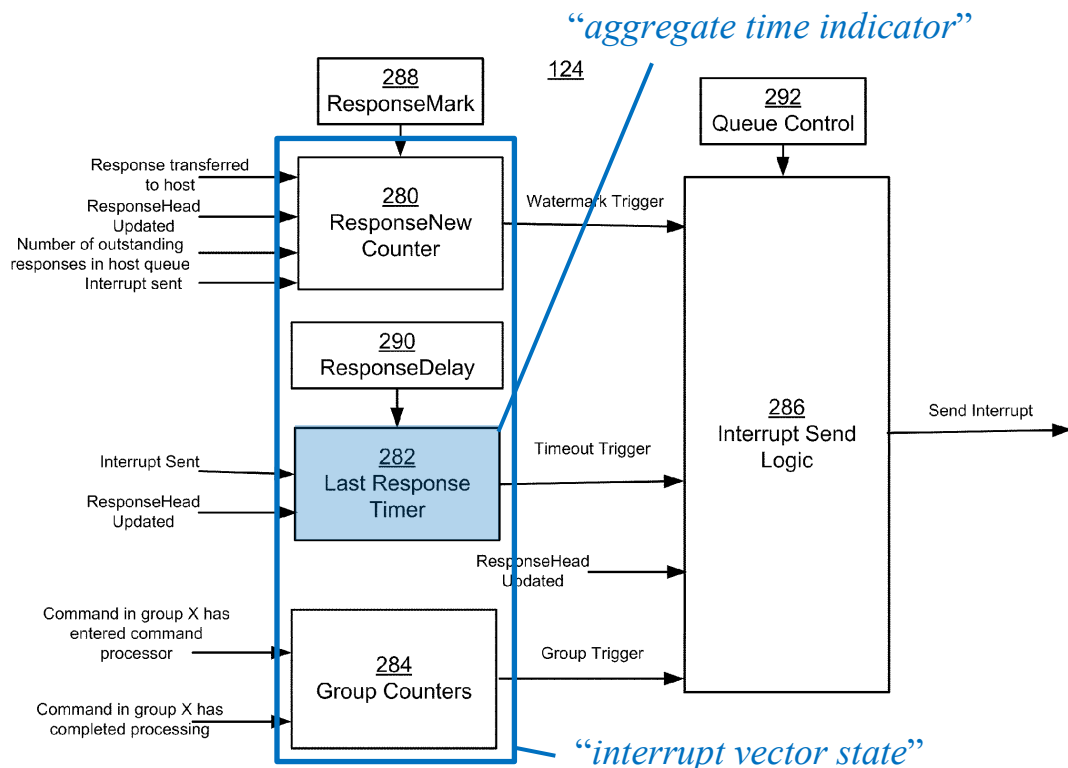
Thus, because the ResponseNew Counter gets incremented when the interrupt processor places events in the response queue, and does not send an interrupt until the watermark threshold is met, Borchers renders this limitation obvious. Ex.1003, ¶¶136-39.

8. Claim 6

[6.1] *The nonvolatile memory controller of claim 3, wherein the interrupt vector state includes an aggregate time indicator for indicating, a time period between the last time the interrupt manager generated an interrupt message packet for the completion queues associated with the interrupt vector state.*

Borchers' interrupt mechanisms ("*interrupt vector state*") include a Last

Response timer 282 (“*aggregate time indicator*”). “The last response timer 282 may be configured to keep track of time since the last interrupt.” Ex.1005, [0062]. “The last response timer 282 also may receive a signal when the ResponseHead is updated, which indicates that the host 106 has removed responses from the response buffer 123.” Ex.1005, [0063]. When the timer reaches its ResponseDelay parameter (“*threshold*”), “and the response buffer 123 contains new responses, then a timeout trigger signal is sent to the interrupt send logic 286.” Ex.1005, [0064].



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶140.

Accordingly, Borchers’ last response timer 282 is for keeping track of the

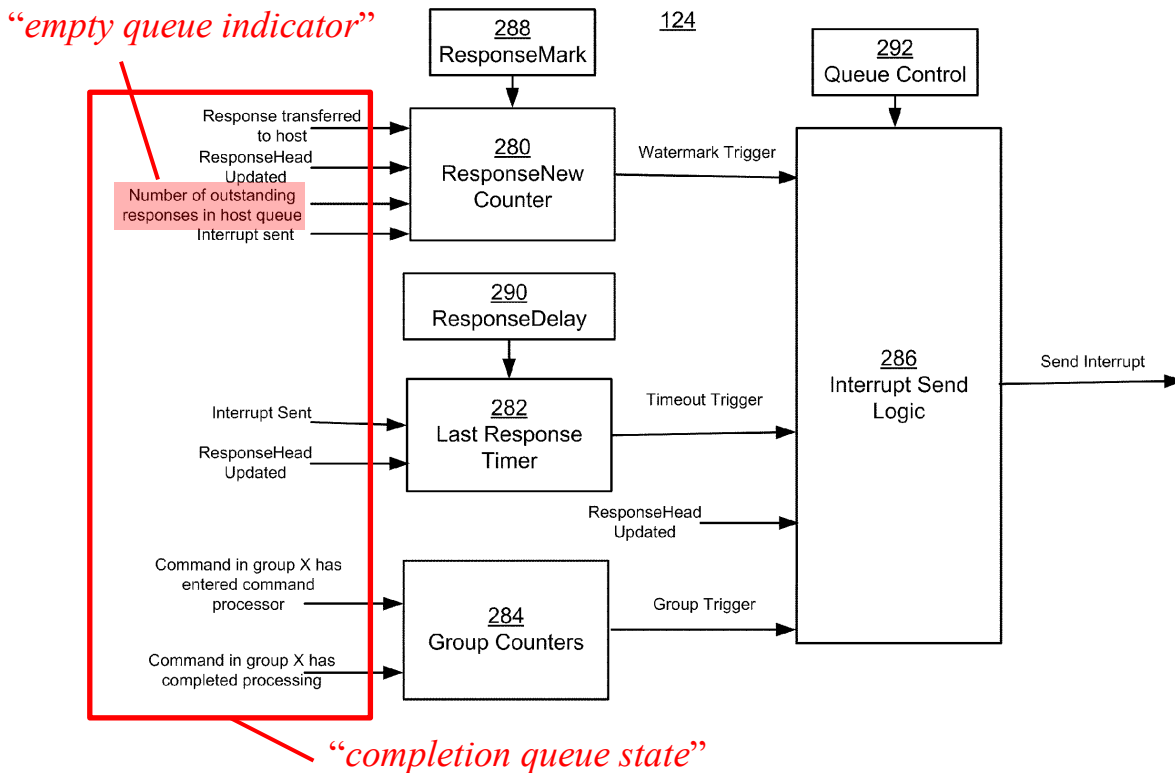
amount of time since the last interrupt was sent (*“for indicating, a time period between the last time the interrupt manager generated an interrupt message packet for the completion queues associated with the interrupt vector state.”*)

Thus, because Borchers’ Last Response Timer indicates the amount of time since the interrupt processor last sent an interrupt signal to the host, Borchers renders this limitation obvious. Ex.1003, ¶¶140-42.

9. Claim 7

[7.1] *The nonvolatile memory controller of claim 3, wherein the completion queue state includes an empty queue indicator for indicating whether the completion queue is empty, and wherein the interrupt manager is inhibited from generating an interrupt message packet if the empty queue indicator indicates the completion queue is empty.*

First, as explained above at [4.1], Borchers’ response queue status information (*“completion queue state”*) includes the number of outstanding responses in the response queue. Ex.1005, [0060]. When that number is zero, it is an *“empty queue indicator”* that indicates the response queue (*“completion queue”*) is empty. Accordingly, the information that indicates the number of outstanding responses in the response queue acts as an empty queue indicator when the number is zero.



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶143.

Second, Borchers’ system does not send an interrupt if there are no outstanding responses in the response queue (*“wherein the interrupt manager is inhibited from generating an interrupt message packet if the empty queue indicator indicates the completion queue is empty”*). Rather, the interrupt send logic 286 generates interrupts only if the watermark level is reached, the timer is triggered, or the group counter is triggered. *See* Ex.1005, [0036]-[0037]. A POSITA would have further understood that there would be no reason to send an interrupt if there are no responses in the response queue for the host to process. Ex.1003, ¶144.

Thus, because the response queue status information includes the number of

outstanding responses in the host queue, which may be zero, and the processor does not needlessly send interrupts when there are no responses in the response queue to process, Borchers renders this limitation obvious. Ex.1003, ¶¶143-46.

10. Claim 8

[8.0] *A nonvolatile memory controller comprising:*

For the reasons explained at [1.0], Borchers teaches a “*nonvolatile memory controller*” as claimed. Ex.1003, ¶147.

[8.1] *a completion queue state memory configured to store a plurality of completion queue states corresponding to a plurality of completion queues in a host processing unit;*

First, as explained above at [1.1], Borchers teaches a “*completion queue state*” as claimed. Ex.1003, ¶148.

Second, the memory location where Borchers’ response queue status information is stored or buffered (e.g., RAM, a register, a flag), is “*a completion queue state memory*” as claimed. As one example, Borchers’ memory controllers includes “register 161” that may be used to store or buffer such status information. Ex.1005, [0030].

Given that the register stores items such as head and tail pointers, it would have been obvious for register 161 to buffer or temporarily store the response queue status information. Indeed, it was known to store such information in a buffer or register. *See e.g.*, Ex.1008, Figs. 2 & 5, [0057], [0090]-[0091]; Ex.1009,

Figs. 2 & 5, [0051], [0083], [0084] (describing how an interrupt manager uses various forms of memory such as cache or registers to store state information). Accordingly, the storage location used by Borchers' interrupt processor to store the interrupt mechanism information is a “*a completion queue state memory configured to store a plurality of completion queue states.*”

Third, Borchers describes an example in which there are a plurality of response queues in the host (“*a plurality of completion queues in a host processing unit*”). “Although FIG. 1A illustrates one command buffer 119 and one response buffer 123, multiple pairs of buffers may be used, as illustrated in FIG. 1B.”

Ex.1005, [0028]; *see also* [0046]-[0058].

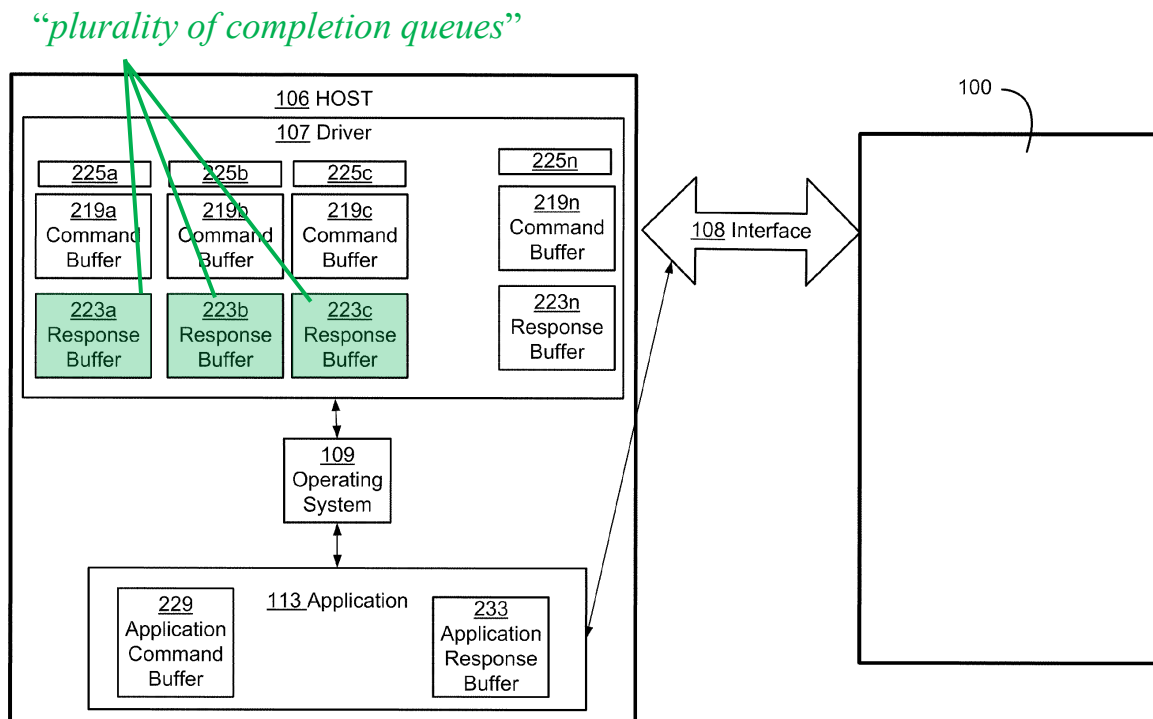


FIG. 1B

Ex.1005, Fig. 1B (annotated); Ex.1003, ¶150.

As also explained above, Borchers explains that the command and response queues of Fig. 1A may be implemented as circular queues: “In one exemplary implementation shown in FIG. 1C the command buffer 119 can be configured as a circular queue 159 that is used to communicate information from the host 106 and to the data storage device 100 of FIG. 1A.” Ex.1005, [0030]. Accordingly, it would have been obvious for Borchers’ host to include multiple buffer pairs (as shown in Fig. 1B and described in accompanying text) which are implemented as circular queues (as shown in Fig. 1C and described in accompanying text). Indeed, when describing the buffer pairs of Fig. 1B, Borchers explains that “Each of the buffer

pairs may include its own set of head and tail pointers. The use of the head and tail pointers may be the same as described above with respect to the command buffer 119 and the response buffer 123 of FIG. 1A.” Ex.1005, [0049].

As explained above at [1.1], Borchers describes that the interrupt processor 124 receives status information as to the response queue. Accordingly, in the example where there are multiple response queues, it would have been obvious that the interrupt processor receives status information for each of the response queues. The status information for each of the response queues thus teaches “*a plurality of completion queue states corresponding to a plurality of completion queues*” as claimed.

Thus, because Borchers’ interrupt controller receives and utilizes response queue status information for a plurality of buffers, Borchers renders obvious “*a completion queue state memory configured to store a plurality of completion queue states corresponding to a plurality of completion queues in a host processing unit*” as claimed. Ex.1003, ¶¶148-53

Additionally, Borchers teaches a “*plurality of completion queue states*” in that there are several pieces of status information, including that a Response has been transferred to the host, the response queue head pointer has been updated, and the number of outstanding responses in the host queue. *See* Ex.1005, Fig. 2; Ex.1005, [0060]; Ex.1003, ¶154.

[8.2] *an interrupt vector state memory configured to store a plurality of interrupt vector states corresponding to a plurality of interrupt vectors in the host processing unit; and*

First, as explained above at [1.2], Borchers teaches various interrupt mechanisms that teach an “*interrupt vector state*” as claimed. These interrupt mechanisms are used to determine whether to send an interrupt signal to alert the host that there are responses in the response queue that have yet to be processed. *See* Ex.1005, [0060].

Second, as explained above at [8.1], Borchers describes an example in which there are multiple response queues on the host. *See* Ex.1005, Fig. 1B, [0046]-[0048]. Accordingly, in such case, there would be multiple interrupt mechanisms—one for each response queue. For example, there would be a ResponseNew Counter 280 for each response queue. There would be a last response timer 282 for each response queue. The multiple ResponseNew Counters or last response timers thus teach “*a plurality of interrupt vector states*” as claimed.

Additionally, Borchers teaches “*a plurality of interrupt vector states*” in that there are multiple types of interrupt vector states. For example, the ResponseNew counter 280 and last response timer 282 together teach “*a plurality of interrupt vector states*” as claimed.

Third, as with the completion queue state memory described above at [8.1],

the memory used by the interrupt processor 124 to store the interrupt mechanism information (e.g., response count or time) teaches an “*interrupt vector state memory*” as claimed. For example, whether the interrupt mechanism information is stored in register 161 or some other register or location, such storage location teaches an “*interrupt vector state memory*” as claimed.

Fourth, for the reasons explained above at [1.2], it would have been obvious for Borchers’ interrupt signals to be implemented as interrupt vectors consistent with the PCI standard. Accordingly, each of Borchers’ interrupt mechanisms corresponds to a different interrupt vector (“*a plurality of interrupt vector states corresponding to a plurality of interrupt vectors*”).

Thus, because Borchers’ host includes a plurality of response buffers, and the memory controller utilizes interrupt mechanisms that determine when to send an interrupt signal for unprocessed responses in those buffers, Borchers renders this limitation obvious. Ex.1003, ¶¶155-60.

Additionally, Borchers teaches a “*plurality of interrupt vector states*” in that there are multiple interrupt mechanisms. These multiple interrupt mechanisms include a ResponseNew Counter 280, a last response timer 282, and a group counter 284. *See* Ex.1005, Fig. 2; Ex.1005, [0060]; Ex.1003, ¶161.

[8.3] an interrupt manager controller coupled to the completion queue state memory and the interrupt vector state memory,

First, as described above at [1.1], Borchers' memory controller includes an interrupt processor 124 (*"interrupt manager controller"*).

Second, the interrupt manager controller utilizes the response queue status information (*"completion queue state"*) and interrupt mechanisms (*"interrupt vector state"*). Accordingly, the interrupt processor is "coupled to" the storage locations of the status information and interrupt mechanism information. For example, the interrupt processor is coupled with register 161, which is one option where the response queue status information and interrupt mechanism information may be stored.

"interrupt vector state memory" and "completion queue state memory"

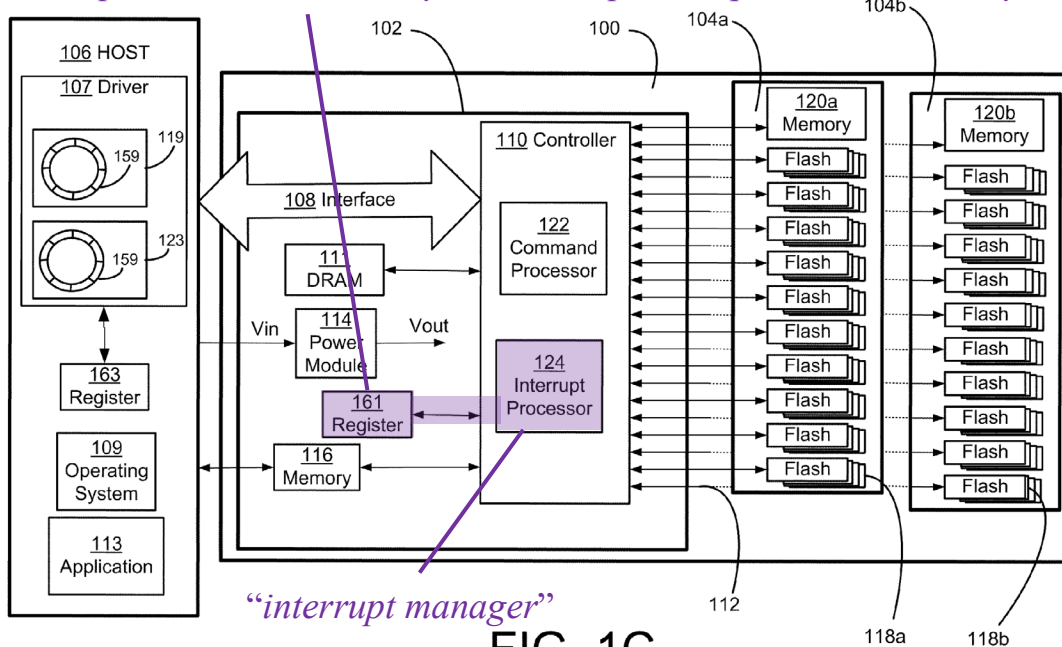


FIG. 1C

Ex.1005, Fig. 1C (annotated); Ex.1003, ¶163.

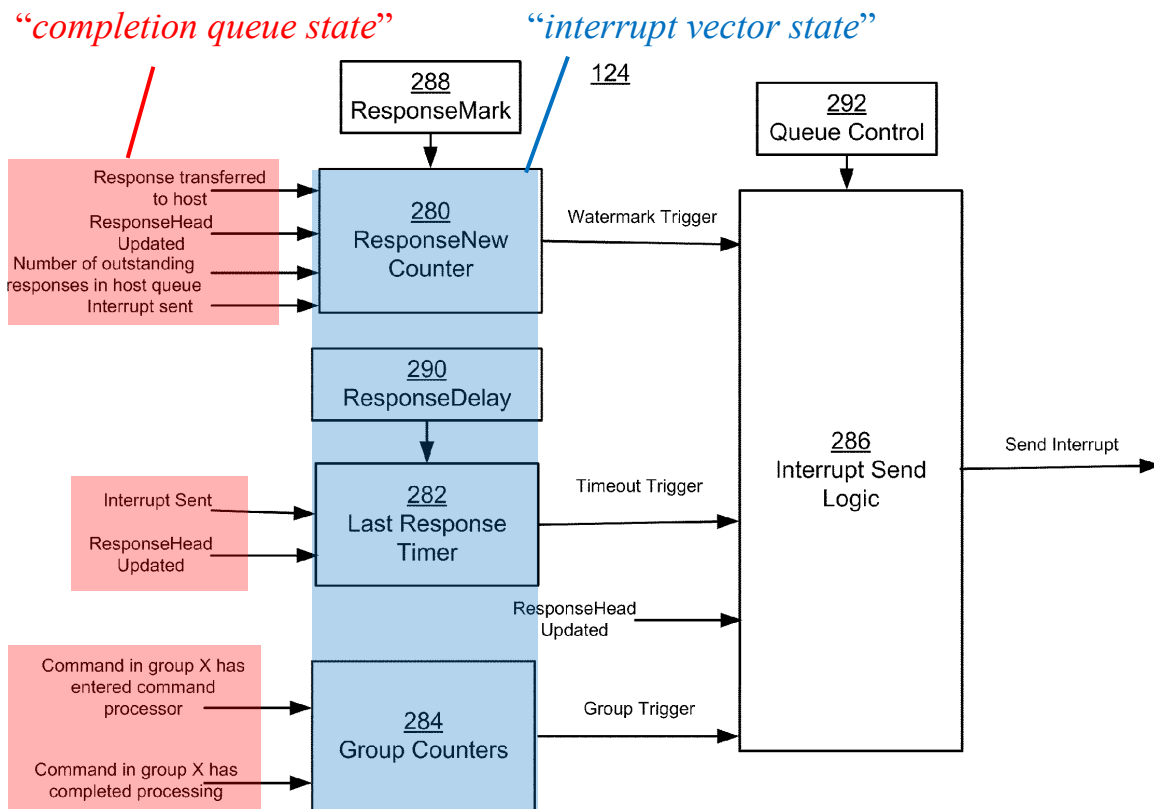
Thus, because Borchers describes an interrupt processor that utilizes response queue status information and interrupt mechanism information stored in a type of data storage, Borchers renders this limitation obvious. Ex.1003, ¶¶162-64.

[8.4] *the interrupt manager controller configured to map at least one completion queue state of the plurality of completion queue states to an interrupt vector state of the plurality of interrupt vector states,*

Borchers explains that when there are multiple command/response queue pairs as shown in Fig. 1B, each buffer pair is processed and handled separately. “The command buffers 219a-219n and the response buffers 223a-223n may be configured to operate and function as described above with respect to the command buffer 119 and the response buffer 123 of FIG. 1A.” Ex.1005, [0049]. “The multiple different head and tail pointers, each of which corresponds to a buffer pair, may be stored on the host 106, the data storage device 100 or a combination of the host 106 and the data storage device 100.” Ex.1005, [0049]. “In some instances, an interrupt may be received by an interrupt handler 225a-225n when a related group of commands has been processed by the data storage device, as discussed in more detail below with respect to FIG. 2.” Ex.1005, [0050]. “Each of the buffer pairs may be granted access to any address mapping, which may be stored on the host 106 and/or on the data storage device 100.” Ex.1005, [0051].

Accordingly, each buffer pair is handled separately in the manner described with Fig. 2. Fig. 2 describes the response queue status information (“*completion*

queue state”) and interrupt mechanisms (*“interrupt vector state”*) as shown below. Thus, for each separate command/response buffer pair, the corresponding response queue status information is associated with the corresponding interrupt mechanisms for that pair (*“map at least one completion queue state of the plurality of completion queue states to an interrupt vector state of the plurality of interrupt vector states”*).

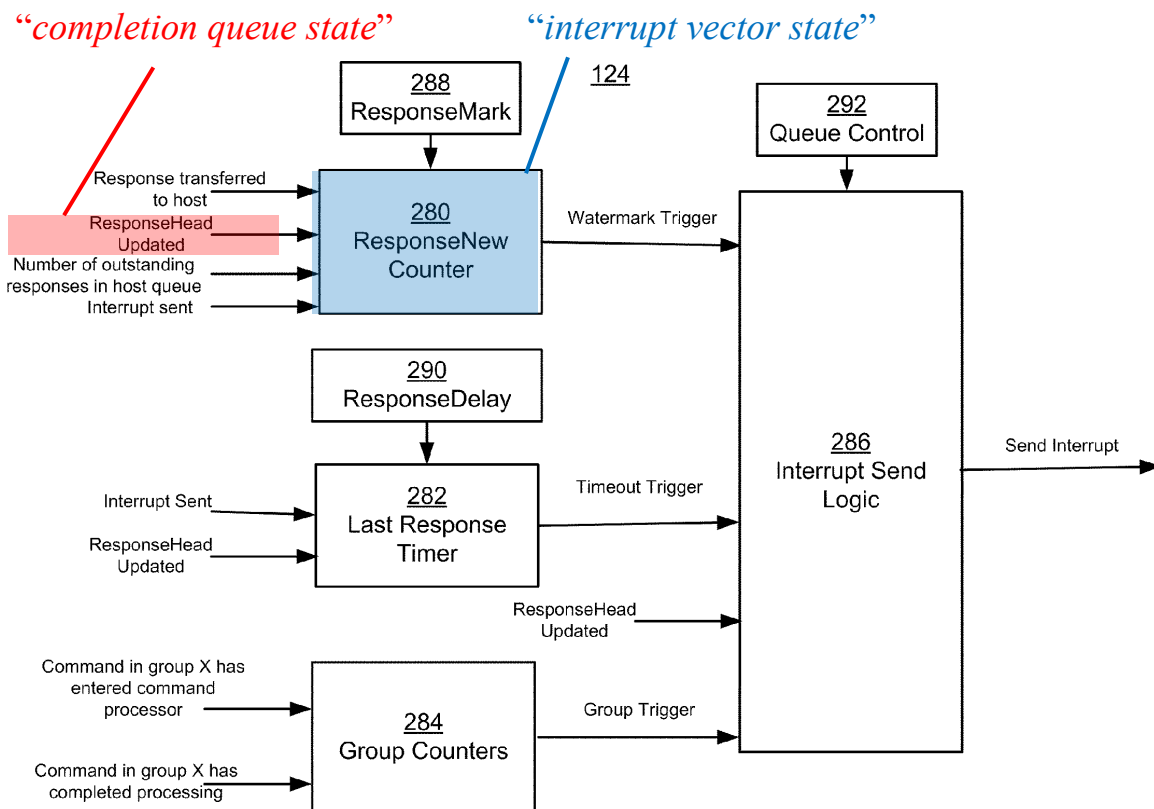


Ex.1005, Fig. 2 (annotated); Ex.1003, ¶166.

Thus, because Borchers’ interrupt controller includes interrupt mechanisms for each separate command/response buffer pair, Borchers renders this limitation

obvious. Ex.1003, ¶¶165-67.

Additionally, Borchers teaches a “*plurality of completion queue states*” in that there are several pieces of status information, including that a Response has been transferred to the host, the response queue head pointer has been updated, and the number of outstanding responses in the host queue. At least one of these pieces of information is mapped to one of the interrupt mechanisms. For example, the ResponseHead updated piece of information is mapped to the ResponseNew Counter 280 as shown below.



Borchers, Fig. 2 (annotated); Ex.1003, ¶168.

Thus, because Borchers maps at least one of the response queue status items (ResponseHead Updated) to one of the interrupt mechanisms (ResponseNew Counter 280), Borchers renders this limitation obvious. Ex.1003, ¶¶168-69.

[8.5] *generate the interrupt vector state based on each completion queue state mapped to the interrupt vector state,*

First, for the reasons explained above at [1.2], Borchers generates interrupt mechanism information (“*interrupt vector state*”) based on the status information of the response queue (“*each completion queue state mapped to the interrupt vector state*”).

Second, in the case where there are multiple command/response buffer pairs as described in Fig. 1B (and described above at [8.1]), Borchers’ system would include counters and timers (“*interrupt vector state[s]*”) for each command/response queue. Accordingly, each of the counters/timers are “*based on*” their respective response queue status information (“*completion queue state*”). Ex.1003, ¶¶170-71.

[8.6] *determine a completion queue corresponding to a completion queue state mapped to the interrupt vector state contains an unprocessed completion status based on the interrupt vector state, and*

For the reasons explained above at [1.3], Borchers renders this limitation obvious. Ex.1003, ¶172.

[8.7] *to generate an interrupt message packet based on the interrupt vector state for triggering an interrupt in the host processing unit to alert the host processing*

unit of the unprocessed completion status in the completion queue.

For the reasons explained above at [1.4], Borchers renders this limitation obvious. Ex.1003, ¶173.

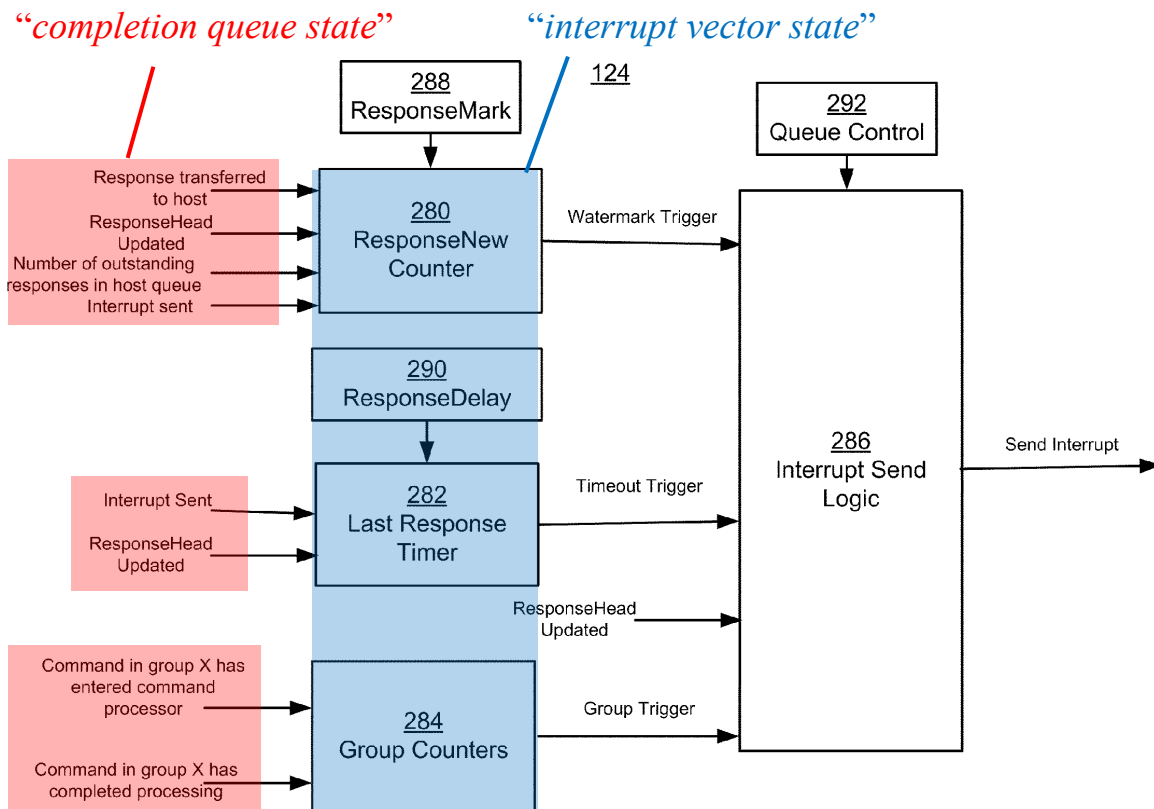
11. Claim 9

[9.1] *The nonvolatile memory controller of claim 8, wherein the interrupt manager controller is configured to map at least two completion queue states of the plurality of completion queue states to the interrupt vector state.*

Borchers explains that when there are multiple command/response queue pairs as shown in Fig. 1B, each buffer pair is processed and handled separately. “The command buffers 219a-219n and the response buffers 223a-223n may be configured to operate and function as described above with respect to the command buffer 119 and the response buffer 123 of FIG. 1A.” Ex.1005, [0049]. “The multiple different head and tail pointers, each of which corresponds to a buffer pair, may be stored on the host 106, the data storage device 100 or a combination of the host 106 and the data storage device 100.” Ex.1005, [0049]. “In some instances, an interrupt may be received by an interrupt handler 225a-225n when a related group of commands has been processed by the data storage device, as discussed in more detail below with respect to FIG. 2.” Ex.1005, [0050]. “Each of the buffer pairs may be granted access to any address mapping, which may be stored on the host 106 and/or on the data storage device 100.” Ex.1005, [0051].

Accordingly, each buffer pair is handled separately in the manner described

with Fig. 2. Fig. 2 describes the response queue status information (“*completion queue state*”) and interrupt mechanisms (“*interrupt vector state*”) as shown below. Thus, for each separate command/response buffer pair, the corresponding response queue status information is associated with the corresponding interrupt mechanisms for that pair (“*map at least two completion queue states of the plurality of completion queue states to the interrupt vector state*”).

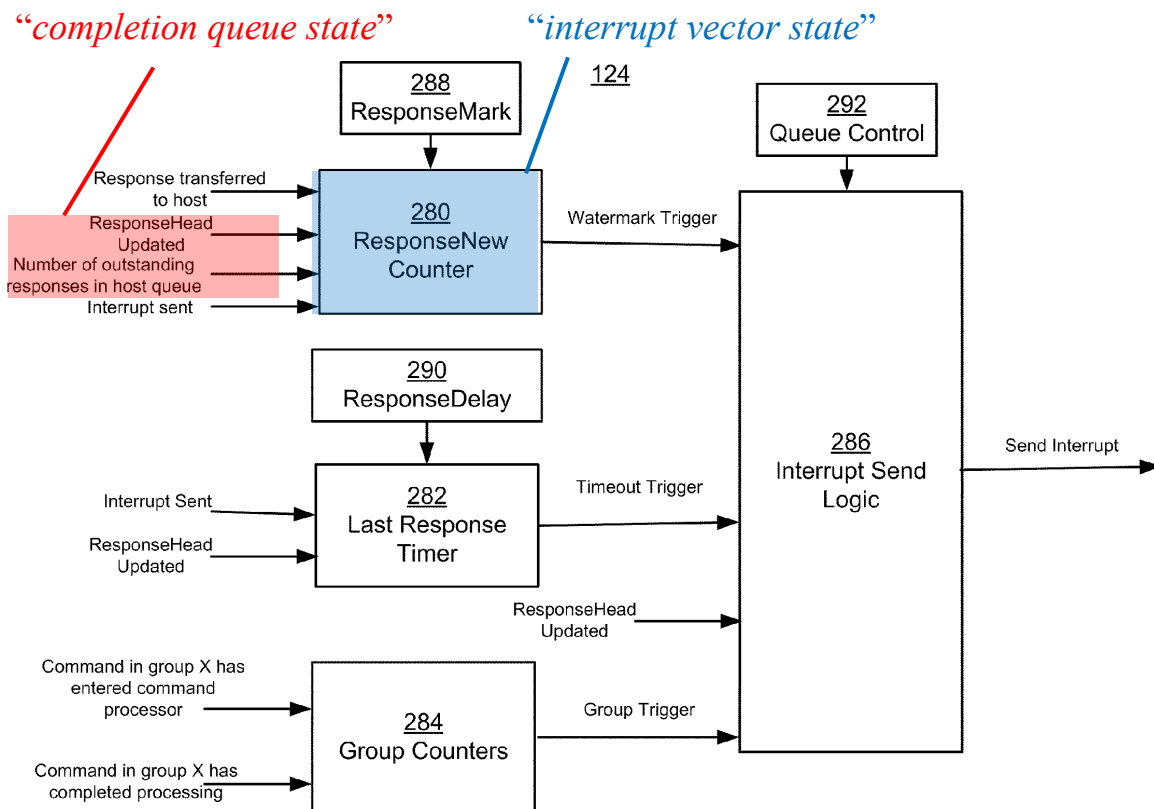


Ex.1005, Fig. 2 (annotated); Ex.1003, ¶175.

Thus, because Borchers’ interrupt controller includes interrupt mechanisms for each separate command/response buffer pair, Borchers renders this limitation

obvious. Ex.1003, ¶¶174-76.

Additionally, Borchers teaches a “*plurality of completion queue states*” in that there are several pieces of status information, including that a Response has been transferred to the host, the response queue head pointer has been updated, and the number of outstanding responses in the host queue. At least two of these pieces of information is mapped to one of the interrupt mechanisms. For example, the ResponseHead updated piece of information and number of outstanding response are mapped to the ResponseNew Counter 280 as shown below.



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶177.

Thus, because Borchers maps at least one of the response queue status items (ResponseHead Updated) to one of the interrupt mechanisms (ResponseNew Counter 280), Borchers renders this limitation obvious. Ex.1003, ¶¶177-78.

12. Claim 10

[10.1] *The nonvolatile memory controller of claim 8, wherein the interrupt manager controller is a sequential logic circuit.*

The specification of the '968 patent describes a sequential logic circuit as a state machine: “In various embodiments, the interrupt manager 1040 is a hardware mechanism for generating interrupt message packets. For example, the interrupt manager controller 1100 in the interrupt manager 1040 may be a sequential logic circuit (e.g., a state machine).” ’968 patent, 17:55-59. A POSITA would have understood a state machine to be any discrete computing system such as a digital computer. *See e.g.*, Ex.1010, 1102 (“**state machine** A model of a system in which all values are discrete, as in a digital computer.”). Borchers explains that the memory controller, of which the interrupt processor is a part, may be implemented in a variety of manners, all of which are some form of a state machine or “*sequential logic circuit.*” “The controller 110 may include a microcontroller, a FPGA controller, other types of controllers, or combinations of these controllers.” Ex.1005, [0113]. “In another exemplary implementation, the controller 110 is a FPGA controller. The FPGA controller may be implemented in hardware,

software, or a combination of hardware and software.” Ex.1005, [0114]. “Method steps also may be performed by, and an apparatus may be implemented as, special purpose logic circuitry, e.g., a FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit).” Ex.1005, [0127].

Thus, because Borchers’ memory controller, and its associated interrupt processor may be implemented in a variety of manners, including FPGAs or special logic circuitry, Borchers renders this limitation obvious. Ex.1003, ¶¶179-80.

13. Claim 11

[11.1] The nonvolatile memory controller of claim 8, wherein a completion queue state of the plurality of completion queue states includes a doorbell update status indicating whether the host processing unit has performed a doorbell update event in which the host processing unit updates a head pointer stored in the nonvolatile memory controller for the completion queue.

For the reasons explained above at [1.5] and [8.1], Borchers renders this limitation obvious. Ex.1003, ¶181.

14. Claim 12

[12.1] The nonvolatile memory controller of claim 8, wherein a completion queue state of the plurality of completion queue states includes an event count indicating a number of completion message packets received by a host controller interface and including a completion status for the completion queue associated with the completion queue state.

For the reasons explained above at [4.1], Borchers renders obvious “*wherein a completion queue state of the plurality of completion queue states includes an*

event count indicating a number of completion message packets received by a host controller interface.” Ex.1003, ¶182.

In addition, for the reasons explained above at [1.1] and [1.3], Borchers teaches “*a completion queue state of the plurality of completion queue states ... including a completion status for the completion queue associated with the completion queue state.*” Ex.1003, ¶183.

15. Claim 13

[13.1] *The nonvolatile memory controller of claim 12, wherein the interrupt vector state includes an aggregate event count for aggregating event counts of the completion queue state over a plurality of update operations, and wherein the interrupt manager is inhibited from generating an interrupt message packet if the aggregate event count is below an aggregate event count threshold.*

See [5.1]. Ex.1003, ¶184.

16. Claim 14

[14.1] *The nonvolatile memory controller of claim 8, wherein the interrupt vector state includes an aggregate time indicator for indicating a time period between the last time the interrupt manager generated an interrupt message packet for the completion queues associated with the interrupt vector state, and wherein the interrupt manager is inhibited from generating an interrupt message packet if the aggregate time indicator is below an aggregate time threshold.*

First, for the reasons explained above at [6.1], Borchers renders obvious “*wherein the interrupt vector state ... associated with the interrupt vector state*” as claimed.

Second, Borchers’ interrupt processor 124 does not send an interrupt signal

until the last response timer (“*aggregate time indicator*”) reaches a time threshold (“*wherein the interrupt manager is inhibited from generating an interrupt message packet if the aggregate time indicator is below an aggregate time threshold*”). “In one exemplary implementation, the host 106, through its driver 107, may control when the interrupt processor 124 should generate interrupts.” Ex.1005, [0036]. “For instance, the host 106 through the driver 107 may configure the interrupt processor 124 to use a water mark interrupt mechanism, a timeout interrupt mechanism, a group interrupt mechanism, or a combination of these interrupt mechanisms.” Ex.1005, [0036]. “If the time (e.g., time in microseconds) since the last interrupt is equal to or greater than the ResponseDelay and there are new responses in the response buffer 123, then the interrupt processor 124 generates an interrupt and the timeout is zeroed.” Ex.1005, [0037].

Thus, because Borchers uses a last response timer and does not send an interrupt signal until the timer reaches a Response Delay value, Borchers renders this limitation obvious. Ex.1003, ¶¶185-87.

17. Claim 15

[15.1] *The nonvolatile memory controller of claim 8, wherein a completion queue state of the plurality of completion queue states includes an empty queue indicator for indicating whether the completion queue is empty, and wherein the interrupt manager is inhibited from generating an interrupt message packet if the empty queue indicator indicates the completion queue is empty.*

See [7.1]. Ex.1003, ¶188.

18. Claim 16

[16.0] *A method comprising:*

Borchers describes a method for using circular command queues for communication between a host and memory controller. “In another general aspect, a method for communicating commands between a host and a flash memory data storage device includes populating a circular command queue of a driver on the host with commands for retrieval by the data storage device.” Ex.1005, [0006].

Accordingly, Borchers teaches “*a method*” as claimed. Ex.1003, ¶¶189-90.

[16.1] *generating a completion queue state by a nonvolatile memory controller for indicating the occurrence of a completion queue event associated with a completion queue in a host processing unit,*

See [1.1]. Ex.1003, ¶191.

[16.2] *the completion queue state including a doorbell update status indicating whether the host processing unit has performed a doorbell update event in which the host processing unit updates a head pointer stored in the nonvolatile memory controller for the completion queue;*

See [1.5]. Ex.1003, ¶192.

[16.3] *generating an interrupt vector state by the nonvolatile memory controller based on the completion queue state, the interrupt vector state associated with an interrupt vector in the host processing unit;*

First, for the reasons explained above at [1.2], Borchers renders obvious “*generating ... queue state.*”

Second, as explained above at [1.2], it would have been obvious for Borchers’ interrupt signals to include interrupt vectors (i.e., a message address and

message data). Accordingly, Borchers renders obvious “*the interrupt vector state associated with an interrupt vector in the host processing unit*” as claimed.

Ex.1003, ¶¶193-94.

[16.4] *determining by the nonvolatile memory controller based on the interrupt vector state that the completion queue contains an unprocessed completion status; and*

See [1.3]. Ex.1003, ¶195.

[16.5] *generating an interrupt message packet by a nonvolatile memory interface for triggering an interrupt in the host processing unit to alert the host processing unit of the unprocessed completion status in the completion queue.*

See [1.4]. Ex.1003, ¶196.

19. Claim 17

[17.1] *The method of claim 16, further comprising sending the interrupt message packet to the host processing unit.*

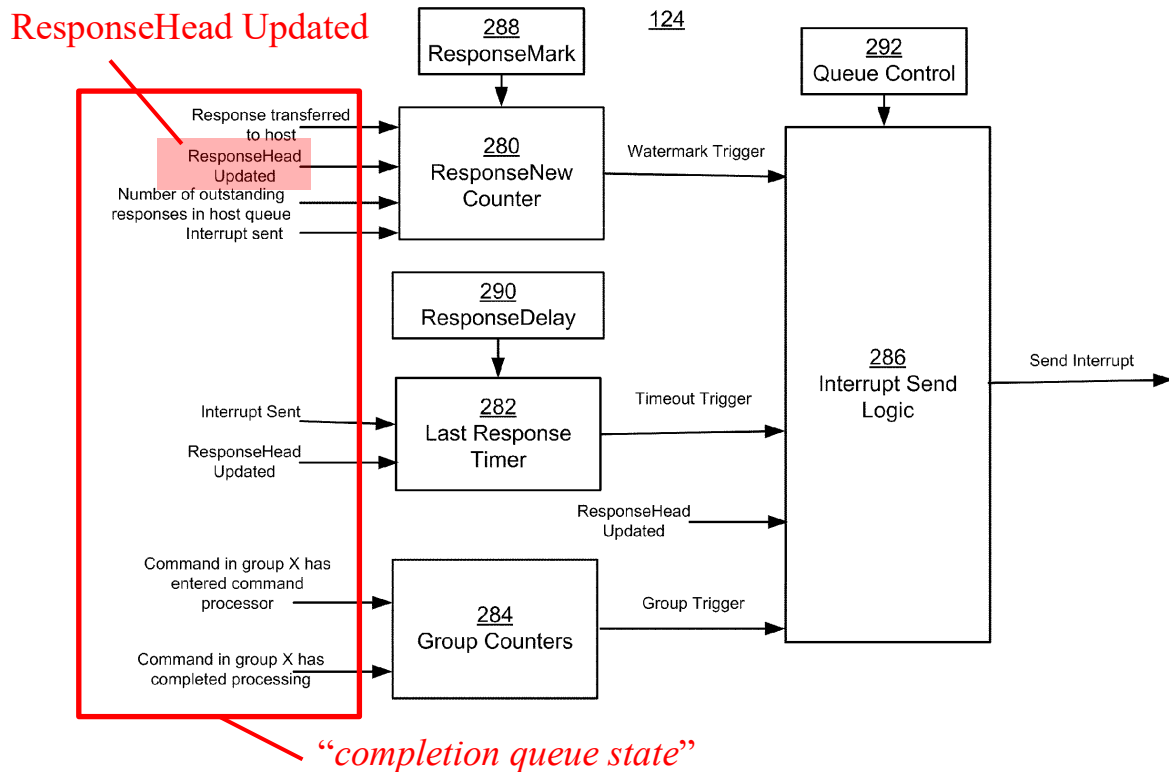
See [1.4]. Ex.1003, ¶197.

20. Claim 18

[18.1] *The method of claim 16, wherein the completion queue event is the host processing unit updating a head pointer stored in the nonvolatile memory controller for the completion queue.*

As explained above at [1.1], Borchers receives status information for the response queue (“*completion queue*”). This status information includes that the host has updated the head pointer of the response queue. During operation, “[t]he host 106 is configured to read the responses from the response buffer 123 and update the head pointer in the MMIO space of the data storage device to indicate

that the host has received the response.” Ex.1005, [0035].



Borchers, Fig. 2 (annotated); Ex.1003, ¶198.

Accordingly, because the response queue status information (“*completion queue state*”) includes an indication that the head pointer of the response queue has been updated, Borchers renders this limitation obvious. Ex.1003, ¶198-99.

D. Ground 2: Claims 1-18 are obvious in view of Borchers and the PCI standards.

Ground 1 shows how Borchers renders obvious each of the challenged claim limitations. Ground 1 further relies on the PCIe/PCI standards as evidence of the background knowledge of a POSITA. To the extent Patent Owner argues that the

PCIe/PCI standards would not have been within the background knowledge of a POSITA, Ground 2 is presented to show that combining the teachings of the PCIe/PCI standards with Borchers would have been obvious to a POSITA. Ex.1003, ¶200.

1. Summary of the PCI Standards

The PCI Express 2.0 Base Specification document “describes the PCI Express architecture, interconnect attributes, fabric management, and the programming interface required to design and build systems and peripherals that are compliant with the PCI Express specification.” Ex.1007, 21. “The PCIe Standard states that “[t]he goal is to enable such devices from different vendors to inter-operate in an open architecture.” Ex.1007, 21.

As an example, the PCIe Standard describes the structure of interrupt signals. PCIe describes a “Message Signaled Interrupt (MSI/MSI-X)” which are “Two similar but separate mechanisms that enable a device to request service by writing a system-specified DWORD of data to a system-specified address using a Memory Write Request. Compared to MSI, MSI-X supports a larger maximum number of vectors and independent message address and data for each vector.” Ex.1007, 26. *See also* Ex.1007, 458; Ex.1003, ¶¶201-06.

2. Reasons to combine the PCI standard with Borchers

A POSITA would have found it obvious to implement Borchers’ techniques

according to standardized practices such as those defined in the PCIe/PCI standards. Ex.1003, ¶207.

Borchers describes a known interface device for communication between a host and memory controller. *See generally*, Ex.1005. Borchers explains that “the high speed interface may include a PCIe interface.” Ex.1005, [0110]. “The PCIe interface 108 may include a connector to the host 106 such as, for example, a PCIe connector cable assembly. Other high speed interfaces, connectors and connector assemblies also may be used.” Ex.1005, [0110].

Accordingly, a POSITA would have looked to known techniques for implementing a PCI Express interface. It would have been obvious to look to the known standards documentation such as the PCI Express Base Standard and the PCI standard. Indeed, the PCI Express standard states that “describes the PCI Express architecture, interconnect attributes, fabric management, and the programming interface required to design and build systems and peripherals that are compliant with the PCI Express specification.” Ex.1007, 21. Accordingly, it would have been natural for a POSITA to implement Borchers’ memory interface 108 in accordance with industry standards. Ex.1003, ¶209.

For example, one aspect of the memory interface is how it implements interrupt signals. Borchers explains that “The interrupt processor 124 is configured to signal the host 106 with an interrupt when new responses are available in the

response buffer 123.” Ex.1005, [0035]. A POSITA would have found it obvious to thus implement the interrupt signals according to the PCI Express standard. As stated in the PCI Express standard, “[a]ll PCI Express device functions that are capable of generating interrupts must implement MSI or MSI-X or both.” Ex.1007, 453. “MSI and the MSI Capability structures are defined in the PCI Local Bus Specification, Revision 3.0.” Ex.1007, 453. PCIe describes a “Message Signaled Interrupt (MSI/MSI-X)” which are “Two similar but separate mechanisms that enable a device to request service by writing a system-specified DWORD of data to a system-specified address using a Memory Write Request. Compared to MSI, MSI-X supports a larger maximum number of vectors and independent message address and data for each vector.” Ex.1007, 26. *See also* Ex.1007, 458. Ex.1003, ¶210.

The PCI Local Bus Specification also explains that MSI and MSI-x interrupts use vectors. “System software initializes the message address and message data (from here on referred to as the ‘vector’) during device configuration, allocating one or more vectors to each MSI capable function.” Ex.1006, 231. Accordingly, it would have been obvious that Borchers’ interrupt signals, implemented according to the PCI Express standard, would have used interrupt vectors. Ex.1003, ¶211. Indeed, the combination is nothing more than “the predictable use of prior art elements according to their established functions.”

KSR Int'l v. Teleflex Inc., 550 U.S. 398, 417 (2007).

Accordingly, the combination of Borchers and the PCIe/PCI standards would have predictably resulted in interrupt signals implemented using interrupt vectors. A POSITA would have had a reasonable expectation of success because the PCIe/PCI standards are tried-and-true, industry vetted implementations. The PCIe standard is “an enhancement to the PCI architecture” which was “the most widely accepted and implemented expansion standard in the world.” Ex.1007, 21; Ex.1006, 15. Ex.1003, ¶212.

Thus, the combination of Borchers and the PCIe/PCI standards are merely applying a known technique (interrupt signals according to PCIe/PCI standards) to a known device (Borchers’ memory interface) ready for improvement to yield predictable results (interrupt signals with interrupt vectors). Ex.1003, ¶¶207-13.

3. Claim Analysis for claims 1-18

The claim analysis for Ground 2 is substantially the same as that of Ground 1. *See supra* X.C; Ex.1003, ¶214.

E. Ground 3: Claims 1-18 are obvious in view of Borchers, the PCIe/PCI standards, and Serebrin.

Grounds 1 and 2 rely on the background knowledge of a POSITA that the timer values, counter values, and other data that represent response queue status information (“*completion queue state*”) and interrupt mechanism information

(“*interrupt vector state*”) would be stored in some type of storage. It would have been obvious for this storage to be a register, flag, or other form of memory. To the extent Patent Owner argues that such is not background knowledge of a POSITA, Ground 3 shows that it would be obvious for an interrupt processor to include storage such as registers to temporarily store such status information, as evidenced by Serebrin. Ex.1003, ¶215.

1. Summary of Serebrin

Like Borchers and the '968 patent, Serebrin relates to “a device interrupt manager for a “memory management unit.” Ex.1008, [0008]. Serebrin provides evidence that it was known for interrupt managers to use cache or registers to store state information. Ex.1003, ¶¶215-19

2. Serebrin is Entitled to its Provisional Filing Date.

Serebrin’s provisional application (Ex.1009) is nearly identical to its specification and thus provides written description support for at least one claim as demonstrated in the chart below.

Serebrin Publication, Claim 1	Serebrin Provisional.
A device interrupt manager configured to receive an interrupt from a device that is assigned to a guest,	“A device interrupt manager configured to receive an interrupt from a device that is assigned to a guest.” Ex.1009, 48.

wherein the device interrupt manager is configured to transmit an operation for recordal of the interrupt in a memory location,	“The device manager may be configured to record the interrupt in a guest interrupt controller data structure.” Ex.1009, [0023], 10.
wherein the memory location is associated with a virtual processor within the guest,	More particularly, the device interrupt manager may be configured to detect which vCPU within the guest is targeted by the interrupt. The device manager may be configured to record the interrupt in a guest interrupt controller data structure. Ex.1009, [0023], 10.
wherein the virtual processor is targeted by the interrupt.	A processor or vCPU may be targeted by an interrupt if the interrupt is to be received by the interrupt controller associated with the processor/vCPU. Ex.1009, [0027], 12.

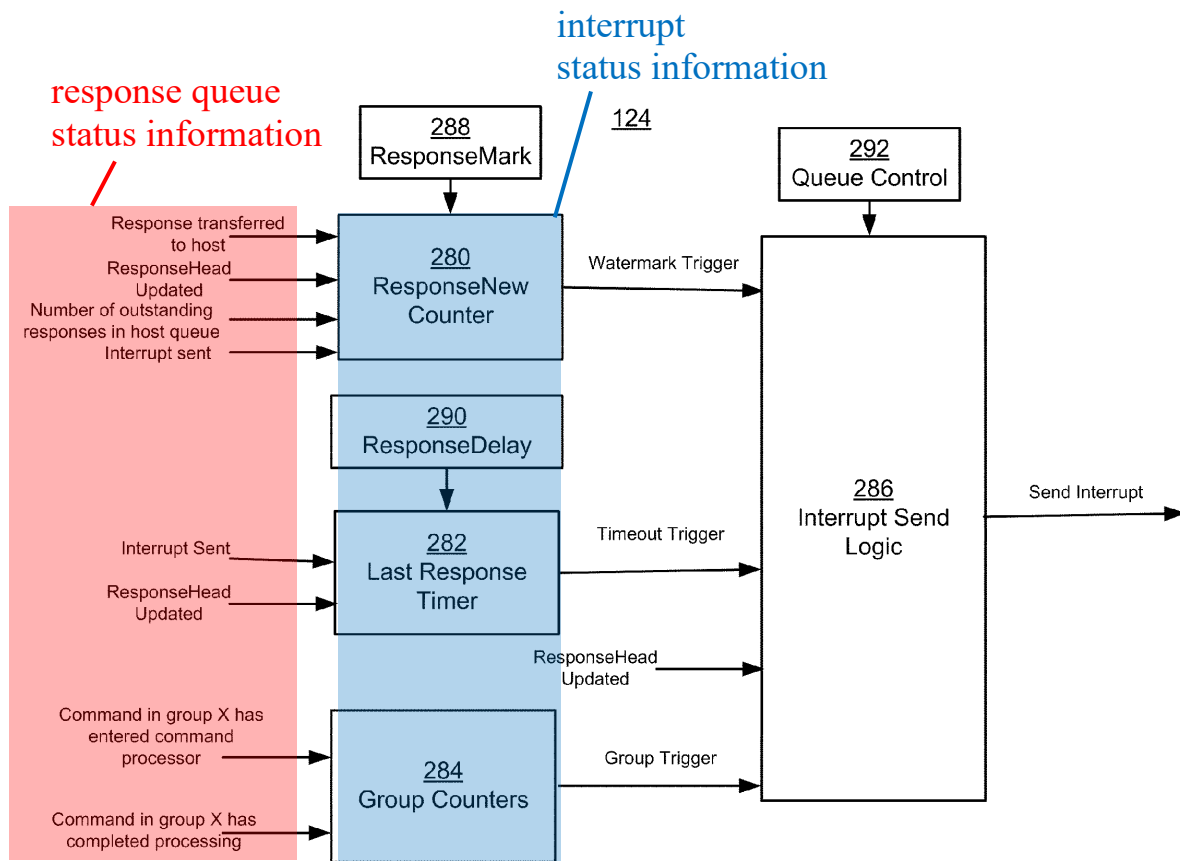
Accordingly, because the Serebrin Provisional provides written description support for each element of the claims, it is entitled to the filing date of its provisional application. The following analysis cites to both Serebrin and the corresponding teaching in the Serebrin provisional. Ex.1003, ¶¶220-22.

3. Reasons to combine Serebrin with Borchers and the PCIe/PCI standards

A POSITA would have found it obvious for Borchers' interrupt processor to utilize cache or registers to store state information, as evidenced by Serebrin.

Ex.1003, ¶223.

As described above at X.C.1, Borchers' memory controller includes an interrupt processor 124. The interrupt processor 124 relies on status information from the host's response queue. The status information is then used by the interrupt processor 124 to maintain interrupt status information (e.g., timers and counters) for determining when to send an interrupt signal to the host. *See* Ex.1005, [0060].



Ex.1005, Fig. 2 (annotated); Ex.1003, ¶224.

A POSITA would have understood that both the response queue status information and the interrupt status information would be stored or buffered in some type of storage location, such as a cache or register. Ex.1003, ¶225. Indeed, it was known for interrupt managers to use caches or registers to store state information. Serebrin provides an example of using caches or registers to store state information. Indeed, the combination is nothing more than “the predictable use of prior art elements according to their established functions.” *KSR* 550 U.S. at 417 (2007).

Serebrin describes “a memory controller 42 and an input/output (I/O) memory management unit (IOMMU) 40 that further includes a device interrupt manager 38.” Ex.1008, [0051]; Ex.1009, 19. The interrupt manager is connected to memory 56 as shown in Fig. 2 below. The memory 56 stores gAPIC (guest Advanced Programmable Interrupt Controller) states. The “device interrupt manager 38 may be configured to record each guest interrupt received in the bridge 36 in the gAPIC state data structure 58 [and] update the gAPIC state in the gAPIC state data structure 58.” Ex.1008, [0055]; Ex.1009, 21. Serebrin explains that a variety of forms of memory such as cache or registers may be used to store the interrupt state information. *See e.g.*, Ex.1008, [0057]; Ex.1009, 22.

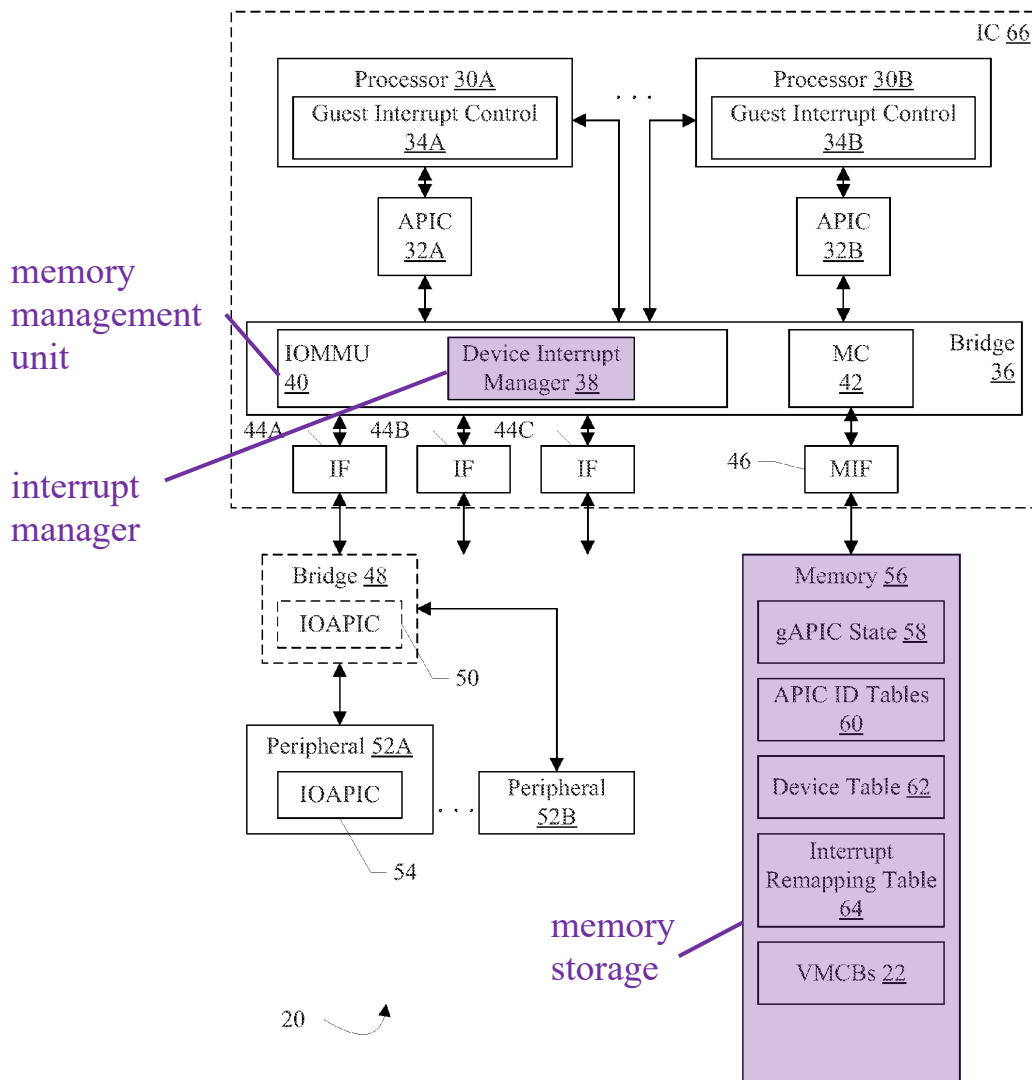


Fig. 2

Ex.1008, Fig. 2 (annotated); Ex.1003, ¶226.

Accordingly, a POSITA would have found it obvious for Borchers' interrupt manager to similarly include cache/registers for storing state data. Ex.1003, ¶227. The combination of Serebrin and Borchers thus predictably results in Borchers' response queue status information and interrupt mechanism information being placed in storage such as cache or registers.

A POSITA would have had a reasonable expectation of success given that Serebrin stores similar types of state information as Borchers. Ex.1003, ¶228. In particular, Serebrin’s interrupt manager stores state information such as “the interrupt requests that have been transmitted to the gAPIC” and “internally-generated interrupts, timers, the local vector table, etc.” Ex.1008, [0090]-[0094]; Ex.1009, 34-35.

Accordingly the combination of Serebrin with the teachings of Borchers is merely applying a known technique (Serebrin’s cache/registers) to a known device (Borchers’ interrupt manager) ready for improvement to yield predictable results (storing state information in cache/registers). Ex.1003, ¶¶223-29.

4. Claim Analysis for claims 1-18

The claim analysis for Ground 3 is substantially the same as that of Ground 1. *See supra* X.C. Ex.1003, ¶230.

XI. DISCRETIONARY DENIAL IS NOT WARRANTED

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

A. The Fintiv factors favor institution.

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

1. No evidence regarding a stay

No motion to stay has been filed, so the Board should not infer the outcome of such a motion. Thus, this factor is neutral on discretionary denial.

2. Parallel proceeding trial date

This factor weighs against discretionary denial because no trial date has currently been scheduled. Moreover, the projected trial date—based on median time-to-trial statistics—is in January of 2027³, which is well after the projected date for the Board to issue a Final Written Decision in July 2026. Because the projected trial date is approximately six months *after* the Board’s expected date for issuing a Final Written Decision, this factor weighs against discretionary denial.

3. Investment in the parallel proceeding

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

4. Overlapping issues with the parallel proceeding

The co-pending litigation is in its early stages. Petitioner served its preliminary invalidity contentions on December 16, 2024. However, Petitioner has

³ The current average time-to-trial for the district court hearing the co-pending litigation is 30.4 months. *See* Ex.1011, Ex.1012.

not served its final invalidity contentions, nor has the deadline for doing so been set. The extent of overlap is thus speculative at this point in time.

5. Petitioner is a defendant

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

6. Other circumstances

“[T]he PTAB will not deny institution of an IPR or PGR under *Fintiv* (i) when a petition presents compelling evidence of unpatentability.” Ex.1013, 9. Here, the petition—along with Dr. Min’s expert testimony—plainly shows that the ’968 patent claims no more than known concepts related to interrupt coalescing.

B. Advanced Bionics Test Favors Institution.

Denial under 35 U.S.C. § 325(d) is not warranted because the challenges presented in this petition are neither cumulative nor redundant to the prosecution of the ’968 patent.

Under the first prong of the *Advanced Bionics* framework, Petitioner acknowledges that Borchers was cited in an IDS provided by the Applicant. *Advanced Bionics, LLC v. MED-EL Elektromedizinische Geräte GmbH*, IPR2019-01469, Paper 6 at 7-8 (Feb. 13, 2020) (precedential).

Under the second prong, however, the evidence shows that the Examiner “erred in a manner material to the patentability of challenged claims.” *Id.* The

second prong is guided by *Becton, Dickinson* factors (c), (e), and (f). Id. at 9-10, n. 10 (citing *Becton, Dickinson & Co. v. B. Braun Melsungen AG*, IPR2017-01586, Paper 8 at 17-18 (Dec. 15, 2017)). These factors weigh against exercising discretion.

With respect to factor (c), Borchers was never used in a rejection nor discussed at all by the Examiner. Borchers' inclusion in an IDS does not show that Borchers was given serious evaluation by the Examiner. *See Navistar, Inc. v. Fatigue Fracture Tech., LLC*, IPR2018-00853, Paper 13 at 17 (Sept. 12, 2018) ("...the fact that [references] were of record... provides little impetus... to deny institution under § 325(d)").

With respect to factors (e) and (f), the Examiner erred by overlooking Borchers' teaching with respect to interrupt coalescing. While Borchers does not refer to certain items as a "doorbell update status," a POSITA would have recognized that Borchers' response head update is a "doorbell update" status as evidenced by at least the background knowledge of a POSITA evidenced in the PCI standards. This Petition presents the PCI standards as additional evidence, along with Dr. Min's testimony, to show that Borchers teaches what is claimed. *See, e.g., Puma N. Am., Inc. v. Nike, Inc.*, IPR2019-01058, Paper 10, 19 (PTAB Oct. 31, 2019) (instituting where petition presented "new non-cumulative evidence.... helpful to our consideration of a prior art combination that was not

before the Examiner”). Accordingly, discretionary denial under 325(d) is not warranted.

XII. CONCLUSION

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

Respectfully submitted,

Dated: January 24, 2025
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, Texas 75201
Customer No. 27683

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

XIII. MANDATORY NOTICES

A. Real Party-in-Interest

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

B. Related Matters

Pursuant to 37 C.F.R. § 42.8(b)(2), to the best knowledge of the Petitioner, the ’968 patent is or was involved in the following cases:

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

C. Lead and Back-up Counsel and Service Information

Lead Counsel

David O’Dell
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, TX 75209

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

Back-up Counsel

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

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

Dallas, TX 75209

USPTO Reg. No. 66,910

Hong Shi
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, TX 75209

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

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

CERTIFICATE OF WORD COUNT

Pursuant to 37 C.F.R. § 42.24(d), Petitioner hereby certifies, in accordance with and reliance on the word count provided by the word-processing system used to prepare this Petition, that the number of words in this paper is 13,987. Pursuant to 37 C.F.R. § 42.24(d), this word count excludes the table of contents, table of authorities, mandatory notices under § 42.8, certificate of service, certificate of word count, appendix of exhibits, and any claim listing.

Dated: January 24, 2025

/David M. O'Dell/

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

CERTIFICATE OF SERVICE

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

Date of service January 24, 2025

Manner of service FEDERAL EXPRESS

Documents served Petition for *Inter Partes* Review Under 35 U.S.C. § 312 and 37 C.F.R. § 42.104 of U.S. 8,554,968; Petitioner's Power of Attorney; Petitioner's Exhibit List; All Exhibits.

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

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