



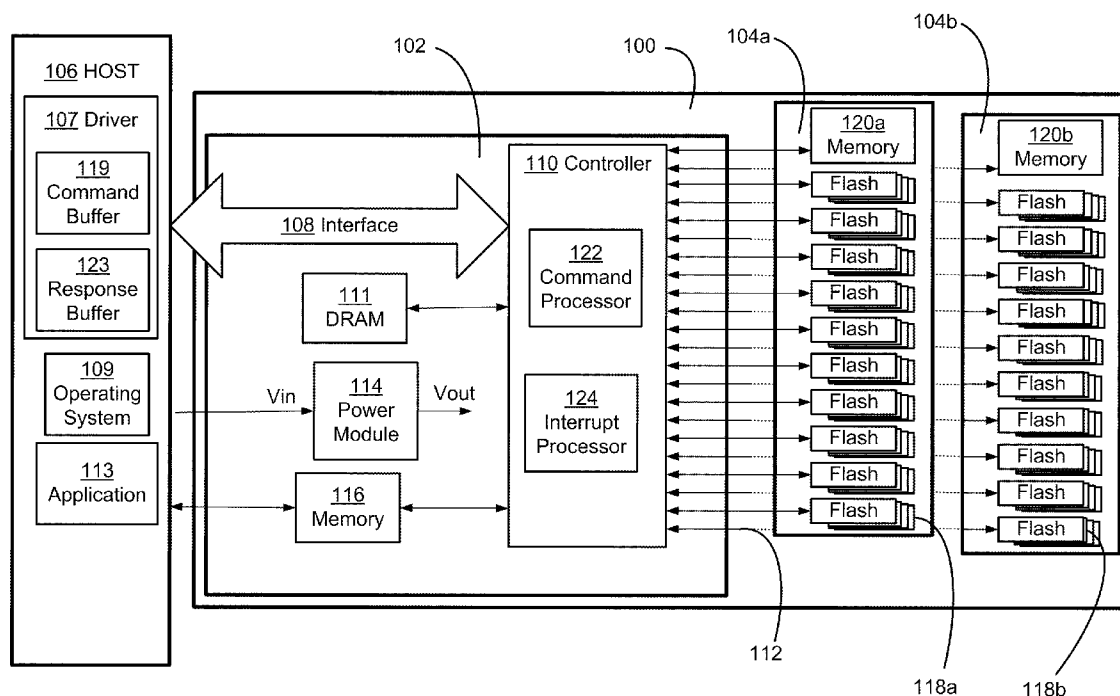
US 20100262979A1

(19) **United States**(12) **Patent Application Publication**
Borchers et al.(10) **Pub. No.: US 2010/0262979 A1**(43) **Pub. Date: Oct. 14, 2010**(54) **CIRCULAR COMMAND QUEUES FOR
COMMUNICATION BETWEEN A HOST AND
A DATA STORAGE DEVICE**(75) Inventors: **Albert T. Borchers**, Santa Cruz,
CA (US); **Andrew T. Swing**, Los
Gatos, CA (US); **Robert S.
Sprinkle**, San Jose, CA (US);
Grant Grundler, Mountain View,
CA (US); **Christopher L. Johnson**,
San Francisco, CA (US)

Correspondence Address:

BRAKE HUGHES BELLERMANN LLP
c/o CPA Global
PO Box 52050
Minneapolis, MN 55402 (US)(73) Assignee: **GOOGLE INC.**, Mountain View,
CA (US)(21) Appl. No.: **12/756,477**(22) Filed: **Apr. 8, 2010****Related U.S. Application Data**(63) Continuation-in-part of application No. 12/537,733,
filed on Aug. 7, 2009.(60) Provisional application No. 61/167,709, filed on Apr.
8, 2009, provisional application No. 61/187,835, filed
on Jun. 17, 2009, provisional application No. 61/304,
469, filed on Feb. 14, 2010, provisional application
No. 61/304,468, filed on Feb. 14, 2010, provisional
application No. 61/304,475, filed on Feb. 14, 2010.**Publication Classification**(51) **Int. Cl.**
G06F 9/44 (2006.01)
G06F 3/00 (2006.01)(52) **U.S. Cl.** **719/321; 710/5**(57) **ABSTRACT**

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, transferring commands from the circular command queue to the data storage device via a device initiated direct memory access operation, populating, via a direct memory access operation initiated by the data storage device, a circular response queue of the host with responses by the data storage device for retrieval by the host device, where each response acknowledges the reception of a command from the host by the data storage device, and consuming responses from the circular response queue at the host.



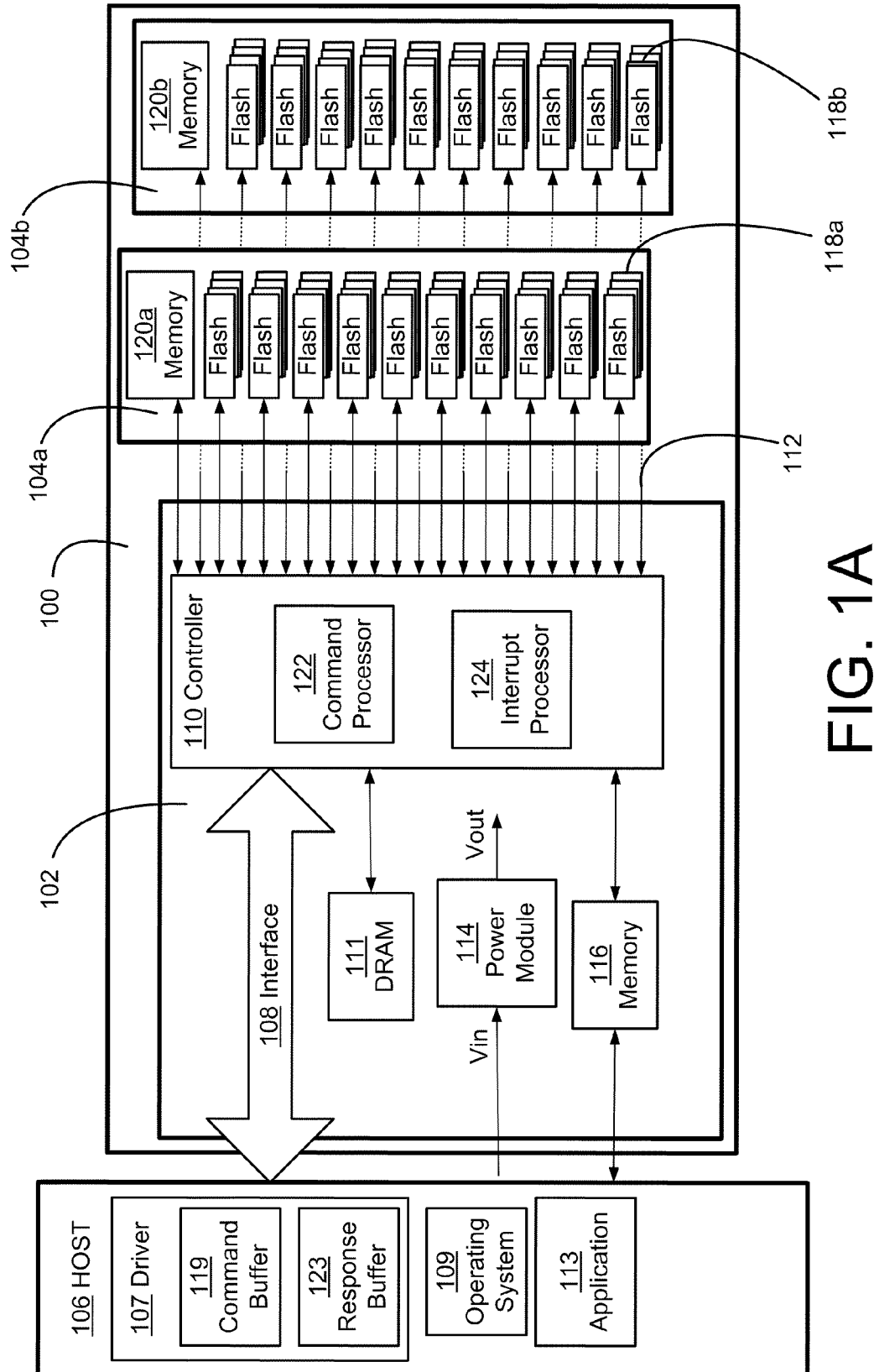


FIG. 1A

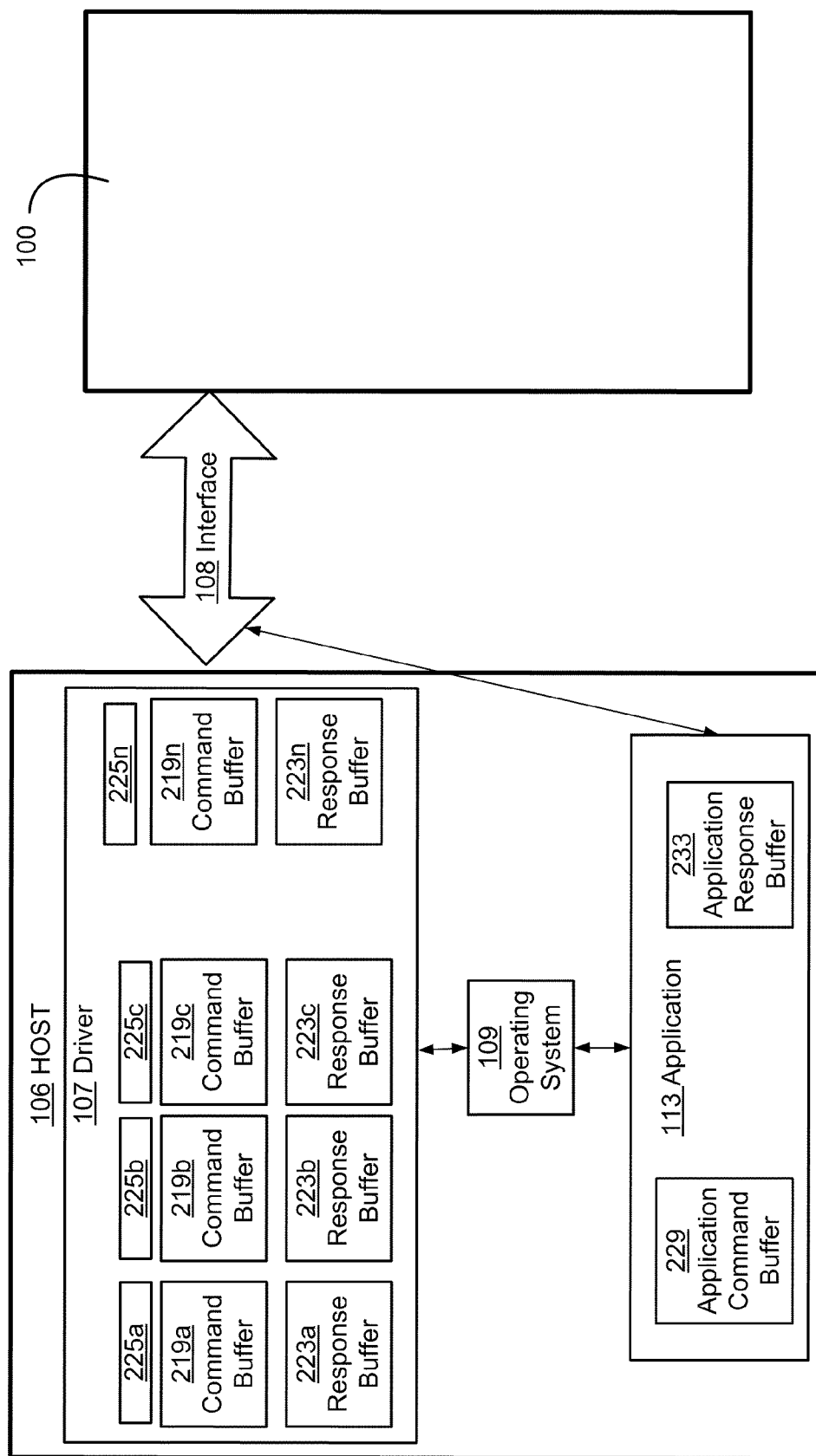
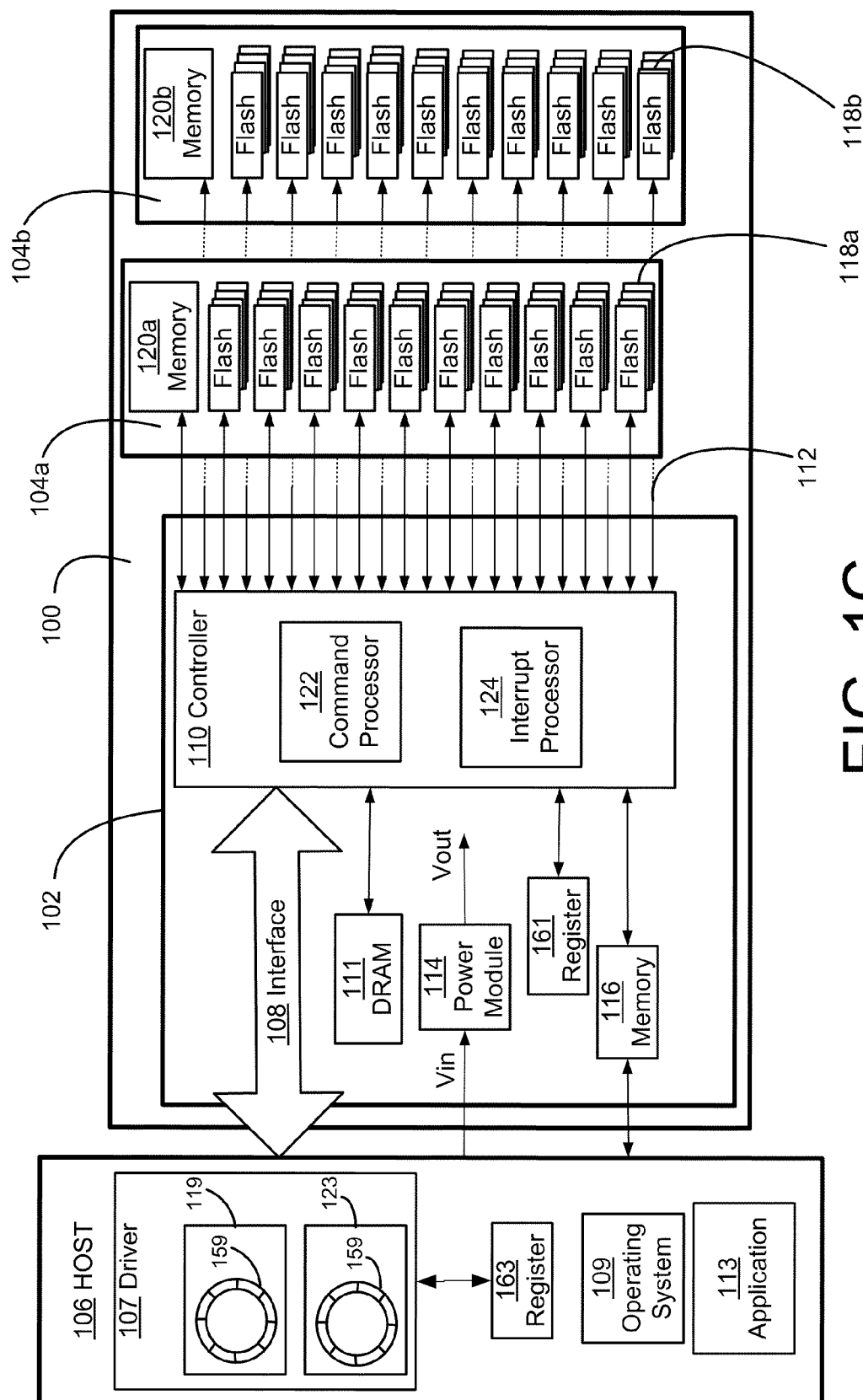


FIG. 1B



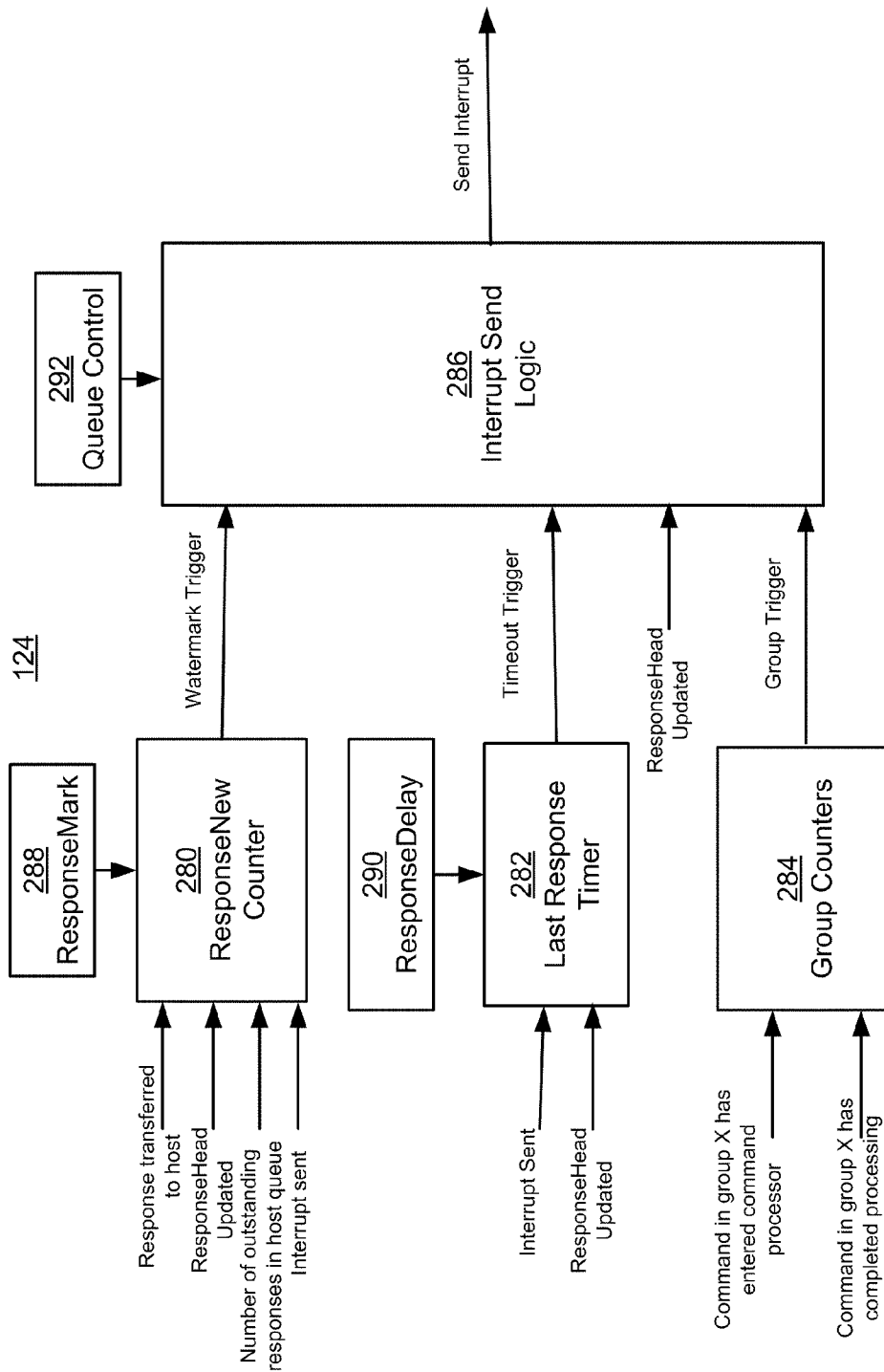


FIG. 2

122

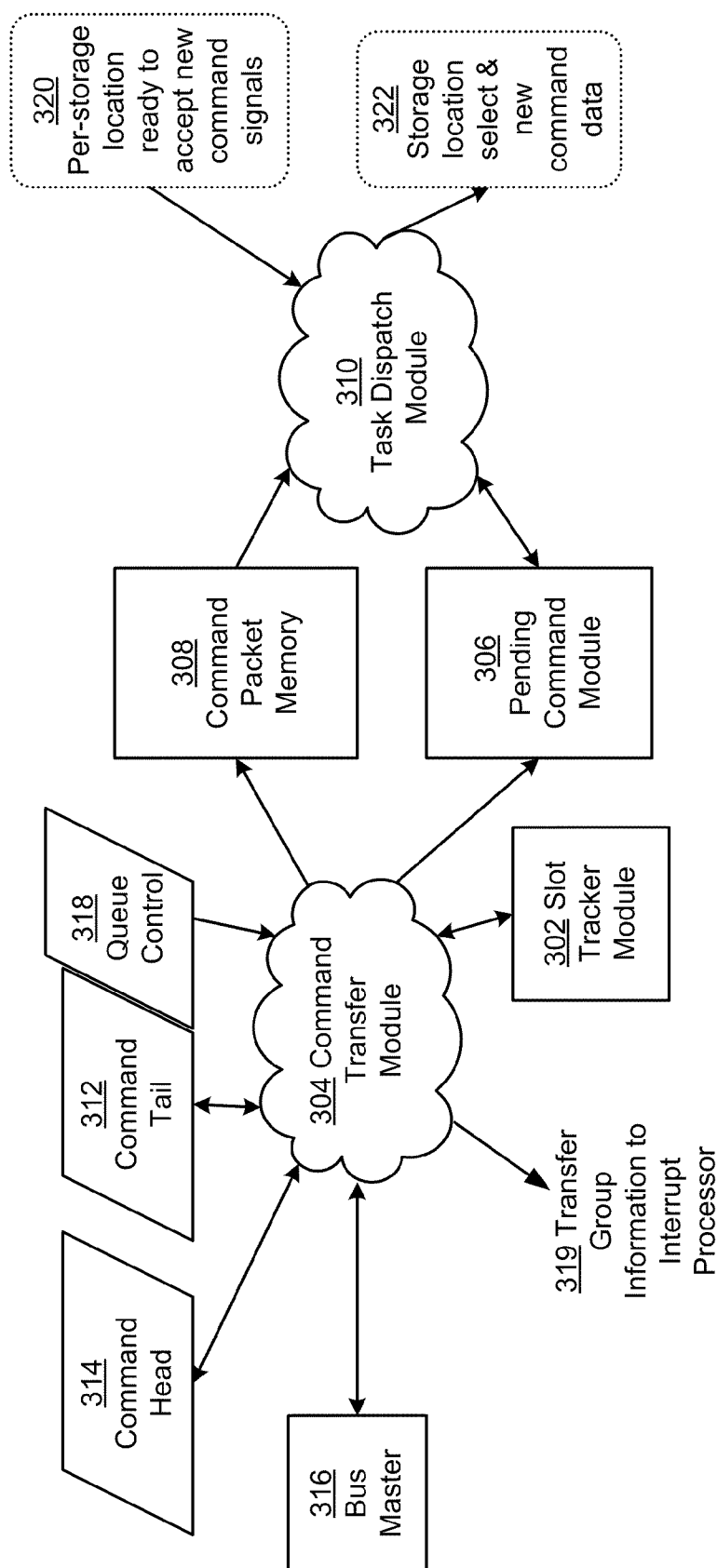


FIG. 3

306

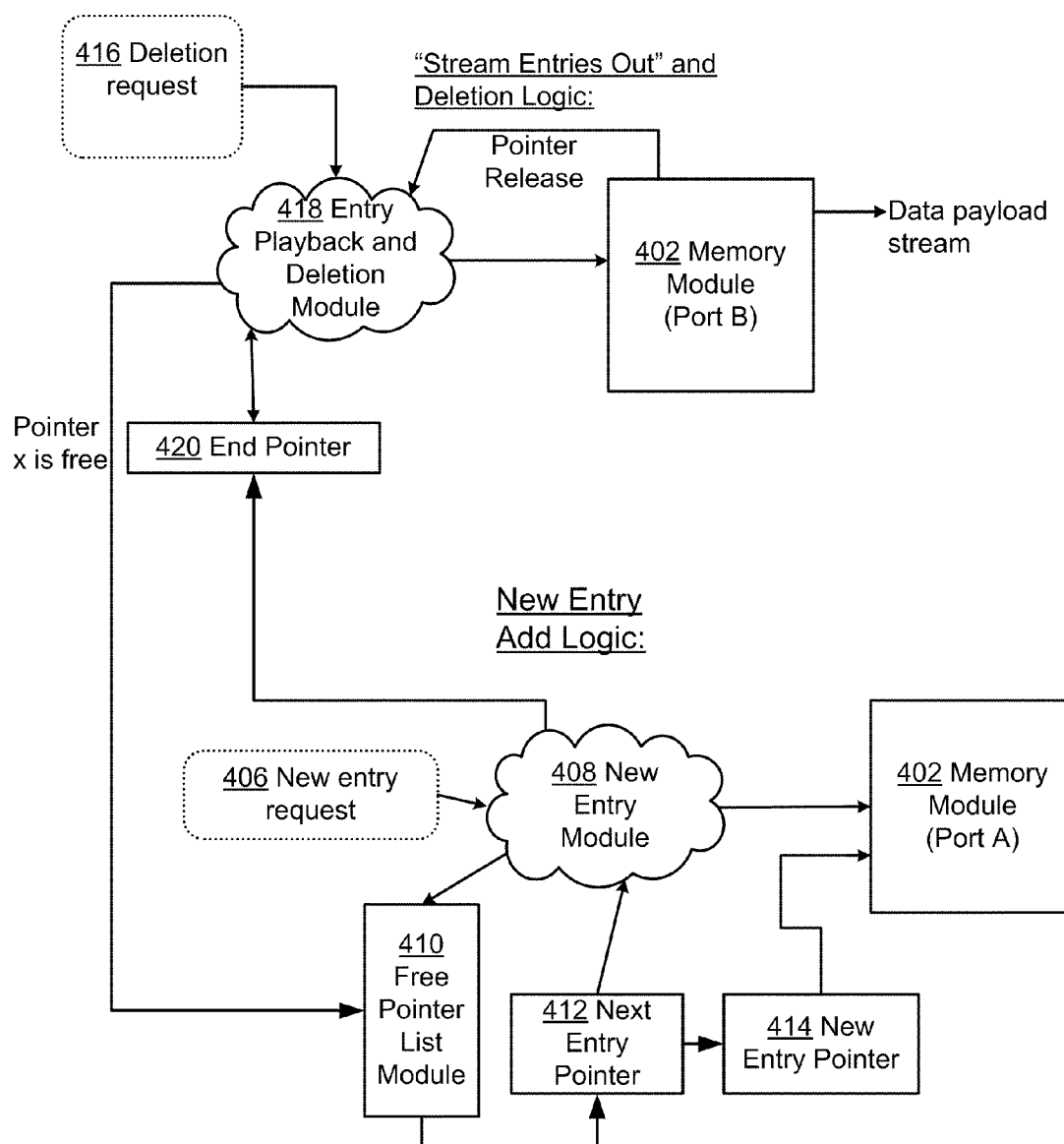


FIG. 4

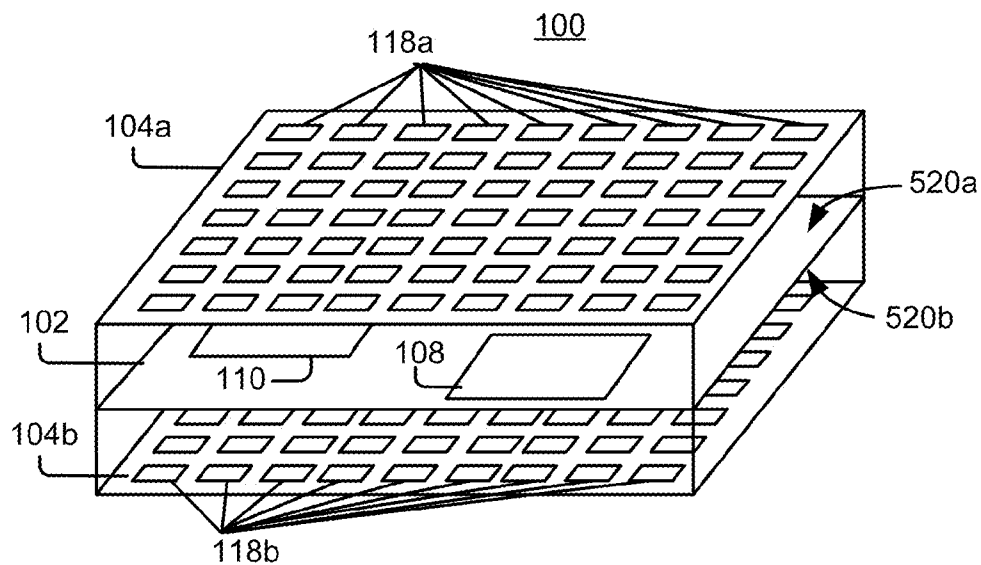


FIG. 5

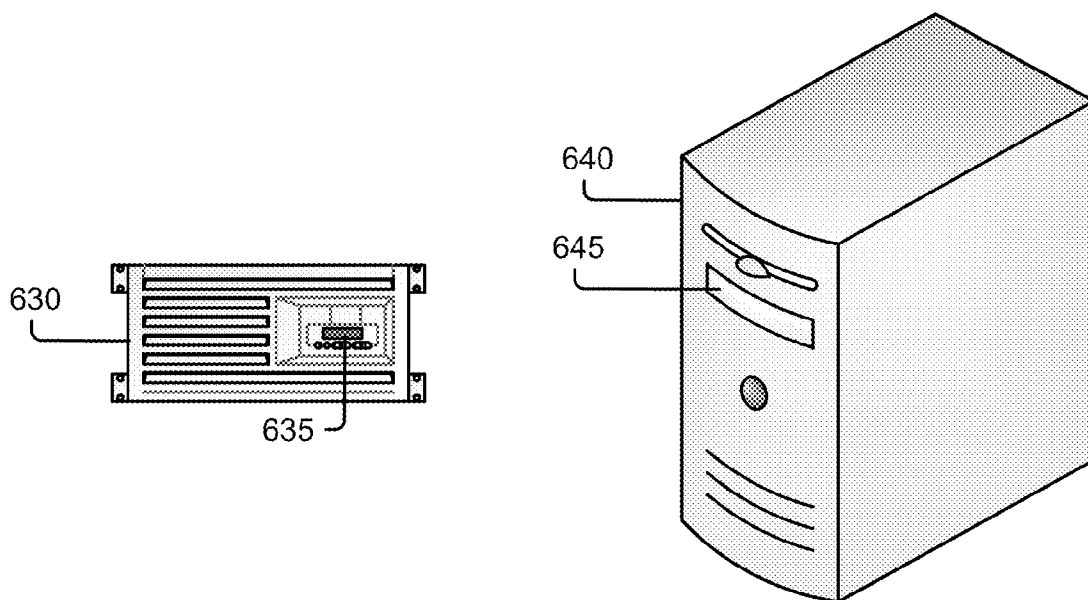


FIG. 6

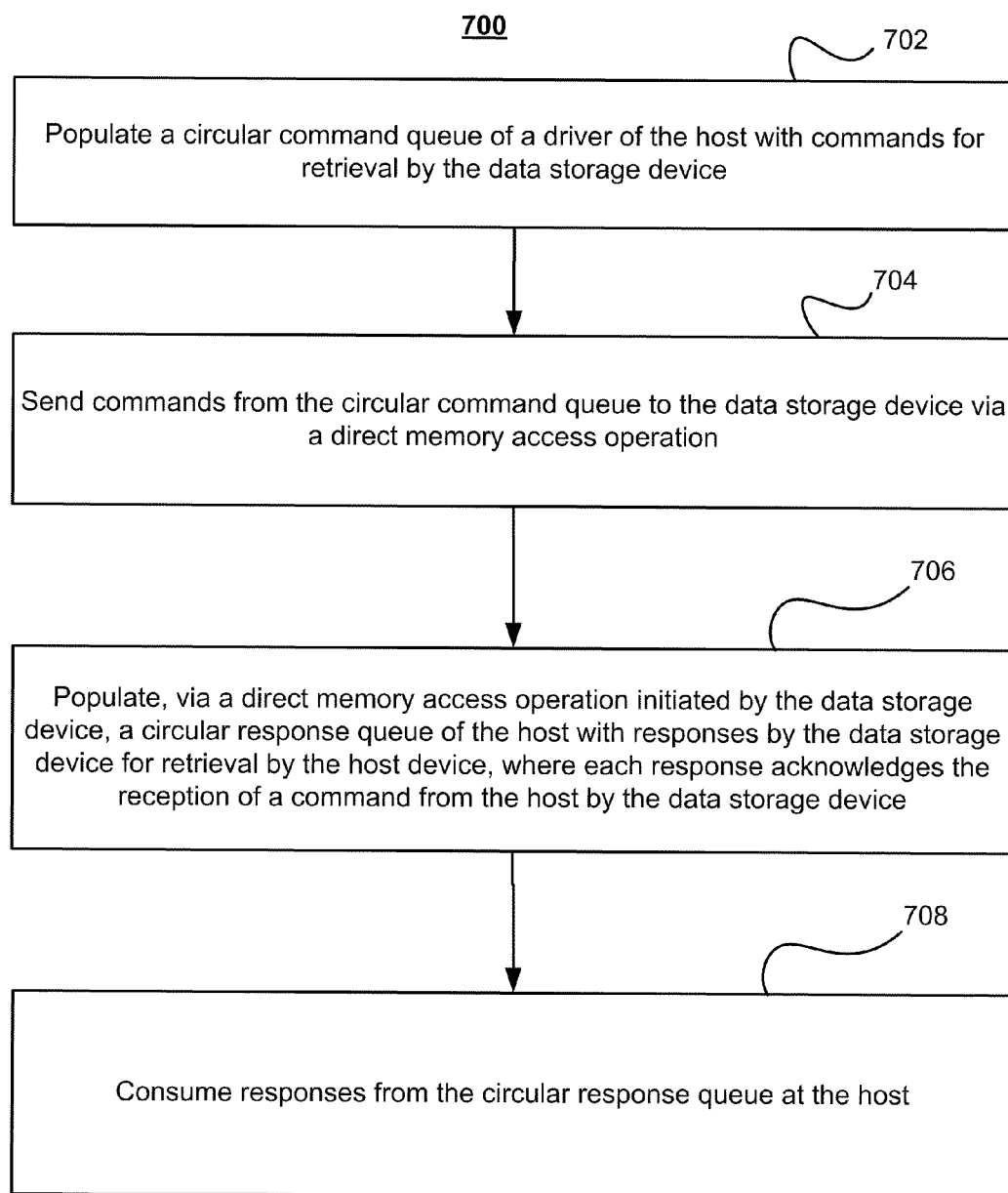


FIG. 7

CIRCULAR COMMAND QUEUES FOR COMMUNICATION BETWEEN A HOST AND A DATA STORAGE DEVICE

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation-in-part application of U.S. patent application Ser. No. 12/537,733, filed on Aug. 7, 2009, entitled “MULTIPLE COMMAND QUEUES HAVING SEPARATE INTERRUPTS,” which in turn, claims the benefit of U.S. Provisional Application No. 61/167,709, filed Apr. 8, 2009, and titled “DATA STORAGE DEVICE” and U.S. Provisional Application No. 61/187,835, filed Jun. 17, 2009, and titled “PARTITIONING AND STRIPING IN A FLASH MEMORY DATA STORAGE DEVICE.” This application claims the benefit of U.S. Provisional Application No. 61/304,469, filed Feb. 14, 2010, and titled “DATA STORAGE DEVICE,” U.S. Provisional Patent Application No. 61/304,468, filed Feb. 14, 2010, and titled “DATA STORAGE DEVICE,” and U.S. Provisional Patent Application No. 61/304,475, filed Feb. 14, 2010, and titled “DATA STORAGE DEVICE,” all of which are hereby incorporated by reference in entirety. Each of the above-referenced applications is hereby incorporated by reference in its entirety.

TECHNICAL FIELD

[0002] This description relates to data storage devices and, in particular, to circular command queues for communication between a host and a data storage device.

BACKGROUND

[0003] Data storage devices may be used to store data. A data storage device may be used with a computing device to provide for the data storage needs of the computing device. In certain instances, it may be desirable to store large amounts of data on a data storage device. Also, it may be desirable to execute commands quickly to read data and to write data to the data storage device.

SUMMARY

[0004] In a first general aspect, a host device configured for storing data on, and retrieving data from, a flash memory data storage device, includes a driver that is arranged and configured to communicate commands to the data storage device, a circular command queue that is populated with commands for retrieval by the data storage device, and a circular response queue that is populated with responses by the data storage device for retrieval by the host device, wherein each response acknowledges the reception of a command from the host by the data storage device.

[0005] Implementations can include one or more of the following features. For example, the circular command queue can include a command head pointer and a command tail pointer, and the circular response queue can include a response head pointer and a response tail pointer, and the host device can further include a first register configured to store command head pointer values, and a second register configured to store response tail pointer values. The data storage device can include a third register configured to store command tail pointer values, and a fourth register configured to store response head pointer values. The third register can exist in a memory mapped address space of the data storage device, and the driver can be configured to write updated command

tail pointer values to the third register. The driver can be configured to send commands to the storage device in response to a direct memory access request from the data storage device, and the first register can be configured to receive updated command head pointer values in response to a direct memory access operation received from the data storage device. The second register can exist in the address space of the host device, and the second register can be configured to receive updated response tail pointer values from the data storage device into the second register. The driver can be configured to receive responses from the storage device through a direct memory access operation sent from the data storage device, and the driver can be configured to send updated response head pointer values to the data storage device via a write to a Memory Mapped register. The host device can further include an application that is configured to generate input and output requests, and an operating system that is operably coupled to the driver and to the application and that is configured to communicate the input and output requests between the application and the driver.

[0006] 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, transferring commands from the circular command queue to the data storage device via a device initiated direct memory access operation, populating, via a direct memory access operation initiated by the data storage device, a circular response queue of the host with responses by the data storage device for retrieval by the host device, where each response acknowledges the reception of a command from the host by the data storage device, and consuming responses from the circular response queue at the host.

[0007] Implementations can include one or more of the following features. For example, the circular command queue can include a command head pointer and a command tail pointer, and the circular response queue can include a response head pointer and a response tail pointer, and the method can further include storing command head pointer values in a first register of the host, and storing response tail pointer values in a second register of the host. The data storage device can include a third register configured to store command tail pointer values, and a fourth register configured to store response head pointer values. The third register can exist in a memory mapped address space of the data storage device, and the method can further include writing updated command tail pointer values to the third register. Updated command head pointer values can be received into the first register in response to a direct memory access operation received from the data storage device. The second register can exist in the address space of the host device, and the method can further include receiving updated response tail pointer values into the second register from the data storage device. Responses from the storage device can be received through a direct memory access operation sent from the data storage device, and updated response head pointer values can be sent to the data storage device via a write to a Memory Mapped register. Input and output requests can be generated from an application running on the host, and the input and output requests can be communicated from an application running on the host through an operating system to the driver.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] FIG. 1A is an exemplary block diagram of a host and a data storage device.

[0009] FIG. 1B is an exemplary block diagram of multiple queues on the host of FIG. 1A.

[0010] FIG. 1C is an exemplary block diagram of circular queues used to communicate information between the host and the data storage device of FIG. 1A.

[0011] FIG. 2 is an exemplary block diagram of an interrupt processor.

[0012] FIG. 3 is an exemplary block diagram of a command processor for the data storage device.

[0013] FIG. 4 is an exemplary block diagram of a pending command module.

[0014] FIG. 5 is an exemplary perspective block diagram of the printed circuit boards of the data storage device.

[0015] FIG. 6 is an exemplary block diagram of exemplary computing devices for use with the data storage device of FIG. 1A.

[0016] FIG. 7 is an exemplary flowchart illustrating a process for communicating commands between a host and a data storage device.

DETAILED DESCRIPTION

[0017] This document describes an apparatus, system(s) and techniques for using one or more pairs of queues at a host to communicate commands and responses between the host and a data storage device. Each pair of queues includes a command queue and a response queue. The pairs of queues enable the host to communicate with the data storage device using multiple threads or cores in an efficient manner.

[0018] Referring to FIG. 1A, a block diagram of a system for processing and tracking commands in a group is illustrated. 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. The data storage device 100 may communicate with the host 106 over an interface 108. The interface 108 may be between the host 106 and the controller board 102.

[0019] The controller board 102 may include a controller 110, a DRAM 111, multiple channels 112, a power module 114, and a memory module 116. The controller 110 may include a command processor 122 and an interrupt processor 124, as well as other components, which are not shown. The memory boards 104a and 104b may include multiple flash memory chips 118a and 118b on each of the memory boards. The memory boards 104a and 104b also may include a memory device 120a and 120b, respectively.

[0020] The host 106 may include a driver 107, an operating system 109 and one or more applications 113. In general, the host 106 may generate commands to be executed on the data storage device 100. For example, the application 113 may be configured to generate commands for execution on the data storage device 100. 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. The operating system 109 may be operably coupled to the driver 107, where the driver 107 may act as an interface between the host 106 and the data storage device 100. In other exemplary implementations, the application 113 may communicate directly with the data storage device 100, as discussed below with respect to FIG. 1B.

[0021] In general, the data storage device 100 may be configured to store data on the flash memory chips 118a and 118b. The host 106 may write data to and read data from the

flash memory chips 118a and 118b, as well as cause other operations to be performed with respect to the flash memory chips 118a and 118b. The reading and writing of data between the host 106 and the flash memory chips 118a and 118b, as well as the other operations, may be processed through and controlled by the controller 110 on the controller board 102. The controller 110 may receive commands from the host 106 and cause those commands to be executed using the command processor 122 and the flash memory chips 118a and 118b on the memory boards 104a and 104b. The communication between the host 106 and the controller 110 may be through the interface 108. The controller 110 may communicate with the flash memory chips 118a and 118b using the channels 112.

[0022] The controller board 102 may include DRAM 111. The DRAM 111 may be operably coupled to the controller 110 and may be used to store information. For example, the DRAM 111 may be used to store logical address to physical address maps and bad block information. The DRAM 111 also may be configured to function as a buffer between the host 106 and the flash memory chips 118a and 118b.

[0023] In one exemplary implementation, the controller board 102 and each of the memory boards 104a and 104b are physically separate printed circuit boards (PCBs). The memory board 104a may be on one PCB that is operably connected to the controller board 102 PCB. For example, the memory board 104a may be physically and/or electrically connected to the controller board 102. Similarly, the memory board 104b may be a separate PCB from the memory board 104a and may be operably connected to the controller board 102 PCB. For example, the memory board 104b may be physically and/or electrically connected to the controller board 102. The memory boards 104a and 104b each may be separately disconnected and removable from the controller board 102. For example, the memory board 104a may be disconnected from the controller board 102 and replaced with another memory board (not shown), where the other memory board is operably connected to controller board 102. In this example, either or both of the memory boards 104a and 104b may be swapped out with other memory boards such that the other memory boards may operate with the same controller board 102 and controller 110.

[0024] In one exemplary implementation, the controller board 102 and each of the memory boards 104a and 104b may be physically connected in a disk drive form factor. The disk drive form factor may include different sizes such as, for example, a 3.5" disk drive form factor and a 2.5" disk drive form factor.

[0025] In one exemplary implementation, the controller board 102 and each of the memory boards 104a and 104b may be electrically connected using a high density ball grid array (BGA) connector. Other variants of BGA connectors may be used including, for example, a fine ball grid array (FBGA) connector, an ultra fine ball grid array (UBGA) connector and a micro ball grid array (MBGA) connector. Other types of electrical connection means also may be used.

[0026] In one exemplary implementation, the memory chips 118a-118n may include flash memory chips. In another exemplary implementation, the memory chips 118a-118n may include DRAM chips or combinations of flash memory chips and DRAM chips. The memory chips 118a-118n may include other types of memory chips as well.

[0027] In one exemplary implementation, the host 106 using the driver 107 and the data storage device 100 may

communicate commands and responses using pairs of queues or buffers in host memory. Throughout this document, the terms buffer and queue are used interchangeably. 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. In one exemplary implementation, the commands and results may be relatively small, fixed size blocks. For instance, the commands may be 32 bytes and the results or responses may be 8 bytes. In other exemplary implementations, other sized blocks may be used including variable size blocks. Tags may be used to match the results to the commands. In this manner, the data storage device **100** may complete commands out of order.

[0028] 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 and discussed in more detail below. For example, up to and including 32 buffer pairs may be used. In one exemplary implementation, the data storage device **100** may service the multiple command buffers **119** in a round robin fashion, where the data storage device **100** may retrieve a fixed number of commands at a time from each of the command buffers **119**. The response buffer **123** may include its own interrupt and interrupt parameters.

[0029] In one exemplary implementation, each command may refer to one memory page (e.g., one flash page), one erase block or one memory chip depending on the command. Each command that transfers data may include one 4K direct memory access (DMA) buffer. Larger operations may be implemented by sending multiple commands. The driver **107** may be arranged and configured to group together a single operation of multiple commands such that the data storage device **100** processes the commands using the flash memory chips **118a** and **118b** and generates and sends a single interrupt back to the host **106** when the multiple grouped commands have been processed.

[0030] 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. The response buffer **123** also can be configured as a circular queue. Each of the circular queues **159** of the command buffer **119** and the response buffer **123** include a head pointer and a tail pointer. Values of the head pointer of the circular queue **159** of the command buffer **119** can be stored in a register **163** on the host, and values of the tail pointer can be stored in a register **161** on the data storage device **100**. Values of a tail pointer of a circular queue of the response buffer **123** can be stored in a register on the host, and values of the head pointer of the response buffer can be stored in a register on the data storage device **100**. Commands and responses may be inserted into the circular queue **159** at the tail pointer and removed at the head pointer. The host **106** may be the producer of the command buffer **119** and the consumer of the response buffer **123**. The data storage device **100** may be the consumer of the command buffer **119** and the producer of the response buffer **123**. The host **106** may write the command tail pointer and the response head pointer and may read the command head pointer and the response tail pointer. 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. In the data storage device **100**, the controller **110** may perform the read and write actions. More specifically, the command processor **122** may be configured to perform the read and write actions for the

data storage device **100**. No other synchronization, other than the head and tail pointers, may be needed between the host **106** and the data storage device **100**.

[0031] In one exemplary implementation, for performance reasons, the command head pointer and the response tail pointer may be stored in register of the host **106** (e.g., in host RAM). 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**.

[0032] The command buffer **119** and the response buffer **123** may be an arbitrary multiple of the command or response sizes, and the driver **107** and the data storage device **100** may be free to post and process commands and results as needed provided that they do not overrun the command buffer **119** and the response buffer **123**. In one implementation, as described above, the command buffer **119** and the response buffer **123** are circular queues, which enable flow control between the host **106** and the data storage device **100**.

[0033] In one exemplary implementation, the host **106** may determine the size of the command buffer **119** and the response buffer **123**. The buffers may be larger than the number of commands that the data storage device **100** can queue internally.

[0034] The host **106** may write a command to the command buffer **119** and update the command tail pointer, which can reside in memory mapped input/output (“MMIO”) space of the data storage device, to indicate to the data storage device **100** (and, in particular, to the command processor **122** within the data storage device **100**) that a new command is present and ready for communication to the data storage device. The writing of the command tail pointer signals the command processor **122** that a new command is present. The command processor **122** is configured to read the command out of the command buffer **119** using a DMA operation and is configured to update the head pointer using another DMA operation to indicate to the host **106** that the command processor **122** has received the command. Thus, writing a command from the host **106** to the data storage device can include just one write operation to memory mapped input/output space (i.e., the updating of the tail pointer in the MMIO space of the data storage device by the host) and two DMA events (i.e., the command processor reading the command out of the command buffer and updating the head pointer of the circular queue **159**).

[0035] 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. 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 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. In one exemplary implementation, the interrupt processor **124** may not send another interrupt to the host **106** until the previous interrupt has been acknowledged by the host **106** writing to the response head pointer. Thus, receiving a response to the writing of a command can include just one write operation to memory mapped input/output space (i.e., the updating of the head pointer by the host) and two DMA events (i.e., the writing of the response by the command processor and the updating of the response tail pointer to indicate that the command is finished). Neither the writing of the command nor the

reception of the response involves a MMIO read event, which can take a relatively long time compared to MMIO write events and DMA events, and in this manner the communication between the host and the device is accelerated.

[0036] In one exemplary implementation, the host **106**, through its driver **107**, may control when the interrupt processor **124** should generate interrupts. The host **106** may use one or more different interrupt mechanisms, including a combination of different interrupt mechanisms, to provide information to the interrupt processor **124** regarding interrupt processing. 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.

[0037] In one exemplary implementation, the host **106** may set a ResponseMark parameter, which determines the water mark, and may set the ResponseDelay parameter, which determines the timeout. The host **106** may communicate these parameters to the interrupt processor **124**. 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. 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. If the host **106** removes the new response from the response buffer **123**, the count of new responses is updated and the timeout is restarted. In this manner, the host **106** may poll ahead and avoid interrupts from the interrupt processor **124**.

[0038] In another exemplary implementation, the host **106** may use a group interrupt mechanism to determine when the interrupt processor **124** should generate and send interrupts to the host **106**. The commands may share a common value, which identifies the commands as part of the same group. For example, the driver **107** may group commands together and assign a same group number to the group of commands. The driver **107** may use an interrupt group field in the command header to assign a group number to the commands in a group. When all of the commands in a command group have completed, and the responses for all of those commands have been transferred from the command processor **122** to the response buffer **123** and the response tail is updated, then the interrupt processor **124** may generate and send the interrupt to the host **106**. In this manner, the group interrupt mechanism may be used to reduce the time the host **106** needs to spend processing interrupts.

[0039] Each of the interrupt mechanisms may be separately enabled or disabled. Also, any combination of interrupt mechanisms may be used. For example, the driver **107** may set interrupt enable and disable flags in a QueueControl register to determine which of the interrupt mechanisms are enabled and which of the interrupt mechanisms are disabled. In this manner, the combination of the interrupts may be used to reduce the time that the host **106** needs to spend processing interrupts. The host **106** may use its resources to perform other tasks.

[0040] In one exemplary implementation, all of the interrupt mechanisms may be disabled. In this situation, the driver **107** may be configured to poll the response buffer **123** to determine if there are responses ready for processing. Having

all of the interrupt mechanisms disabled may result in a lowest possible latency. It also may result in a high overhead for the driver **107**.

[0041] In another exemplary implementation, the group interrupt mechanism may be enabled along with the timeout interrupt mechanism and/or the water mark interrupt mechanism. In this manner, if the number of commands in a designated group is larger than the response buffer **123**, one of the other enabled interrupt mechanisms will function to interrupt the driver **107** to clear the responses from the response buffer **123** to provide space for the command processor **122** to add more responses to the response buffer **123**.

[0042] The use of the different interrupt mechanisms, either alone or in combinations, may be used to adjust the latency and/or the overhead with respect to the driver **107**. For example, in one exemplary implementation, only the timeout interrupt mechanism may be enabled. In this situation, the overhead on the driver **107** may be reduced. In another exemplary implementation, only the water mark interrupt mechanism may be enabled. In this situation, the latency may be reduced to a lower level.

[0043] In some exemplary situations, a particular type of application being used may factor into the determination of which interrupt mechanisms are enabled. For example, a web search application may be latency sensitive and the interrupt mechanisms may be enabled in particular combinations to provide the best latency sensitivity for the web search application. In another example, a web indexing application may not be as sensitive to latency as a web search application. Instead, processor performance may be a more important parameter. In this application, the interrupt mechanisms may be enabled in particular combinations to allow low overhead, even at the expense of increased latency.

[0044] In one exemplary implementation, the driver **107** may determine a command group based on an input/output (I/O) operation received from an application **113** through the operating system **109**. For example, the application **113** may request a read operation of multiple megabytes. In this instance, the application **113** may not be able to use partial responses and the only useful information for the application **113** may be when the entire operation has been completed. Typically, the read operation may be broken up into many multiple commands. The driver **107** may be configured to recognize the read operation as a group of commands and to assign the commands in that group the same group number in each of the command headers. An interface between the application **113** and the driver **107** may be used to indicate to the driver **107** that certain operations are to be treated as a group. The interface may be configured to group operations based on different criteria including, but not limited to, the type of command, the size of the data request associated with the command, the type of data requested including requests from multiple different applications, the priority of the request, and combinations thereof.

[0045] In some implementations, the application **113** may pass individual command information relating to an operation to the operating system **109** and ultimately to the driver **107**. In other exemplary implementations, the driver **107** may designate one or more commands to be considered a group.

[0046] Referring to FIG. 1B, a block diagram of an exemplary host **106** having multiple queues or buffers. As discussed above with respect to FIG. 1A, the host **106** may include the driver **107**, the operating system **109** and one or more applications **113**. In the example of FIG. 1B, the driver

includes multiple pairs of buffers **219a-219n** and **223a-223n**. The multiple pairs of buffers include a command buffer **219a-219n** and a response buffer **223a-223n** in each pair.

[0047] The pairs work together. For example, the driver **107** may populate the command buffer **219a** with commands for retrieval by the data storage device **100** through the interface **108**. The data storage device **100** generates and communicates responses to those commands, where the responses populate the corresponding response buffer **223a**. The following pairs of buffers are illustrated: command buffer **219a** is paired with response buffer **223a**; command buffer **219b** is paired with response buffer **223b**; command buffer **219c** is paired with response buffer **223c**; and command buffer **219n** is paired with response buffer **223n**.

[0048] The driver **107** may be configured to enable multiple instances of the driver **107** to operate simultaneously. For instance, a separate instance of the driver **107** may be configured to operate with each of the pairs of buffers. In this manner, the driver **107** may use multiple different threads of commands to communicate with the data storage device. For example, one thread may be used to communicate commands and associated responses with the command buffer **219a** and the response buffer **223a**. Another thread may be used to communicate commands and associated response with the command buffer **219b** and the response buffer **223b**.

[0049] 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. 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. 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**.

[0050] Each of the response buffers **223a-223n** may have an associated interrupt handler **225a-225n**. In this manner, each response buffer **223a-223n** may process the interrupts received from the data storage device **100** on an individual basis. 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.

[0051] 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**. For example, each of the buffer pairs may be granted access to the logical to physical address mapping, which may be stored in DRAM **111** of FIG. 1A. In one exemplary implementation, any address mapping or tables such as, for example, the logical to physical address mapping may be shared such that each pair of buffers may have access to the mapping.

[0052] In one exemplary implementation, each of the one or more applications **113** may use one of the command buffer **219a-219n** and response buffer **223a-223n** pairs to communicate with the data storage device **100** through the operating system **109** and an associated instance of the driver **107**.

[0053] In one exemplary implementation, each of the applications **113** may include its own pair of buffers. For example, the application **113** may include an application command buffer **229** and an application response buffer **233**. By having its own pair of buffers **229** and **233**, the application **113** may communicate directly with the data storage device **100**

through the interface **108**. Thus, instead of communicating through the operating system **109** and the driver **107** and a pair of buffers associated with the driver, the application **113** may bypass those components and communicate directly with the data storage device **100**. In this manner, input and output requests generated by the application **113** may be processed by the data storage device **100** faster than if the requests were communicated to the data storage device **100** through the operating system **109** and the driver **107**.

[0054] The application command buffer **229** and the application response buffer **233** may be configured to perform and function in the same manner as described above with respect to the command buffer **119** and the response buffer **123** of FIG. 1A, except that the application command buffer **229** and the application response buffer **233** are associated directly with the application **113** and not the driver **107**.

[0055] In one exemplary implementation, the application **113** may communicate specific command types and input/output requests directly with the data storage device **100** using its own application command buffer **229** and application response buffer **233**. Other command types and input/output requests generated by the application **113** may be processed through the operating system **109** and the driver **107** using one of the pairs of buffers associated with the driver **107**. For example, the application **113** may be configured to communicate read requests directly to the data storage device **100** using the application command buffer **229** and the application response buffer **233**. In this manner, the overall processing time of read requests may be faster than read requests that are processed through the operating system **109** and the driver **107** to the data storage device **100**.

[0056] In the above example where read requests may be communicated directly between the application **113** and the data storage device **100**, other requests and command types may be communicated to the data storage device **100** using the operating system **109** and the driver **107**. For example, write requests generated by the application **113** and garbage collection commands may be processed through the operating system **109** and the driver **107** using one of the driver buffer pairs.

[0057] In one exemplary implementation, the command processor **122** may assign an identifier to the command to indicate to which buffer pair it is associated. The command processor **122** may be configured to direct responses to the appropriate response buffer using the assigned identifier. Similarly, the interrupt processor **124** may be configured to generate an interrupt associated with the appropriate response buffer using the assigned identifier.

[0058] In one exemplary implementation, the controller **110** may include multiple interrupt processors **124** such that each command buffer and response buffer pair is associated with one of the interrupt processors **124**. In this manner, each buffer pair may have one or more different interrupt mechanisms enabled on a per buffer pair basis.

[0059] Referring to FIG. 2, a block diagram of an exemplary interrupt processor **124** is illustrated. The interrupt processor **124** may be configured to generate and send interrupts based on the interrupts mechanism or mechanisms enabled by the host **106**. The interrupt processor **124** may include a ResponseNew counter **280**, a last response timer **282**, group counters **284** and interrupt send logic **286**.

[0060] The ResponseNew counter **280** may be enabled by the host **106** when the watermark interrupt mechanism is desired. The host **106** may set the ResponseMark **288**, which

is a parameter provided as input to the ResponseNew counter **280**, as discussed above. 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. 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. When the counter **280** reaches or exceeds 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**. The watermark level, i.e., the ResponseMark **288**, is the number of new responses in the response buffer **123** needed to generate an interrupt. If the host **106** removes new responses from the response buffer **123**, they do not count toward meeting the watermark level. When an interrupt is generated, the count toward the ResponseMark is reset.

[0061] If the watermark interrupt mechanism is the only interrupt enabled, when the watermark is reached, then the interrupt send logic **286** generates and sends an interrupt to the host **106**. No further interrupts will be sent until the host **106** acknowledges the interrupt and updates the ResponseHead. The updated ResponseHead is communicated to the interrupt send logic **286** as a clear interrupt signal. If other interrupt mechanisms also are enabled, then the interrupt send logic **286** may generate and send an interrupt to the host **106** taking into account the other enabled interrupt mechanisms as well.

[0062] The last response timer **282** may be enabled when the timer interrupt mechanism is desired. The last response timer **282** may be configured to keep track of time since the last interrupt. For instance, the last response timer **282** may track the amount of time since the last interrupt in microseconds. The host **106** may set the amount of time using a parameter, for example, a ResponseDelay parameter **290**. 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.

[0063] The last response timer **282** receives as input a signal indicating when an interrupt is sent. 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**. An interrupt may be generated only if the response buffer **123** contains outstanding responses.

[0064] The last response timer **282** is configured to generate a timeout trigger when the amount of time being tracked by the last response timer **282** is greater than the ResponseDelay parameter **290**. When this occurs and the response buffer **123** contains new responses, then a timeout trigger signal is sent to the interrupt send logic **286**. If the last response timer **282** is the only interrupt mechanism enabled, then the interrupt send logic **286** generates and sends an interrupt to the host. If other interrupt mechanisms also are enabled, then the interrupt send logic **286** may take into account the other interrupt mechanisms as well.

[0065] Each interrupt mechanism includes an enable bit and the interrupt send logic **286** may be configured to generate an interrupt when an interrupt trigger is asserted for an enabled interrupt mechanism. The logic may be configured

to generate another interrupt until the host **106** acknowledges the interrupt and updates the ResponseHead. The Queue Control parameter **292** may provide input to the interrupt send logic **286** to indicate the status of the interrupt mechanisms such as which of the interrupt mechanisms are enabled and which of the interrupt mechanisms are disabled.

[0066] The group counters **284** mechanism may be arranged and configured to track commands that are part of a group as designated by the driver **107**. The group counters **284** may be enabled by the host **106** when the host **106** desires to track commands as part of a group such that a single interrupt is generated and sent back to the host **106** only when all of the commands in a group are processed. In this manner, an interrupt is not generated for each of the individual commands but only for the group of commands.

[0067] The group counters **284** may be configured with multiple counters to enable the tracking of multiple different groups of commands. In one exemplary implementation, the group counters **284** may be configured to track up to and including 128 different groups of commands. In this manner, for each group of commands there is a counter. The number of counters may be related to the number of group numbers that may be designated using the interrupt group field in the command header.

[0068] The group counters **284** may be configured to operate to increment the counter for a group when a new command for the group has entered the command processor **122**. The group counters **284** may decrement the counter for a group when one of the commands in the group has completed processing. In this manner of incrementing as new commands enter for a group and decrementing when commands are completed for the group, the number of commands in each group is potentially unlimited. The counters do not need to be sized to account for the largest number of potential commands in a group. Instead, the counters may be sized based on the number of commands that the data storage device **100** may potentially process at one time, which may be smaller than the unlimited number of commands in a particular group.

[0069] In one exemplary implementation, each of the group counters **284** may track the commands in a specific group using the group number assigned by the driver **107** and appearing in the interrupt group field in the command header of each command. The group counters **284** receive a signal each time a command having a group number enters the command processor **122** for processing. In response to this signal, the counter increments for that group. The group counters **284** also receive a signal each time a command having a group number completes processing. In response to this signal, the counter decrements for that group.

[0070] The last command in the command group may be marked by the driver **107** with a flag to indicate to the group counters **284** that the command is the last command in the group. In one exemplary implementation, the last bit in the interrupt group field in the command header may be used as the flag. The group counters **284** are configured to recognize when the flag is set. In this manner, the group counters **284** keep a counter of the number of commands in a particular group that are in processing in the data storage device **100**. The group counters **284** also track when the end of the group has been seen.

[0071] When a command is sent from the host **106** to the data storage device **100**, the counter for its interrupt group is incremented. When a response is sent from the data storage device **100** to the host **106**, the counter for its interrupt group

is decremented. When the last command in the group is received at the groups counters **284** and the count for that group goes to zero, the group trigger signal is generated and sent to the interrupt send logic **286**. When the group trigger signal is received at the interrupt send logic **286**, then an interrupt is sent to the host **106**. The group counters **284** then clear the end group flag for that group.

[0072] The driver **107** may be configured to track the groups in use. The driver **107** may not re-use an interrupt group number until the previous commands to use that interrupt group have all completed and the interrupt has been acknowledged.

[0073] In one exemplary implementation, the driver **107** may be configured to determine dynamically how many interrupts it wants to have generated. For example, the driver **107** may dynamically determine the size of a command group depending on various criteria including, for instance, volume, latency and other factors on the host **106**.

[0074] In one exemplary implementation, the interrupt send logic **286** may be configured to consolidate multiple interrupts for multiple interrupt groups and only send a single interrupt for multiple groups of commands.

[0075] FIG. 3 is a block diagram of a command processor **122**. The command processor **122** may include a slot tracker module **302**, a command transfer module **304**, a pending command module **306**, a command packet memory **308**, and a task dispatch module **310**. The command processor **122** may be implemented in hardware, software or a combination of hardware and software. In one exemplary implementation, the command processor **122** may be implemented as a part of a field programmable gate array (FPGA) controller. The FPGA controller may be configured using firmware or other instructions to program the FPGA controller to perform the functions discussed herein.

[0076] The command processor **122** may be arranged and configured to retrieve commands from a host and to queue and order the commands from the host for processing by various storage locations. In one exemplary implementation, the command processor **122** may be configured to retrieve commands from each of the command buffers **219a-219n** using a round robin scheme. In another exemplary implementation, the command processor **122** may be configured to retrieve commands from each of the command buffers **219a-219n** using a priority scheme, where the priority of a particular command buffer may be designated by the host **106**. In other exemplary implementations, the command processor **122** may be configured to retrieve commands from each of the command buffers **219a-219n**.

[0077] The command processor **122** may be configured to maximize the availability of the storage locations by attempting to keep all or substantially all of the storage locations busy. The command processor **122** may be configured to dispatch commands designated for the same storage location in order such that the order of the commands received from the host is preserved. The command processor **122** may be configured to reorder and dispatch commands designated for different storage locations out of order. In this manner, the commands received from the host may be processed in parallel by reordering the commands designated for different storage locations and, at the same time, the order of the commands designated for the same storage location is preserved.

[0078] In one exemplary implementation, the command processor **122** may use an ordered list to queue and order the

commands from the host. In one exemplary implementation, the ordered list may be sorted and/or otherwise ordered based on the age of the commands from the host. For instance, as new commands are received from the host, those commands are placed at the bottom of the ordered list in the order that they were received from the host. In this manner, commands that are dependent on order (e.g., commands designated for the same storage location) are maintained in the correct order.

[0079] In one exemplary implementation, the storage locations may include multiple flash memory chips. The flash memory chips may be arranged and configured into multiple channels with each of the channels including one or more of the flash memory chips. The command processor **122** may be arranged and configured to dispatch commands designated for the same channel and/or the same flash memory chip in order based on the ordered list. Also, the command processor **122** may be arranged and configured to dispatch commands designated for different channels and/or different flash memory chips out of order. In this manner, the command processor **122** may, if needed, reorder the commands from the ordered list so that the channels and the flash memory chips may be kept busy at the same time. This enables the commands from the host to be processed in parallel and enables more commands to be processed at the same time on different channels and different flash memory chips.

[0080] The commands from the host may be dispatched and tracked under the control of a driver (e.g., driver **107** of FIG. 1A and FIG. 1B), where the driver may be a computer program product that is tangibly embodied on a storage medium and may include instructions for generating and dispatching commands from the host (e.g., host **106** of FIG. 1A and FIG. 1B). The commands from the host may designate a specific storage location, for example, a specific flash memory chip and/or a specific channel. From the host perspective, it may be important that commands designated for the same storage location be executed in the order as specified by the host. For example, it may be important that certain operations generated by the host occur in order on a same flash memory chip. For example, the host may generate and send an erase command and a write command for a specific flash memory chip, where the host desires that the erase command occurs first. It is important that the erase operation occurs first so that the data associated with the write command doesn't get erased immediately after it is written to the flash memory chip.

[0081] As another example, for flash memory chips, it may be important to write to pages of an erase block in order. This operation may include multiple commands to perform the operation on the same flash memory chip. In this example, it is necessary to perform these commands for this operation in the order specified by the host. For instance, a single write operation may include more than sixty commands. The command processor **122** may be configured to ensure that commands to the same flash memory chip are performed in order using the ordered list.

[0082] In one exemplary implementation, the command processor **122** may be configured to track a number of commands being processed. The command processor **122** may be configured to track a number of available slots for commands to be received and processed. One of the components of the command processor **122**, the slot tracker module **302**, may be configured to track available slots for commands from the host. The slot tracker module **302** may keep track of the open

slots, provide the slots to new commands transferred from the host and designate the slots as open upon completion of the commands.

[0083] In one exemplary implementation, the slot tracker module 302 may include a fixed number of slots, where each slot may be designated for a single command. For example, the slot tracker module 302 may include 128 slots. In other exemplary implementations, the slot tracker module 302 may include a different number of fixed slots. Also, for example, the number of slots may be variable or configurable. The slot tracker module 302 may be implemented as a register or memory module in software, hardware or a combination of hardware and software.

[0084] The slot tracker module 302 may include a list of slots, where each of the slots is associated with a global slot identifier. As commands are received from the host, the commands are assigned to an available slot and associated with the global slot identifier for that slot. The slot tracker module 302 may be configured to assign each of the commands a global slot identifier, where the number of global slot identifier is fixed to match the number of slots in the slot tracker module 302. The command is associated with the global slot identifier throughout its processing until the command is completed and the slot is released. In one exemplary implementation, the global slot identifier is a tag associated with a particular slot that is assigned to a command that fills that particular slot. The tag is associated with the command and remains with the command until processing of the command is complete and the slot it occupied is released and made available to receive a new command. The commands may not be placed in order of slots, but instead may be placed in any of the available slots and assigned the global slot identifier associated with that slot.

[0085] In one exemplary implementation, one of the components of the command processor 122, the command transfer module 304, may be configured to retrieve new commands from the host based on a number of available slots in the slot tracker module 302 and an availability of new commands at the host. In one exemplary implementation, the command transfer module 304 may be implemented as a state machine.

[0086] The slot tracker module 302 may provide information to the command transfer module 304 regarding the number of available slots. Also, the command transfer module 304 may query the slot tracker module 302 regarding the number of available slots.

[0087] In one exemplary implementation, the command transfer module 304 may use a command tail pointer 312 and a command head pointer 314 to indicate when and how many new commands are available at the host for retrieval. The command transfer module 304 may compare the command tail pointer 312 and the command head pointer 314 to determine whether there are commands available for retrieval from the host. If the command tail pointer 312 and the command head pointer 314 are equal, then no commands are available for transfer. If the command tail pointer 312 is greater than the command head pointer 314, then commands are available for transfer.

[0088] In one exemplary implementation, the command tail pointer 312 and the command head pointer 314 may be implemented as registers that are configured to hold a pointer value and may be a part of the command processor 122. The command tail pointer 314 may be written to by the host. For example, the driver may use a memory mapped input/output (MMIO) write to update the command tail pointer 312 when

commands are available at the host for retrieval. As commands are retrieved from the host, the command transfer module 304 updates the command head pointer 314.

[0089] When the conditions of available slots and available commands at the host are met, the command transfer module 304 may retrieve some or all of the available commands from the host. In one exemplary implementation, the command transfer module 304 may retrieve a group of commands in a single access. For example, the command transfer module 304 may be configured to retrieve a group of eight commands at a time using a direct memory access (DMA) operation from the host. When the commands are retrieved, the command transfer module 304 updates the command head pointer 314. The commands may be retrieved from the host through the bus master 316. The command transfer module 304 also may write to a host command head pointer (not shown) through the bus master 316 using a DMA operation to update the host command head pointer.

[0090] The queue control 318 may be configured to enable and disable the command transfer module 304. The queue control 318 may be implemented as a register that receives instructions from the host through the driver. The queue control 318 may be a component of the command processor 122. When the queue control 318 register is set to enable, then the command transfer module 304 may retrieve and process commands from the host. The driver controls the setting of the queue control 318 so that the command transfer module 304 retrieves commands only when the host is ready and has provided the indication that it is ready. When the queue control 318 register is set to disable, then the command transfer module 304 may not retrieve and process command from the host.

[0091] The retrieved commands are each assigned to one of the available slots by the slot tracker module 302 and associated with the global slot identifier for that available slot. The data for the commands may be stored in the command packet memory 308. For example, the command packet memory 308 may be implemented as a fixed buffer that is indexed by global slot identifier. The data for a particular command may be stored in the command packet memory 308 and indexed by its assigned global slot identifier. The data for a particular command may remain in the command packet memory 308 until the command is dispatched to the designated storage location by the task dispatch module 310.

[0092] The command transfer module 304 also may be configured to provide other components of a controller with information related to the commands as indexed by slot. For example, the command transfer module 304 may provide data to a DMA engine. The command transfer module 304 also may provide status packet header data to a status processor. The command transfer module 304 may provide interrupt group data to an interrupt processor. For example, the command transfer module 304 may transfer group information 319 to the interrupt processor (e.g., interrupt processor 124 of FIGS. 1A and 2).

[0093] The pending command module 306 may be configured to queue and order the commands using an ordered list that is based on an age of the commands. In one exemplary implementation, the pending command module 306 may be implemented as a memory module that is configured to store multiple pointers to queue and order the commands. The pending command module 306 may include a list of the global slot identifiers for the commands that are pending along with a storage location identifier. For example, the

storage location identifier may include the designated storage location for where the command is to be processed. The storage location identifier may include a channel identifier and/or a flash memory chip identifier. The storage location identifier is a part of the command and is assigned by the host through its driver.

[0094] When a new command is retrieved, the global slot identifier and storage location information are added to the bottom of the ordered list in the pending command module 306. As discussed above, the data for the commands is stored in the command packet memory 308 and indexed by the global slot identifier. When the command is added to the ordered list, a pointer to the previous command is included with the command. Also included is a pointer to the next command. In this manner, each item in the ordered list includes a global task identifier, a storage location identifier, a pointer to the previous command and a pointer to the next command. In this exemplary implementation, the ordered list may be referred to as a doubly linked list. The ordered list is a list of the commands ordered from oldest to newest.

[0095] The task dispatch module 310 is configured to remove commands from the ordered list in the pending command module 306 and to dispatch them to the appropriate storage location for processing. The task dispatch module 310 may receive input from the storage locations to indicate that they are ready to accept new commands. In one exemplary implementation, the task dispatch module 310 may receive one or more signals 320 such as signals indicating that one or more of the storage locations are ready to accept new commands. The pending command module 306 may be configured to start at the top of the ordered list with the oldest command first and to make that command available to the task dispatch module 310. The pending command module 306 may continue to make commands available to the task dispatch module 310 in order using the ordered list until a command is removed from the list by the task dispatch module 310. After a command is removed from the ordered list in the pending command module 306, the pending command module 306 plays back the commands remaining in the list to the task dispatch module 310 starting again at the top of the ordered list.

[0096] The task dispatch module 310 may be configured to start at the top of the ordered list with the oldest command first and determine whether the storage location is available to receive new commands using the signals 320. If the storage location is ready, then the task dispatch module 310 retrieves the command data from the command packet memory 308 and communicates the command data and a storage location select signal 322 to the storage location. The pending command module 306 then updates the ordered list and the pointers to reflect that the command was dispatched for processing. Once a command has been dispatched, the task dispatch module 310 starts at the top of the ordered list again.

[0097] If the storage location is not ready to receive new commands, then the task dispatch module 310 moves to the next command on the ordered list. The task dispatch module 310 determines if the next command is to the same or a different storage location than the skipped command. If the next command is to a same storage location as a skipped command, then the task dispatch module 310 also will skip this command. In this manner, the commands designated for the same storage location are dispatched and processed in order, as received from the host. The task dispatch module 310 preserves the order of commands designated for the same

storage location. If the commands are designated for a different storage location, the task dispatch module 310 again determines if the storage location for the next command on the list is ready to accept the new command. If the task dispatch module 310 receives a signal 320 that the storage location is ready to accept a new command, then the command is dispatched by the task dispatch module 310 from the command packet memory 308 to the storage location along with a storage location select signal 322. The pending command module 306 removes the dispatched command from the ordered list and updates the ordered list including updating the pointers that were associated with the command. In this manner, the remaining pointers are linked together upon removal of the dispatched command.

[0098] Referring also to FIG. 4, a block diagram of the pending command module 306 is illustrated. The pending command module 306 may include a single memory module 402 having multiple ports, port A and port B. The memory module 402 may store information related to the pending commands, including the pointer information for each command, where the pointer information may point to the next command and the previous command.

[0099] In operation, the command transfer module 304 of FIG. 3 sends a new entry request 406 for a new command to be added to the ordered list to the pending command module 306. The new entry request 406 is received by a new entry module 408. In one exemplary implementation, the new entry module 408 may be implemented as a state machine.

[0100] The new entry module 408 receives the new entry request 406 and adds it to the ordered list at the end of the list as the newest command in memory module 402. Also, the new entry module 408 requests pointers from the free pointer list module 410. The free pointer list module 410 may be implemented as a first-in, first-out (FIFO) memory that maintains a list of pointers that can be used for new entries. When the new entry module 408 requests pointers from the free pointer list module 410, the free pointer list module 410 provides a next entry pointer 412 to the new entry module 408. The next entry pointer 412 is a pointer to where the entry following the current new entry will reside on the ordered list. The current new entry in the list points to this address as its next address.

[0101] The new entry pointer 414 is a pointer to where the next new entry will reside on the ordered list, which was the previous entry's next entry pointer 412. The last entry in the list points to this address as its next address. The memory module 402 stores the data fields related to the commands and the pointers. When a new entry is added, an end pointer 420 also is updated.

[0102] For example, if an entry "X" is to be added, the next entry pointer 412 points to the next entry "Y" and the new entry pointer 414 points to the current entry that is to be added, "X". After "X" is entered and an entry "Y" is to be added, the next entry pointer 412 points to the next entry "Z" and the new entry pointer 414 points to the current entry that is to be added, "Y".

[0103] When the task dispatch module 310 of FIG. 3 determines that an entry is to be removed from the ordered list in the memory module 402, the task dispatch module sends a deletion request 416. The deletion request is received by an entry playback and deletion module 418. The entry playback and deletion module 418 may be configured to start at the top of the ordered list with the oldest command first and to make that command available to the task dispatch module 310. The

entry playback and deletion module 418 may continue to make commands available to the task dispatch module 310 in order using the ordered list until a command is removed from the list by the task dispatch module 310. After a command is removed from the ordered list, the entry playback and deletion module 418 causes the memory module 402 to dispatch the command and remove it from the ordered list. The pointers are then freed up and the entry playback and deletion module 418 provides an indication to the free pointer list module 410 that the pointers for the removed command are free. The entry playback and deletion module 418 also updates the pointers in the memory module 402 when the command is removed to maintain the correct order of the list. The entry playback and deletion module 418 also plays back the commands remaining in the list to the task dispatch module 310 starting again at the top of the ordered list.

[0104] In one exemplary implementation, the entry playback and deletion module 418 may be implemented as a state machine. The entry playback and deletion module 418 also receives an input of the end pointer 420 from the new entry module 408. The end pointer 420 may be used when the entry playback and deletion module 418 is making commands available to the task dispatch module 310 and when a last entry in the ordered list is removed from the list. In this manner, the end pointer 420 may be updated to point to the end of the ordered list.

[0105] Referring back to FIG. 1A, in one exemplary implementation, the controller board 102, which is its own PCB, may be located physically between each of the memory boards 104a and 104b, which are on their own separate PCBs. Referring also to FIG. 5, the data storage device 100 may include the memory board 104a on one PCB, the controller board 102 on a second PCB, and the memory board 104b on a third PCB. The memory board 104a includes multiple flash memory chips 118a and the memory board 104b includes multiple flash memory chips 118b. The controller board 102 includes the controller 110 and the interface 108 to the host (not shown), as well as other components (not shown).

[0106] In the example illustrated by FIG. 5, the memory board 104a may be operably connected to the controller board 102 and located on one side 520a of the controller board 102. For instance, the memory board 104a may be connected to a top side 520a of the controller board 102. The memory board 104b may be operably connected to the controller board 102 and located on a second side 520b of the controller board 102. For instance, the memory board 104b may be connected to a bottom side 520b of the controller board 102.

[0107] Other physical and/or electrical connection arrangements between the memory boards 104a and 104b and the controller board 102 are possible. FIG. 5 merely illustrates one exemplary arrangement. For example, the data storage device 100 may include more than two memory board such as three memory boards, four memory boards or more memory boards, where all of the memory boards are connected to a single controller board. In this manner, the data storage device may still be configured in a disk drive form factor. Also, the memory boards may be connected to the controller board in other arrangements such as, for instance, the controller board on the top and the memory cards on the bottom or the controller board on the bottom and the memory cards on the top.

[0108] The data storage device 100 may be arranged and configured to cooperate with a computing device. In one exemplary implementation, the controller board 102 and the

memory boards 104a and 104b may be arranged and configured to fit within a drive bay of a computing device. Referring to FIG. 6, two exemplary computing devices are illustrated, namely a server 630 and a server 640. The servers 630 and 640 may be arranged and configured to provide various different types of computing services. The servers 630 and 640 may include a host (e.g., host 106 of FIG. 1A and FIG. 1B) that includes computer program products having instructions that cause one or more processors in the servers 630 and 640 to provide computing services. The type of server may be dependent on one or more application programs (e.g., application(s) 113 of FIG. 1A and FIG. 1B) that are operating on the server. For instance, the servers 630 and 640 may be application servers, web servers, email servers, search servers, streaming media servers, e-commerce servers, file transfer protocol (FTP) servers, other types of servers or combinations of these servers. The server 630 may be configured to be a rack-mounted server that operates within a server rack. The server 640 may be configured to be a stand-alone server that operates independent of a server rack. Even though the server 640 is not within a server rack, it may be configured to operate with other servers and may be operably connected to other servers. Servers 630 and 640 are meant to illustrate example computing devices and other computing devices, including other types of servers, may be used.

[0109] In one exemplary implementation, the data storage device 100 of FIGS. 1A, 1B and 5 may be sized to fit within a drive bay 635 of the server 630 or the drive bay 645 of the server 640 to provide data storage functionality for the servers 630 and 640. For instance, the data storage device 100 may be sized to a 3.5" disk drive form factor to fit in the drive bays 635 and 645. The data storage device 100 also may be configured to other sizes. The data storage device 100 may operably connect and communicate with the servers 630 and 560 using the interface 108. In this manner, the host may communicate commands to the controller board 102 using the interface 108 and the controller 110 may execute the commands using the flash memory chips 118a and 118b on the memory boards 104a and 104b.

[0110] Referring back to FIG. 1A, the interface 108 may include a high speed interface between the controller 110 and the host 106. The high speed interface may enable for fast transfers of data between the host 106 and the flash memory chips 118a and 118b. In one exemplary implementation, the high speed interface may include a PCIe interface. For instance, the PCIe interface may be a PCIe x4 interface or a PCIe x8 interface. 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.

[0111] In one exemplary implementation, the communication between the controller board 102 and the flash memory chips 118a and 118b on the memory boards 104a and 104b may be arranged and configured into multiple channels 112. Each of the channels 112 may communicate with one or more flash memory chips 118a and 118b and may be controlled by the channel controllers (not shown). The controller 110 may be configured such that commands received from the host 106 may be executed by the controller 110 using each of the channels 112 simultaneously or at least substantially simultaneously. In this manner, multiple commands may be executed simultaneously on different channels 112, which may improve throughput of the data storage device 100.

[0112] In the example of FIG. 1A, twenty (20) channels 112 are illustrated. The completely solid lines illustrate the ten (10) channels between the controller 110 and the flash memory chips 118a on the memory board 104a. The mixed solid and dashed lines illustrate the ten (10) channels between the controller 110 and the flash memory chips 118b on the memory board 104b. As illustrated in FIG. 1A, each of the channels 112 may support multiple flash memory chips. For instance, each of the channels 112 may support up to 32 flash memory chips. In one exemplary implementation, each of the 20 channels may be configured to support and communicate with 6 flash memory chips. In this example, each of the memory boards 104a and 104b would include 60 flash memory chips each. Depending on the type and the number of the flash memory chips 118a and 118b, the data storage device 100 may be configured to store up to and including multiple terabytes of data.

[0113] The controller 110 may include a microcontroller, a FPGA controller, other types of controllers, or combinations of these controllers. In one exemplary implementation, the controller 110 is a microcontroller. The microcontroller may be implemented in hardware, software, or a combination of hardware and software. For example, the microcontroller may be loaded with a computer program product from memory (e.g., memory module 116) including instructions that, when executed, may cause the microcontroller to perform in a certain manner. The microcontroller may be configured to receive commands from the host 106 using the interface 108 and to execute the commands. For instance, the commands may include commands to read, write, copy and erase blocks of data using the flash memory chips 118a and 118b, as well as other commands.

[0114] 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. For example, the FPGA controller may be loaded with firmware from memory (e.g., memory module 116) including instructions that, when executed, may cause the FPGA controller to perform in a certain manner. The FPGA controller may be configured to receive commands from the host 106 using the interface 108 and to execute the commands. For instance, the commands may include commands to read, write, copy and erase blocks of data using the flash memory chips 118a and 118b, as well as other commands.

[0115] In one exemplary implementation, the FPGA controller may support multiple interfaces 108 with the host 106. For instance, the FPGA controller may be configured to support multiple PCIe x4 or PCIe x8 interfaces with the host 106.

[0116] The memory module 116 may be configured to store data, which may be loaded to the controller 110. For instance, the memory module 116 may be configured to store one or more images for the FPGA controller, where the images include firmware for use by the FPGA controller. The memory module 116 may interface with the host 106 to communicate with the host 106. The memory module 116 may interface directly with the host 106 and/or may interface indirectly with the host 106 through the controller 110. For example, the host 106 may communicate one or more images of firmware to the memory module 116 for storage. In one exemplary implementation, the memory module 116 includes an electrically erasable programmable read-only memory (EEPROM). The memory module 116 also may include other types of memory modules.

[0117] The power module 114 may be configured to receive power (Vin), to perform any conversions of the received power and to output an output power (Vout). The power module 114 may receive power (Vin) from the host 106 or from another source. The power module 114 may provide power (Vout) to the controller board 102 and the components on the controller board 102, including the controller 110. The power module 114 also may provide power (Vout) to the memory boards 104a and 104b and the components on the memory boards 104a and 104b, including the flash memory chips 118a and 118b.

[0118] In one exemplary implementation, the power module 114 may include one or more direct current (DC) to DC converters. The DC to DC converters may be configured to receive a power in (Vin) and to convert the power to one or more different voltage levels (Vout). For example, the power module 114 may be configured to receive +12 V (Vin) and to convert the power to 3.3 v, 1.2 v, or 1.8 v and to supply the power out (Vout) to the controller board 102 and to the memory boards 104a and 104b.

[0119] The memory boards 104a and 104b may be configured to handle different types of flash memory chips 118a and 118b. In one exemplary implementation, the flash memory chips 118a and the flash memory chips 118b may be the same type of flash memory chips including requiring the same voltage from the power module 114 and being from the same flash memory chip vendor. The terms vendor and manufacturer are used interchangeably throughout this document.

[0120] In another exemplary implementation, the flash memory chips 118a on the memory board 104a may be a different type of flash memory chip from the flash memory chips 118b on the memory board 104b. For example, the memory board 104a may include SLC NAND flash memory chips and the memory board 104b may include MLC NAND flash memory chips. In another example, the memory board 104a may include flash memory chips from one flash memory chip manufacturer and the memory board 104b may include flash memory chips from a different flash memory chip manufacturer. The flexibility to have all the same type of flash memory chips or to have different types of flash memory chips enables the data storage device 100 to be tailored to different application(s) 113 being used by the host 106.

[0121] In another exemplary implementation, the memory boards 104a and 104b may include different types of flash memory chips on the same memory board. For example, the memory board 104a may include both SLC NAND chips and MLC NAND chips on the same PCB. Similarly, the memory board 104b may include both SLC NAND chips and MLC NAND chips. In this manner, the data storage device 100 may be advantageously tailored to meet the specifications of the host 106.

[0122] In another exemplary implementation, the memory boards 104a and 104b may include other types of memory devices, including non-flash memory chips. For instance, the memory boards 104a and 104b may include random access memory (RAM) such as, for instance, dynamic RAM (DRAM) and static RAM (SRAM) as well as other types of RAM and other types of memory devices. In one exemplary implementation, the both of the memory boards 104a and 104b may include RAM. In another exemplary implementation, one of the memory boards may include RAM and the other memory board may include flash memory chips. Also, one of the memory boards may include both RAM and flash memory chips.

[0123] The memory modules 120a and 120b on the memory boards 104a and 104b may be used to store information related to the flash memory chips 118a and 118b, respectively. In one exemplary implementation, the memory modules 120a and 120b may store device characteristics of the flash memory chips. The device characteristics may include whether the chips are SLC chips or MLC chips, whether the chips are NAND or NOR chips, a number of chip selects, a number of blocks, a number of pages per block, a number of bytes per page and a speed of the chips.

[0124] In one exemplary implementation, the memory modules 120a and 120b may include serial EEPROMs. The EEPROMs may store the device characteristics. The device characteristics may be compiled once for any given type of flash memory chip and the appropriate EEPROM image may be generated with the device characteristics. When the memory boards 104a and 104b are operably connected to the controller board 102, then the device characteristics may be read from the EEPROMs such that the controller 110 may automatically recognize the types of flash memory chips 118a and 118b that the controller 110 is controlling. Additionally, the device characteristics may be used to configure the controller 110 to the appropriate parameters for the specific type or types of flash memory chips 118a and 118b.

[0125] Referring to FIG. 7, a process 700 is illustrated for communicating commands between a host and a flash memory data storage device. Process 700 may include populating a circular command queue of a driver of the host with commands for retrieval by the data storage device (702). Commands can be sent from the circular command queue to the data storage device via a direct memory access operation (704). A direct memory access operation initiated by the data storage device can be used to populate a circular response queue of the host with responses by the data storage device for retrieval by the host device, where each response acknowledges the reception of a command from the host by the data storage device (706). And responses can be consumed from the circular response queue at the host (708).

[0126] Implementations of the various techniques described herein may be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. Implementations may be implemented as a computer program product, i.e., a computer program tangibly embodied in an information carrier, e.g., in a machine-readable storage device, for execution by, or to control the operation of, data processing apparatus, e.g., a programmable processor, a computer, or multiple computers. A computer program, such as the computer program(s) described above, can be written in any form of programming language, including compiled or interpreted languages, and can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a communication network.

[0127] Method steps may be performed by one or more programmable processors executing a computer program to perform functions by operating on input data and generating output. 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).

[0128] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read-only memory or a random access memory or both. Elements of a computer may include at least one processor for executing instructions and one or more memory devices for storing instructions and data. Generally, a computer also may include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. Information carriers suitable for embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The processor and the memory may be supplemented by, or incorporated in special purpose logic circuitry.

[0129] To provide for interaction with a user, implementations may be implemented on a computer having a display device, e.g., a cathode ray tube (CRT) or liquid crystal display (LCD) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input.

[0130] Implementations may be implemented in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation, or any combination of such back-end, middleware, or front-end components. Components may be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network (LAN) and a wide area network (WAN), e.g., the Internet.

[0131] While certain features of the described implementations have been illustrated as described herein, many modifications, substitutions, changes and equivalents will now occur to those skilled in the art. It is, therefore, to be understood that the appended claims are intended to cover all such modifications and changes as fall within the scope of the embodiments.

What is claimed is:

1. A host device configured for storing data on, and retrieving data from, a flash memory data storage device, the host device comprising:

- a driver that is arranged and configured to communicate commands to the data storage device; and
- a circular command queue that is populated with commands for retrieval by the data storage device, and
- a circular response queue that is populated with responses by the data storage device for retrieval by the host device, wherein each response acknowledges the reception of a command from the host by the data storage device.

2. The host device of claim 1, wherein the circular command queue includes a command head pointer and a command tail pointer and wherein the circular response queue includes a response head pointer and a response tail pointer, the host device further comprising:

- a first register configured to store command head pointer values; and
- a second register configured to store response tail pointer values.

3. The host device of claim 1, wherein the data storage device includes:

- a third register configured to store command tail pointer values; and
- a fourth register configured to store response head pointer values.

4. The host device of claim 3, wherein the third register exists in a memory mapped address space of the data storage device and wherein the driver is configured to write updated command tail pointer values to the third register.

5. The host device of claim 4, wherein the driver is configured to send commands to the storage device in response to a direct memory access request from the data storage device, and wherein the first register is configured to receive updated command head pointer values in response to a direct memory access operation received from the data storage device.

6. The host device of claim 2, wherein the second register exists in the address space of the host device and wherein the second register is configured to receive updated response tail pointer values from the data storage device into the second register.

7. The host device of claim 6, wherein the driver is configured to receive responses from the storage device through a direct memory access operation sent from the data storage device, and wherein the driver is configured to send updated response head pointer values to the data storage device via a write to a Memory Mapped register.

8. The host device of claim 1 further comprising:

- an application that is configured to generate input and output requests; and
- an operating system that is operably coupled to the driver and to the application and that is configured to communicate the input and output requests between the application and the driver.

9. A method for communicating commands between a host and a flash memory data storage device, the method comprising:

- populating a circular command queue of a driver on the host with commands for retrieval by the data storage device;

transferring commands from the circular command queue to the data storage device via a device initiated direct memory access operation;

populating, via a direct memory access operation initiated by the data storage device, a circular response queue of the host with responses by the data storage device for retrieval by the host device, wherein each response acknowledges the reception of a command from the host by the data storage device; and

consuming responses from the circular response queue at the host.

10. The method of claim 9, wherein the circular command queue includes a command head pointer and a command tail pointer and wherein the circular response queue includes a response head pointer and a response tail pointer, the method further comprising:

- storing command head pointer values in a first register of the host; and
- storing response tail pointer values in a second register of the host.

11. The method of claim 9, wherein the data storage device includes:

- a third register configured to store command tail pointer values; and
- a fourth register configured to store response head pointer values.

12. The method of claim 11, wherein the third register exists in a memory mapped address space of the data storage device, the method further comprising writing updated command tail pointer values to the third register.

13. The method of claim 12, further comprising receiving updated command head pointer values into the first register in response to a direct memory access operation received from the data storage device.

14. The method of claim 10, wherein the second register exists in the address space of the host device, the method further comprising receiving updated response tail pointer values into the second register from the data storage device.

15. The method of claim 14, further comprising:

- receiving responses from the storage device through a direct memory access operation sent from the data storage device; and
- sending updated response head pointer values to the data storage device via a write to a Memory Mapped register.

16. The method of claim 9 further comprising:

- generating input and output requests from an application running on the host; and
- communicating the input and output requests from an application running on the host through an operating system to the driver.

* * * * *