

EXHIBIT A

(12) **United States Patent**
Bress et al.

(10) **Patent No.:** US 6,813,682 B2
(45) **Date of Patent:** Nov. 2, 2004

(54) **WRITE PROTECTION FOR COMPUTER
LONG-TERM MEMORY DEVICES**

(76) Inventors: **Steven Bress**, 9801-C Gable Ridge Ter.,
Rockville, MD (US) 20850; **Mark
Joseph Menz**, 114 Rawlings Ct.,
Folsom, CA (US) 95630

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 266 days.

(21) Appl. No.: **09/961,417**

(22) Filed: **Sep. 25, 2001**

(65) **Prior Publication Data**

US 2002/0040418 A1 Apr. 4, 2002

Related U.S. Application Data

(60) Provisional application No. 60/237,761, filed on Sep. 29,
2000.

(51) **Int. Cl.**⁷ **G06F 12/00**

(52) **U.S. Cl.** **711/112**; 711/163; 711/164;
713/161; 713/164; 713/165; 713/166

(58) **Field of Search** 711/112, 164;
713/161, 164, 165, 166

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,654,829 A *	3/1987	Jiang et al.	365/189.03
4,734,851 A	3/1988	Director	364/200
4,811,293 A	3/1989	Knothe et al.	365/189
5,144,660 A	9/1992	Rose	380/4
5,268,960 A	12/1993	Hung et al.	380/4
5,289,540 A	2/1994	Jones	380/4
5,325,499 A	6/1994	Kummer et al.	395/425
5,369,616 A	11/1994	Wells et al.	365/218
5,557,674 A	9/1996	Yeow	380/4
5,687,379 A	11/1997	Smith et al.	395/726
5,991,402 A *	11/1999	Jia et al.	705/59
6,041,385 A	3/2000	Shipman et al.	710/200
6,041,394 A	3/2000	Halligan et al.	711/166

6,092,161 A	7/2000	White et al.	711/163
6,126,070 A *	10/2000	Fukuzumi	235/380
6,212,635 B1 *	4/2001	Reardon	713/165
6,216,205 B1 *	4/2001	Chin et al.	711/131
6,230,288 B1 *	5/2001	Kuo et al.	714/38
6,330,648 B1 *	12/2001	Wambach et al.	711/163
6,336,187 B1 *	1/2002	Kern et al.	713/161
6,345,368 B1 *	2/2002	Bergsten	714/11
6,446,209 B2 *	9/2002	Kern et al.	713/193
6,516,999 B1	2/2003	Bélonožnik	235/382
6,629,184 B1	9/2003	Berg et al.	710/306
6,654,864 B2 *	11/2003	Shaath et al.	711/163

FOREIGN PATENT DOCUMENTS

WO	WO 93/09495	5/1993
WO	WO 93/13477	7/1993
WO	WO 01/88724 A2	11/2001

OTHER PUBLICATIONS

Patent Abstracts of Japan, Publication No. 06337781, Dec.
6, 1994, NEC Home Electron Ltd.

Patent Abstracts of Japan, vol. 1995, No. 05, Jun. 30, 1995
& JP 7 037207, Feb. 7, 1995; 1 page.

Peter Gutmann: "Secure Deletion of Data from Magnetic
and Solid-State Memory," Proceedings of the USENIX
Security Symposium, Jul. 22, 1996; 14 pages.

* cited by examiner

Primary Examiner—Donald Sparks

Assistant Examiner—Ngoc V Dinh

(74) *Attorney, Agent, or Firm*—Harrity & Snyder, LLP

(57) **ABSTRACT**

A blocking device provides read and write protection for
computer long-term storage devices, such as hard drives.
The blocking device is placed between a host computer and
the storage device. The blocking device intercepts commu-
nications between the host and the storage device and
examines any commands from the host to the storage device.
Certain commands, such as commands that may modify the
storage device, may be discarded.

45 Claims, 9 Drawing Sheets

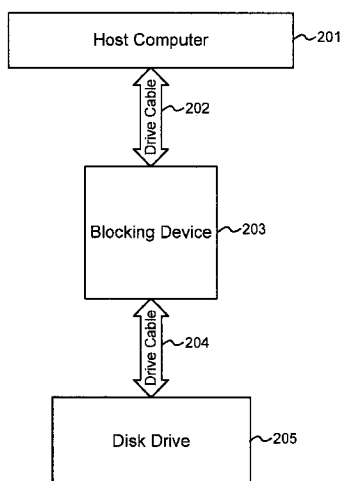


Fig. 1A

Command Register	Read Sector Command
Register	Data
Features	
Sector Count	Sector Count
Sector Number	Sector Number or LBA
Cylinder Low	Cylinder Low or LBA
Cylinder High	Cylinder High or LBA
Device/Head	Device ID and Head Number or LBA
Command	020h

Fig. 1B

Command Register	Write Sector Command
Register	Data
Features	
Sector Count	Sector Count
Sector Number	Sector Number or LBA
Cylinder Low	Cylinder Low or LBA
Cylinder High	Cylinder High or LBA
Device/Head	Device ID and Head Number or LBA
Command	030h

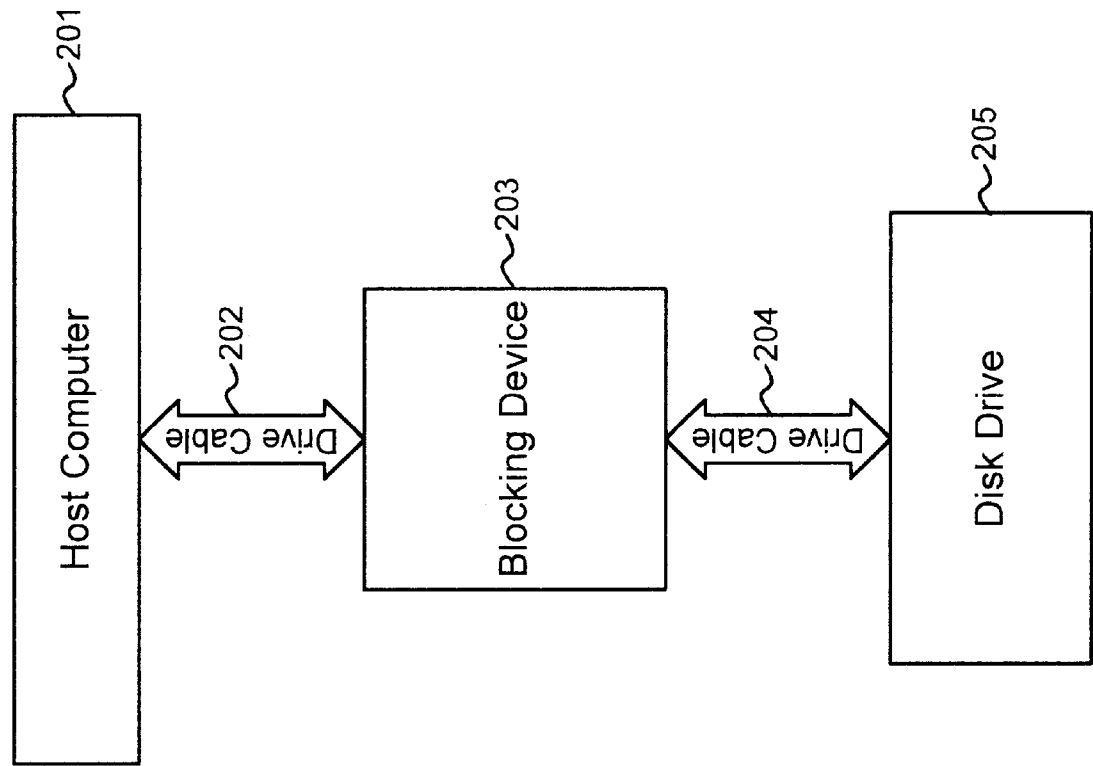


Fig. 2

U.S. Patent

Nov. 2, 2004

Sheet 3 of 9

US 6,813,682 B2

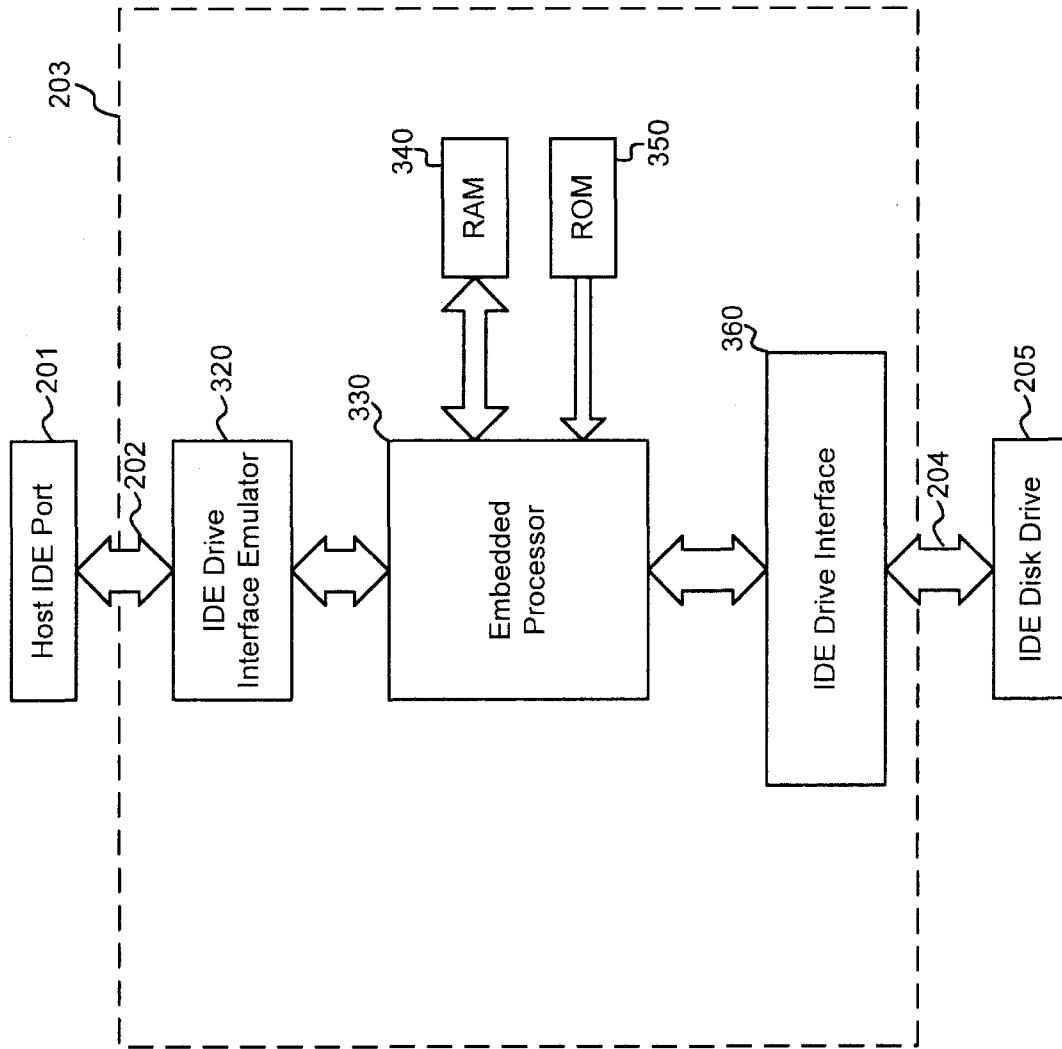


Fig. 3

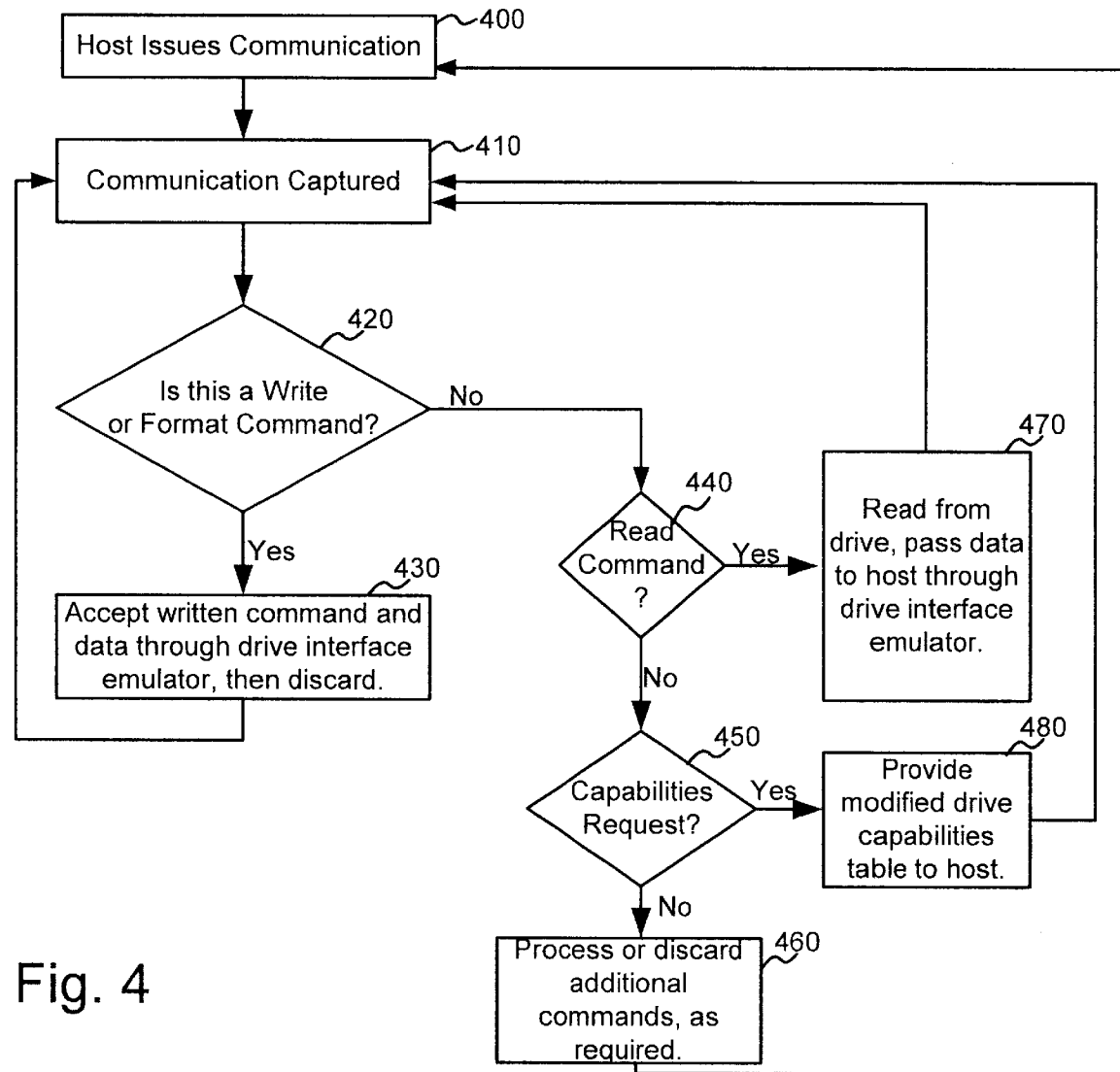


Fig. 4

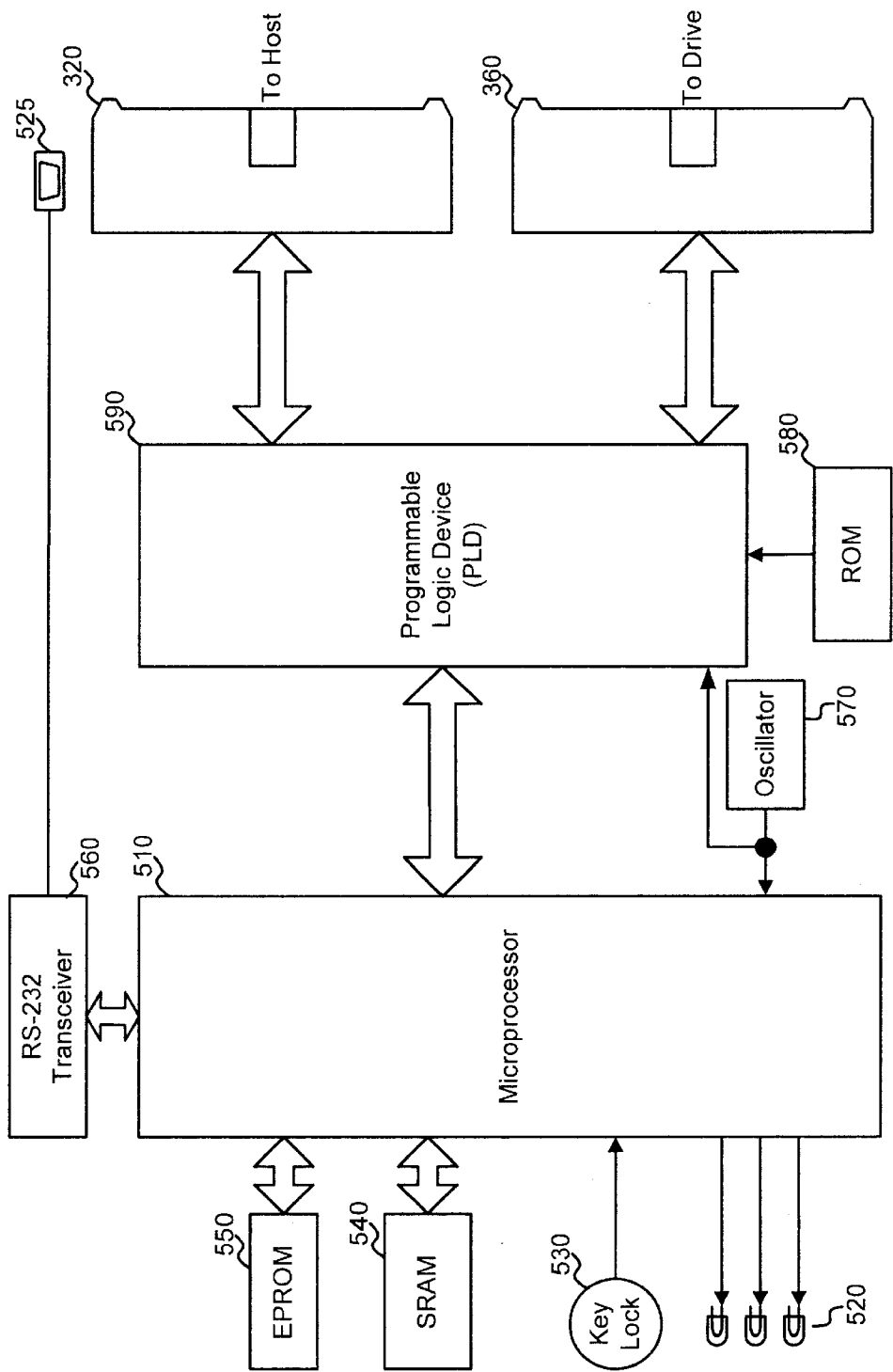
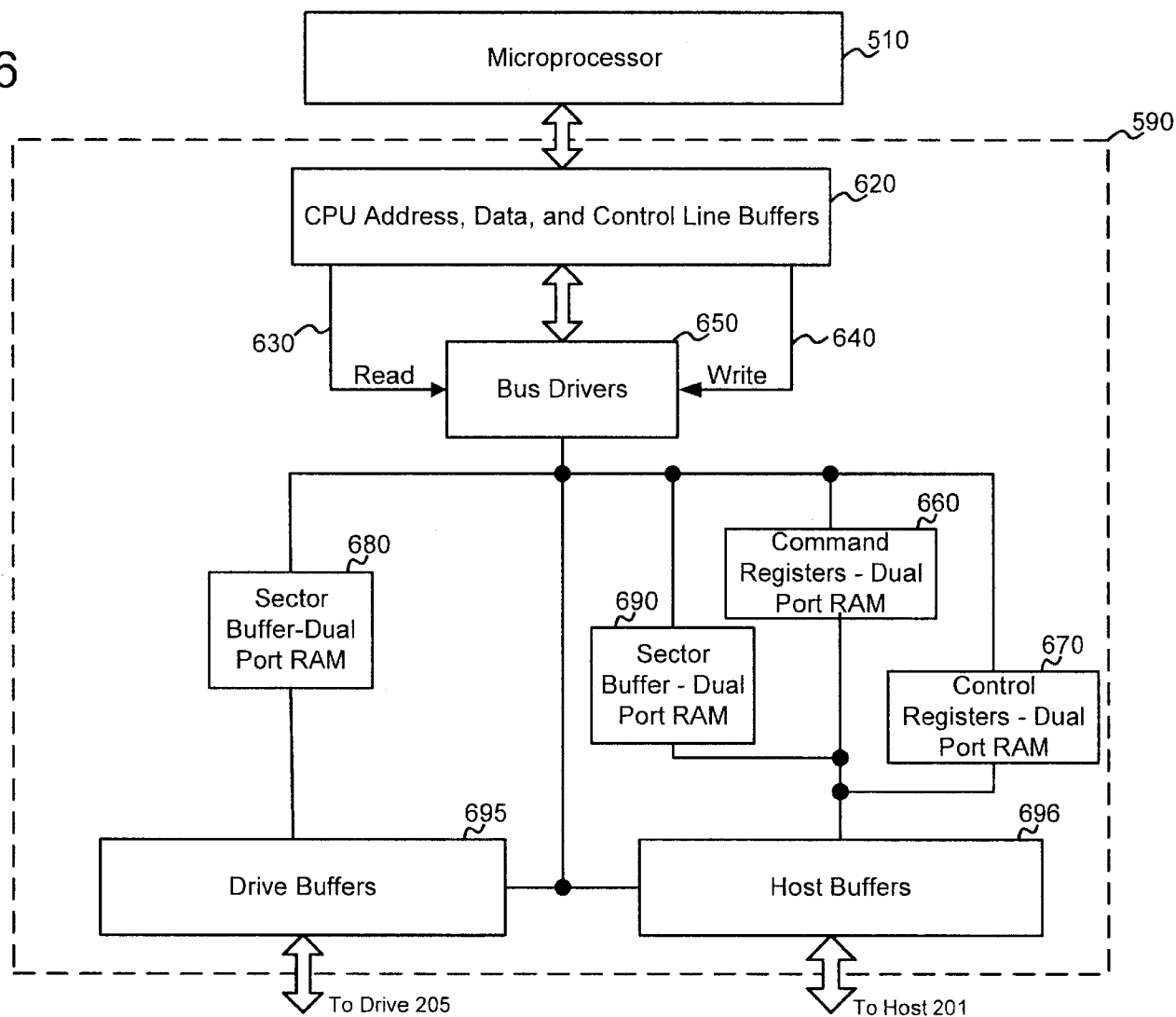


Fig. 5

Fig. 6



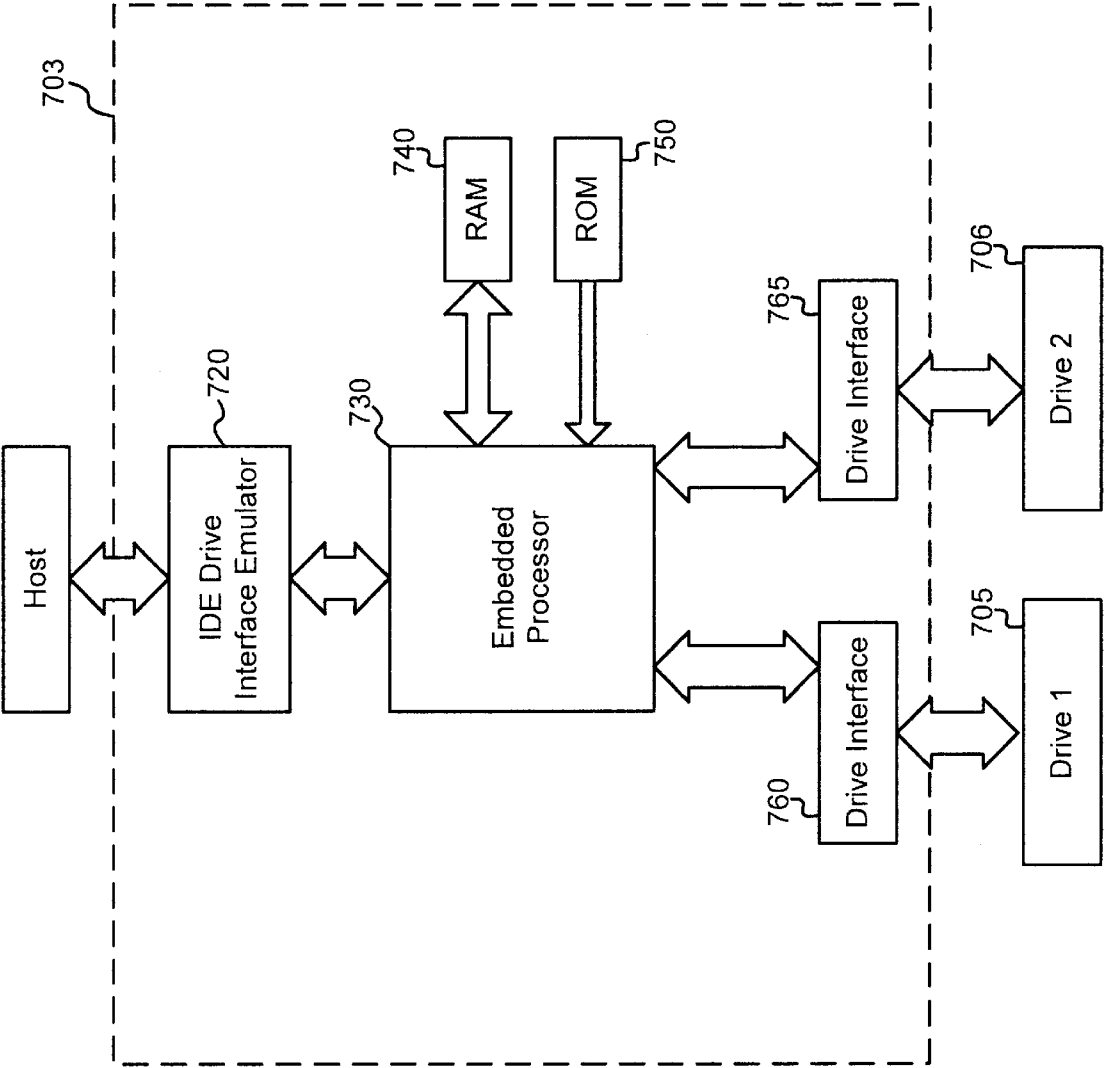


Fig. 7

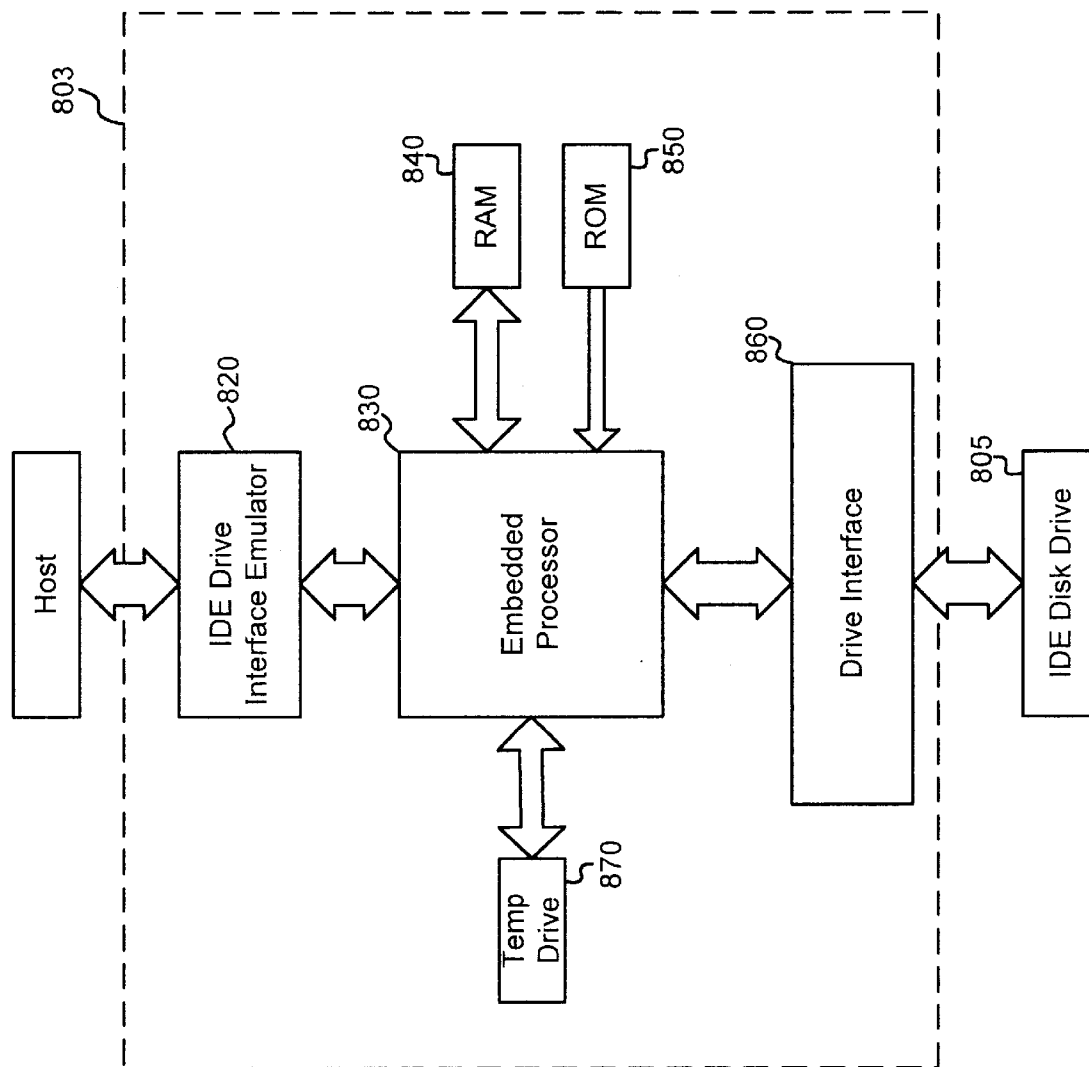


Fig. 8

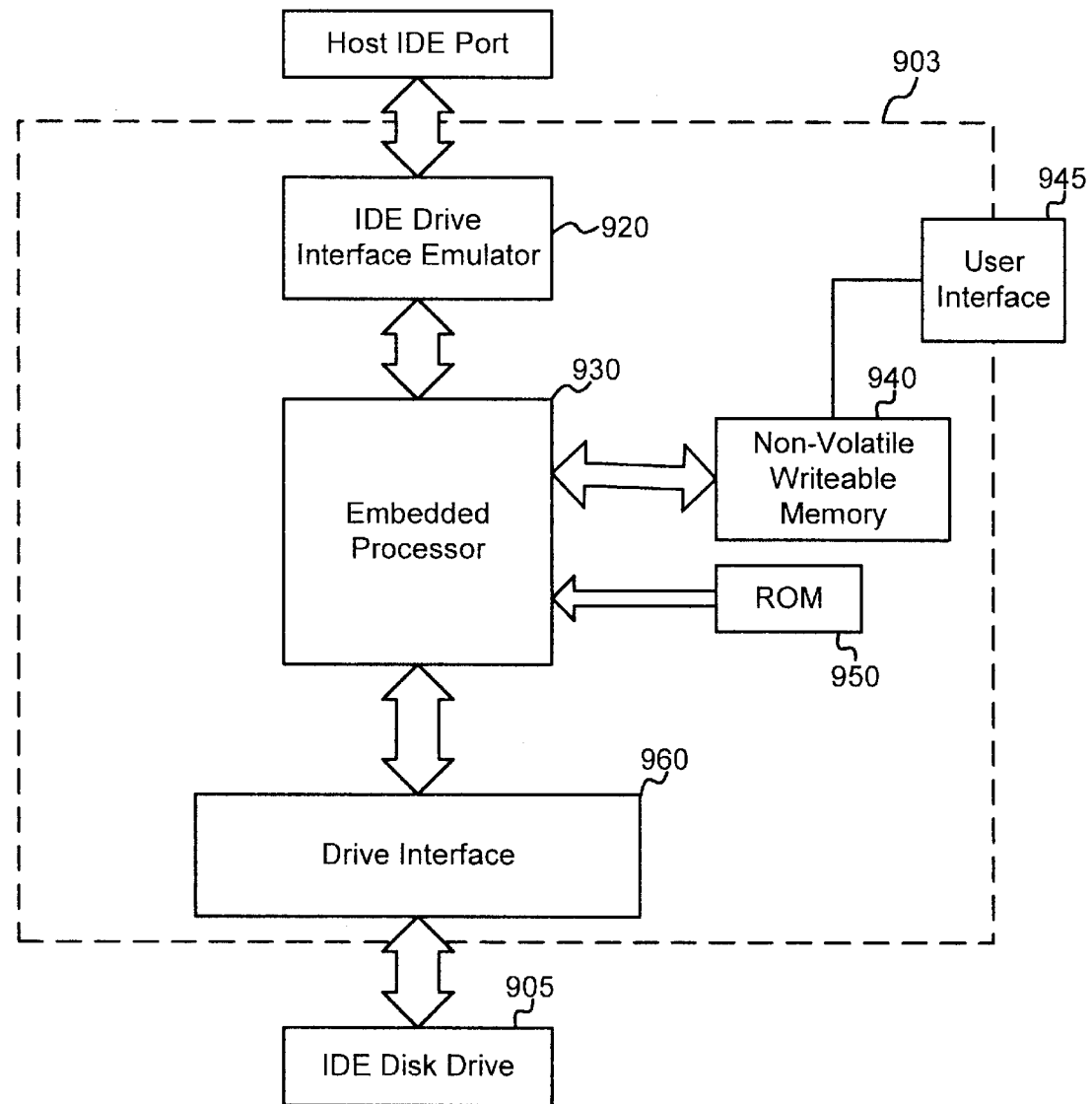


Fig. 9

US 6,813,682 B2

1

**WRITE PROTECTION FOR COMPUTER
LONG-TERM MEMORY DEVICES****RELATED APPLICATION**

This application claims priority under 35 U.S.C. § 119 based on U.S. Provisional Application No. 60/237,761, filed Sep. 29, 2000, the disclosure of which is incorporated herein by reference.

BACKGROUND OF THE INVENTION**1. Field of the Invention**

The present invention relates to computer memory devices and, more specifically, to mechanisms for controlling user access to the memory devices.

2. Description of Related Art

There are many situations in which it is desirable to allow data to be read from a non-volatile long-term memory storage device, such as a computer hard drive, but not allow data to be written to the device. For example, law enforcement officials have occasion to confiscate long-term memory storage devices. Once confiscated, the law enforcement officials need to be able to examine the storage device without changing the storage state of the device. Some operating systems, such as the Windows® operating systems from Microsoft Corporation, may modify the storage device when accessing files on the device, even if the user is only trying to read files from the device. In addition, during startup, operating systems such as Windows® will write up to hundreds of megabytes of data to a storage device as the operating system initializes. These situations are not acceptable when trying to preserve the state of a storage device to its as-confiscated state.

An example of another situation in which it is desirable to allow data to be read from a storage device but not written is in the area of computer security. A computer that is connected to other computers through a network, such as the Internet, is vulnerable to attack. A malicious hacker may attempt to write/change data on a target computer. Although most modern computers have some form of software password protection, these passwords can be bypassed, or "hacked," by a determined attacker. In addition, attacks can come from viruses that are inadvertently transmitted between computers. One way to minimize or prevent damage from such attacks is to block the modification of all of or of certain predetermined sensitive areas of the computer's storage device.

There are a number of known conventional techniques for write protecting memory devices, such as hard drives. One class of early techniques revolved around the concept of disabling a write gate signal transmitted between the drive's controller and the drive's storage media. These techniques are not easy to implement with more modern hard drives, however, because the drive controller and storage media are integrated into a single closed system. Accordingly, the write gate signal is no longer easily accessible. Further, within the more modern integrated hard drives, the write gate signal may be implemented as a signal etched onto an integrated circuit and would, thus, be difficult to electrically contact even if the drive were opened.

A second class of drive write protection techniques is based on software protection of the drive. In general, these techniques involve the installation of software that modifies the read/write parameters of the system. One disadvantage of these techniques is that they tend to be operating system specific. This creates the potential burden of properly

2

installing, updating, and operating the software. Additionally, because installing software may change the state of the storage device, software techniques are not appropriate in situations, such as in law enforcement, in which not changing the state of the storage device is a priority.

A final class of drive protection techniques is based on inserting hardware devices designed to operate with particular computer configurations, such as a card inserted into a host's PCI bus. Such devices are also not without their limitations. For example, the devices may only work with certain types of computer systems or may only block predefined write commands. These limitations can be problematic if the device is not compatible with the desired computer system or if new write commands are introduced which are not recognized by the device.

Accordingly, there is a need in the art for an improved mechanism for write protecting a memory device, such as a disk drive.

SUMMARY OF THE INVENTION

Systems and methods consistent with the present invention address these and other needs by providing for an operating system independent blocking device that is physically inserted between a host computer and a storage device.

One aspect of the invention is directed to a blocking device including a plurality of elements. Specifically, the blocking device includes an interface emulator configured to emulate an interface presented by a storage device and an interface for connecting to a storage device. Additionally, the blocking device includes a processor coupled to the interface emulator and the interface. The processor examines commands received through the interface emulator that are generated by a host and intended for the storage device and allows only those of the commands that match a predetermined set of commands to pass.

A second aspect of the invention is directed to a device that includes an IDE emulator component, an IDE interface, and a logic circuit. The IDE emulator component includes a physical interface designed to engage a first cable that connects to a host that controls an IDE storage device. The IDE interface is configured to engage a second cable that connects to the IDE storage device. The logic circuit connects the IDE emulator component to the IDE interface and compares commands received at the IDE emulator component to a predetermined set of commands and blocks transmission of one or more of the commands from the IDE emulator component to the IDE interface when the comparison indicates that the logic circuit does not recognize the received command or the comparison indicates that the received command is a command that modifies the storage device.

Another device consistent with the invention includes an emulator component, the emulator component including a physical interface designed to connect to a host that controls a storage device. Additionally, an interface is configured to connect to the storage device and a logic circuit connects the emulator component to the interface and is configured to compare information received at the emulator component to a computer virus definition file and to block transmission of storage commands from the emulator component to the interface when the comparison indicates a match with the computer virus definition file.

Another aspect of the invention is a method that intercepts communications between a computer motherboard and a local storage device and compares commands in the com-

US 6,813,682 B2

3

munications between the motherboard and the storage device to a predetermined set of commands. Additionally, the method includes forwarding selected ones of the commands to the storage device based on the comparison and blocking selected other ones of the commands from being received by the storage device based on the comparison.

Yet another aspect of the invention is directed to a computer system. The computer system includes a host computer, a long-term storage device, and a blocking device coupled between the host computer and the storage device. The blocking device is configured to intercept commands from the host to the storage device and to block certain commands from reaching the storage device and to pass other ones of the commands to the storage device.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate the invention and, together with the description, explain the invention. In the drawings,

FIGS. 1A and 1B are diagrams illustrating register layouts for an IDE interface;

FIG. 2 is a diagram illustrating a blocking device consistent with concepts of the invention;

FIG. 3 is a diagram illustrating the blocking device of FIG. 2 in additional detail;

FIG. 4 is a flow chart illustrating the operation of the blocking device;

FIG. 5 is block diagram illustrating the blocking device of FIGS. 2 and 3 in even more detail;

FIG. 6 is diagram graphically illustrating the functionality of portions of the blocking device shown in FIG. 5;

FIG. 7 is a diagram illustrating an embodiment of a blocking device capable of supporting two drives;

FIG. 8 is a diagram illustrating a blocking device capable of correctly operating with an operating system that uses read-back-after-write commands; and

FIG. 9 is a diagram illustrating a blocking device capable of implementing more complex blocking rules.

DETAILED DESCRIPTION

The following detailed description of the invention refers to the accompanying drawings. The same reference numbers in different drawings identify the same or similar elements. Also, the following detailed description does not limit the invention. Instead, the scope of the invention is defined by the appended claims and equivalents.

A blocking device is described herein that blocks certain operations, such as read or write operations, as they are transmitted to a storage device. The blocking device is physically inserted between a host computer system and the storage device and is transparent to the host and the storage device.

The storage device may be any type of long-term non-volatile memory device. For example, the storage device may be a hard disk drive or compact flash memory. In one implementation, the storage device uses an Integrated Drive Electronics (IDE) interface. An IDE interface is a well-known electronic interface that is frequently used to connect a computer's motherboard and disk drive. In IDE drives, the disk drive controller is built into the physical case of the disk drive. The IDE interface provides a relatively high level interface between the motherboard and the disk drive.

Although concepts consistent with the present invention are primarily described herein in relation to an IDE magnetic

4

hard disk drive, these concepts may be implemented with other types of IDE media, such as flash memory with an IDE interface. Flash memories are a special type of semiconductor random access memory that retains its data after power has been removed from the system. Other types of media useable with an IDE interface include magnetic tape and optical media, such as a compact disc (CD) and a digital versatile disc (DVD). In addition to the IDE interface, concepts consistent with the invention may be applied in a straightforward manner to other types of high level storage interfaces, such as the well known Small Computer System Interface (SCSI) standard or a hard drive connected through an IEEE 1394 (Firewire) connection.

For the sake of clarity the remaining description herein will be described with reference to an IDE magnetic hard drive, although, as mentioned above, the concepts of the invention are not limited to such drives. One skilled in the art would appreciate that other modern long-term storage device interfaces share similar functionality that could be incorporated into the concepts described herein.

IDE Drive

As previously mentioned, communications with an IDE drive occurs through its IDE interface. The IDE interface is a well-defined interface that has addressable memory registers in which the host device (e.g., the computer motherboard) can write commands. The host may also read these registers to, for example, retrieve status information. The IDE interface may additionally include memory used to buffer data going to or coming from the storage media.

FIGS. 1A and 1B are diagrams illustrating register layouts for an IDE interface through which a host transmits read commands (FIG. 1A) and write commands (FIG. 1B). When reading from a drive, the host uses sector count register **101**, sector number register **102**, cylinder low register **103**, cylinder high register **104**, device ID/head number register **105**, and command register **106**. The host writes the number of sectors that it would like to read into sector count register **101**. The host writes to registers **102**, **103**, **104**, and part of register **105** to tell the drive how many sectors are to be read by the command. In order to read a sector from a drive, the host writes a series of bytes to these registers.

Register **101** is a sector count register that tells the drive how many sectors should be read by the read command. For older drives, the sector was specified in a more convoluted way, using sector number register **102**, cylinder number registers **103** and **104**, and a head number in register **105**. For backwards compatibility, these designations are still shown in the illustrated drive command table. More recent drives may use the Logical Block Addressing mode, or LBA, to describe the starting sector number. In this mode, the starting sector number is directly specified.

Register **105** has one bit reserved to specify a device number. Up to two drives may share the same IDE cable. This reserved bit is used to select between the two drives. With two drives, one is designated as a "master" device and the other is designated as a "slave" device. Both drives on the cable receive all of the commands, but only the drive that corresponds to the state of the reserved bit will act on commands.

The drive begins to execute commands when the host writes a read command (illustrated as hexadecimal number **020** in FIG. 1A) to register **106**. For a read command, the drive will retrieve data from the drive platters and start transferring data to the host.

FIG. 1B illustrates the register layout for a write command. The register layout is similar to that of a read

US 6,813,682 B2

5

command. Sector count register **101** holds the number of sectors that are to be written. Also, the address of the starting sector is set in sector number register **102**, cylinder low register **103**, cylinder high register **104**, and device ID/head number register **105** in the same manner in which the host sets the starting sector for a read command. The drive begins to execute the write commands when the host writes a write command (illustrated as hexadecimal number 030) to command register **106**. In general, the only difference between the read and write command is in the value written to the command register **106**.

In FIGS. **1A** and **1B**, register **115** is a “feature” register through which an IDE drive may pass data. As can be seen from the above description of the read and write register layouts, the host must write data to at least some of registers **101–106** in order to issue either a read or a write command to the drive. Therefore, for the drive to be read, the interface lines connecting the host to the IDE drive must be allowed to operate. The drive has no way to determine whether the incoming command will be a read or write until the command register is written.

BLOCKING DEVICE

FIG. **2** is a diagram illustrating a blocking device **203** consistent with the present invention. Blocking device **203** may be a physical device inserted between a host computer **201** and a long-term storage device, such as hard disk drive **205**. Host computer **201** may be connected to blocking device **203** through a standard cable **202**. Similarly, drive **205** may be connected to blocking device **203** through a standard cable **204**.

To host computer **201**, blocking device **203** appears to be a standard drive interface, such as an IDE drive interface, and presents to the host **201** the memory, registers, and control signals that a drive would normally present to host **201**. To drive **205**, blocking device **203** appears to be a host computer, and presents to drive **205** the memory, registers, and control signals that host **201** would normally present to drive **205**. In other words, blocking device **203** is transparent to the system. This is advantageous, as blocking device **203** is therefore operating system independent and does not require software to be installed on host **201**. Moreover, blocking device **203** may be physically designed to aid an untrained user in connecting it in the correct direction. More specifically, in one implementation, the cables may be color coded or physically mated to encourage the user to connect the cables in the correct direction. When cables **202** and **204** are plugged into blocking device **203**, the blocking device is completely installed and ready to operate. Accordingly, installation of blocking device **203** can be performed by users that are relatively unsophisticated in the computer field.

FIG. **3** is a diagram illustrating blocking device **203** in additional detail. Blocking device **203** includes three main components: an IDE drive emulator **320**, embedded processor **330**, and IDE drive interface **360**. When host **201** attempts to communicate with drive **205**, the host **201** is actually communicating with IDE drive emulator **320**. Drive emulator **320** delays the communication from host **201** until embedded processor **330** has examined the communication. Embedded processor **330**, based on its examination of the command from host **201**, may either pass the command to IDE drive interface **360** or drop (block) the command. IDE drive interface **360** is a standard IDE drive interface that connects blocking device **203** to drive **205**.

Embedded processor **330** may be additionally coupled to RAM **340** and ROM **350**. RAM **340** and ROM **350** are

6

computer readable media that may store processing instructions and data used by embedded processor **330**.

In operation, if embedded processor **360** determines that a command received at IDE drive interface emulator **320** is an acceptable command to pass along to the drive, such as a read request or a capabilities request, embedded processor **330** passes the command to the registers in drive **205** through IDE drive interface **360**. Additionally, with certain commands, embedded processor **360** may modify the command such that it will not modify the drive before passing the modified command to the drive. IDE drive interface **360** may receive any requested information back from drive **205**. This received information may then pass through embedded processor **330** and IDE drive interface emulator **320** before it is transmitted to host **201**.

If embedded processor **330** determines that a command received through IDE drive interface **320** is a write command, embedded processor **330** drops the command and, thus, does not write anything to drive **205**. Blocking device **203**, however, will continue to accept the correct amount of data from host **201** as specified in the write command. Embedded processor **330** may simply discard this data and may then return status information to host **201** that indicates that the write was successful. From the point of view of host **201**, the data transfer will have succeeded.

Because the only data path to drive **205** goes through blocking device **203**, there is no data path to the drive for even an accidental write, thereby providing absolute write protection. Further, because the blocking device **203** only allows commands that it knows are safe (i.e., commands that will not modify the drive) to pass, the drive cannot be modified by the host.

FIG. **4** is a flow chart illustrating the operation of blocking device **203** in additional detail. To begin, host **201** communicates a command to drive **205** (act **400**). The blocking device **203** captures and holds communications until they are examined (act **410**). The communication is examined for whether it is a write or format command and/or any command that changes data on the drive **205**. If yes, the command and any associated data is accepted by blocking device **203**, and then discarded, blocking it from the protected drive **205** (acts **420** and **430**). Because the blocking device **203** accepts the command and any data associated with the command, such as the data the host **201** intends to write to drive **205**, the host **201** believes the command and associated data has been successfully sent to drive **205**.

Host communications that are not data changing commands are examined to determine if they are read commands (act **440**). If it is a read command, embedded processor **330** passes the command to the protected drive **205** and any returned data is passed to host **201** (act **470**). If the command is not a read command, embedded processor **330** examines the command to determine if it is a capabilities request (act **450**). If the command is a capabilities request, embedded processor **330** reports the drive’s capabilities back to host **201** (act **480**). In some cases, certain drive capabilities may be modified by the processing time required by blocking device **203**. Accordingly, in this situation, embedded processor **330** may modify the reported capabilities to reflect the actual capability of the drive **205** including any latency introduced by blocking device **203** (act **480**). For example, embedded processor **330** may report back the actual drive storage space but may report a modified drive throughput rate that reflects any delay introduced by the blocking device **203**. Finally, if the command is not recognized by embedded processor **330**, the embedded processor **330** may discard the

US 6,813,682 B2

7

command (act 460). By discarding non-recognized commands, embedded processor 330 can ensure that only safe commands are passed.

In an alternate implementation to that described above, embedded processor 330 may maintain a list of "forbidden" commands. Any received commands that are in the list are dropped.

DETAILS OF AN IMPLEMENTATION OF THE BLOCKING DEVICE

FIG. 5 is block diagram illustrating an exemplary implementation of blocking device 203 in additional detail. The blocking device includes microprocessor 510 and programmable logic device (PLD) 590. Microprocessor 510 may be an embedded processor, such as the 80386 EX embedded processor manufactured by Intel Corporation, of Santa Clara, Calif. The integrated design of microprocessor 510 allows relatively little additional circuitry to be used to create a small, dedicated computer system. PLD 590 complements microprocessor 510 by performing logical operations required by the microprocessor 510 or other circuitry of the device 203. ROM 580 stores configuration data that is initially loaded into PLD 590 on start-up. Similarly, EPROM 550 stores the initial code necessary to initialize and run microprocessor 510. Static RAM (SRAM) 540 is also connected to microprocessor 510, and is used for temporary program and data storage. Crystal oscillator 570 provides clocking signals to microprocessor 510 and PLD 590. In one implementation, crystal oscillator 570 generates a 50 MHz clock signal.

Microprocessor 510 may control a number of external devices, such as LED status indicators 520 and a processor key lock 530. Through LED status indicators 520, microprocessor 510 may provide easily understandable feedback to a user. Processor key lock 530 is a physical interface through which a user must insert a physical key to enable microprocessor 510.

In addition to connecting to host 201 and drive 205 through interfaces 320 and 360, respectively, microprocessor 510 may be connected to external devices via RS-232 port 525 and RS-232 transceiver 560. RS-232 port 525 may be a standard DB9 connector that allows connections using a standard DB9 male to female serial cable.

One of ordinary skill in the art will recognize that the components shown in FIG. 5 may be selected from a wide variety of commercially available components. In one implementation, the components in FIG. 5 may be selected as follows: PLD 590, part number EP1 K50QC208-2, available from Altera Corporation of San Jose, Calif.; ROM 580, part number EPC1 PC8, available from Altera Corporation; EPROM 550, part number AT27LV020A90JC, available from Atmel Corporation, of San Jose, Calif.; and SRAM 540, part number CY7C1021V33L12ZCT, available from Cypress Corporation, of San Jose, Calif.

FIG. 6 is diagram that graphically illustrates the functionality of PLD 590 in additional detail. Address, data, and control lines from the processor 510 are routed to PLD 590 where their information is buffered and latched as necessary in buffers 620. Buffers 620 serve to reduce the electrical load on the processor and to stabilize the signal timing. Buffer read and write signals 630 and 640 control the direction of the bus drivers 650. Thus, bus drivers 650 may write data into buffers 620 when read signal 630 is active and read data out of buffers 620 when write signal 640 is active. Buffers 620 and bus drivers 650 help control the data flow and distribution of the address and data busses from the processor 510 to other portions of PLD 590.

8

Buffering and signal conditioning for the disk drive 205 is provided by drive buffers 695, which form the drive interface with the disk drive 205. Through the bus drivers 650, the microprocessor 510 can directly read and write to the drive interface. Instead of directly communicating with drive buffers 695, bus drivers 650 may indirectly communicate with drive buffers 695 through dual ported RAM sector buffer 680. Sector buffer 680 provides an additional layer of buffering between the microprocessor 510 and the drive 205. This allows the drive to write one sector's worth of data to RAM at high speed, while the processor 510 reads a previous sector's worth of data. By allowing the operations to overlap in this fashion, the processor 510 is not restricted to running at the speed of the drive 205, and is free to handle other functions until it needs the data in the sector buffer 680.

Buffering and signal conditioning for host 201 is provided by drive buffers 696, which form the interface with the host 201. Through bus drivers 650, microprocessor 510 can directly read and write to the host buffers 696. A second way that microprocessor 510 and the drive 205 may communicate is through the dual ported RAM sector buffer 690. In a manner similar to the drive interface, the sector buffer 690 allows the host 201 to communicate at high speed without requiring immediate attention from the processor 510.

In addition to sector buffer 690, two other sections of dual ported RAM are provided by the PLD 590. These are control registers dual port RAM 670 and command registers dual port RAM 660. The control registers 670 may include eight bytes of RAM that appears to host 201 to be the hard drive's control register. Similarly, register 660 may be eight bytes of data that appears to the host to be the hard drive's command register. It is through these registers that the host 201 issues commands to the drive 205.

After a command has been written to the command byte of the command register 660, PLD 590 notifies microprocessor 510 that a command is pending. At this point, the acts illustrated in FIG. 4 are initiated. Because the command and control registers are both created with dual port RAM, the processor 510 may wait until the entire command has been issued to interrogate the contents of these registers. This provides for a zero wait state performance to the host 201, allowing for optimal system performance.

MULTIPLE DRIVE SUPPORT

As previously mentioned, a standard IDE interface can support up to two different drives connected through a single cable to the host. Generally, one of the drives is set as the master drive and the other is set as the slave drive. The host selects whether a command is intended for the master or the slave by setting a bit in the device ID/head number register 105.

FIG. 7 is a diagram illustrating an embodiment of a blocking device 703 capable of supporting two drives 705 and 706. Blocking device 703 is similar to blocking device 203, as shown in FIGS. 2, 3, and 5. That is, blocking device 703 is connected to the host through IDE drive interface emulator 720, which connects the host to embedded processor 730. RAM 740 and ROM 750 may store processing instructions and data used by embedded processor 730. Unlike blocking device 203, however, blocking device 703 includes two IDE drive interfaces, labeled as drive interface 760 and drive interface 765.

In operation, embedded processor 730 examines register 105 to determine the intended destination for data received from the host and then forwards the data to the appropriate

US 6,813,682 B2

9

one of drive interfaces **760** and **765**. Generally, jumper switches on the drives are used to set whether a particular drive operates as a master or a slave drive. During start up, embedded processor **730** may read the state of the jumper switches to determine the correct settings for the drives. Subsequently, embedded processor **730** transfers the data received from the host to the appropriate one of the two drives **705** and **706**.

In a conventional dual-drive IDE interface, because a single cable connects both drives to the host, line noise or a line glitch could cause communications from the host meant for one drive to go to the other. With blocking device **703**, however, because there are two independent cables from the blocking device **703** to drives **705** and **706**, the drives are not exposed to the possibility of misinterpreting for which drive a signal is intended. Thus, once a valid signal reaches blocking device **703**, a glitch on the drive cable will not cause an operation by the wrong drive.

Configuring a drive as a master or a slave requires that a user change physical jumpers on the drive. Conflicts may arise if two drives are configured as both master or both slave. Because blocking device **703** provides a unique and isolated IDE interface to each drive that it protects, even if there is a conflict, blocking device **203** can still independently communicate with each of the drives.

Further, blocking device **703** may operate so that while a data transfer is going on between the host and the blocking device **703**, there is no activity on the signals going to a protected drive. This minimizes the possibility that any glitch on the data or control lines could inadvertently cause the drive to write data. In order for a write to occur, numerous registers need to be written to the drive with correct data. With blocking device **703** keeping the signals in a known state, it is unlikely that a random glitch would happen upon a sequence of data that could cause a write.

Blocking device **703** provides other benefits through virtue of its having one drive per IDE interface. Given the wide range of speeds of IDE devices, it may be unwise to run one device at a dramatically different speed than another on the same cable. The slower drive may misinterpret commands not intended for it, with unpredictable results. Accordingly, often when multiple devices are on an IDE cable, the fastest possible data transfer is at the speed of the slowest device. Because blocking device **703** provides each drive with its own interface, each drive can operate at its highest supported speed.

READ VERIFICATION AFTER WRITING

There are situations in which a host writes data to a drive and immediately tries to verify that the data was in fact written by attempting to read it back. In particular, some commands used by some older operating systems, such as DOS, act in this manner. FIG. **8** is a diagram illustrating an exemplary embodiment of a blocking device **803** capable of correctly operating with an operating system that uses read-back-after-write commands.

In general, blocking device **803** is similar to blocking device **203**, except that blocking device **803** additionally includes a "temp drive" **870**, which is a long term storage device such as a hard disk drive. More particularly, as shown in FIG. **8**, blocking device **803** includes an embedded processor **830** that is coupled to drive interface emulator **820**, drive interface **860**, and temp drive **870**. Additionally, embedded processor **830** may be connected to RAM **840** and ROM **850**, which may store data and instructions used by embedded processor **830**.

10

In operation, embedded processor **830**, instead of discarding the data associated with blocked write commands, stores the data in temp drive **870**. The drive being protected (i.e., drive **805**) is not modified. Embedded processor **830** keeps track of the addresses written to temp drive **870**. During a subsequent read operation that reads an address previously written to temp drive **870**, embedded processor **830** returns the data stored in temp drive **870**. For read requests that do not correspond to data previously written to temp drive **870**, data is returned from drive **805**. In this manner, blocking device **803** and drive **805** appear to the host as a normally functioning disk drive.

In one implementation, because it is reasonable that the host may write as much data as drive **805** can hold, temp drive **870** is as large as drive **805**.

After a predetermined amount of time after a write to temp drive **870**, when it is no longer reasonable to expect a read after write verification, embedded processor **830** may erase the data (or otherwise cause it to become not available) written to the temp drive **870**. After this time all new read requests provide data from the protected drive **805**. Blocking device **803** may also erase data on temp drive **870** if a new command comes in to read or write a different area of drive **805**. This would indicate that any read after write verification of the earlier data had already occurred.

Although temp drive **870** is shown integrated within blocking device **803**, in other implementations, temp drive **870** may be a separately attached drive. For example, in the implementation shown in FIG. **7**, the temp drive may be a drive attached to second drive interface **765**.

SELECTIVE BLOCKING

The implementations of the blocking devices described thus far have had a relatively simple blocking mechanism, such as blocking all write commands. In certain situations, however, it may be desirable for the embedded processor to implement more complex blocking rules.

FIG. **9** is a diagram illustrating an exemplary embodiment of a blocking device **903** capable of implementing more complex blocking rules. In general, blocking device **903** is similar to blocking device **203**, except that blocking device **903** additionally includes a user interface **945** and non-volatile writeable memory **940** instead of RAM. Through user interface **940**, which may be, for example, a USB (universal serial bus) port or a simple serial connection, users may store custom programming in memory **940**. Memory **940** may be, for example, flash memory.

Rewritable non-volatile memory **940** may be used to hold user changeable configuration information. The configuration information may, for example, instruct embedded processor **930** to allow write operations to pass to certain portions of drive **905**, but to block all other write operations. In one implementation, in order to improve device security, users may only modify memory **940** through a separate physical cable connected to interface **945**. Before modifying memory **940**, embedded processor **930** may require that the user enter an appropriate user ID and password. Alternatively, blocking device **903** may require the user to insert a hardware key instead of a password before accepting user communications. In this fashion, blocking device **903** can be protected from attack from a remote device. In addition, blocking device **903** is protected from a local attack as long as the user IDs and passwords are kept secure.

Users may specify in memory **940** areas of drive **905** that the embedded processor **930** is to block (or not block) from write access. A user may configure memory **940** such that

US 6,813,682 B2

11

only a portion of drive **905** is blocked. This may be described in terms of a range of sectors. For a host connected to a network such as the Internet, this could provide absolute security against unauthorized access, whether accidental or intentional.

Often, the goal of a hacker is to disrupt the operating system on a host computer, causing it to crash. In other cases, the hacker tries to change restricted data on the system. With blocking device **903**, the area of the drive **905** containing the operating system could be protected against writing, while user data areas, such as Web sites where data is collected from customers, could remain changeable. In this implementation, no type of attack can permanently harm important system files.

In a similar manner, it is possible to block unauthorized reads with blocking device **903**. In this implementation, blocking device **903** may be configured with a list of passwords or other authorization information, and a corresponding list of areas of the drive **905** that require authorization to access. If a user does not provide appropriate authorization information, read and write access to the drive **905** is denied by blocking device **903**.

VIRUS BLOCKING

Blocking device **903**, in an alternate implementation, may be programmed to provide computer virus protection. Memory **940**, or another memory accessible directly from the host, may be programmed with a virus definition table. Embedded processor **930** may then compare the data associated with write commands from the host against the virus definition table. If the comparison indicates that a virus may be present, embedded processor **930** can block the suspected write operation(s). In this manner, virus checks can be made in real-time by embedded processor **930** without taking any processing resources from the host.

Blocking device **903** may additionally provide feedback to users or to the host relating to the status of its virus checking. The feedback may be communicated in a number of ways. For example, a USB port or a serial port may be used to send information to an external device or to the host. Alternatively, a simple light may be actuated or the information may be transmitted back to the host as a drive error.

NEW FEATURE PROTECTION

More recent IDE drives may support additional features or commands not supported by older IDE drives. A host may determine the feature supported by a particular drive by issuing an "identify device" command to the drive. The drive responds by returning a data structure indicating its supported features. In one implementation, blocking device **203** (FIG. 3) may examine the data structure received from drive **205** and zero out any features not supported by the blocking device **203**. In this manner, if a future version of an IDE drive supports a command, such as an erase command, that embedded processor **330** was not programmed to recognize, blocking device **203** could inform the host that the drive does not support this feature. Accordingly, the host would not attempt to modify the drive using that command.

HOSTS THAT USE LOCAL WRITE CACHE

Some host computer systems may use a local cache to improve the performance of their disk drive. In this local cache, data written to the disk drive is also written to local memory, such as local semiconductor random access memory. If a subsequent read command requests a portion

12

of the data that was previously written to the cache, the host can retrieve the data directly from the cache and will not have to read from the slower disk drive.

Hosts using a local cache may encounter problems when connected to a disk drive through the above-described implementations of the blocking device. In particular, with the blocking device installed, the data in the cache may not accurately reflect the data on the disk drive, as the data on the disk drive was never truly written. Accordingly, in systems that include such a local hard drive cache, the user may, if allowed by the operating system, simply turn off the local hard drive cache.

In another implementation, blocking device **203** may support a removable drive feature set and report itself to the host as a removable drive. Many operating systems support the removable drive feature set. One feature of the removable drive feature set is that a write request may be rejected by the target drive with a "write protected error" code. This allows the operating system to gracefully fail and to not mark the write cache as valid. Accordingly, in operating systems that support the removable drive feature set and that use a local disk cache, embedded processor **330**, in addition to dropping data that the host attempts to write to the disk drive, may also issue a write protected error code to the host.

In yet another implementation, the blocking device may report to the host that the drive media is either not present or has been changed since the last write. In both of these cases, the operating system should fail gracefully and not mark the write cache data as valid.

SUPPORT FOR OBSOLETE COMMANDS

Embedded processor **330** may be configured to respond to certain commands with a predetermined response pattern. For example, in systems with older BIOS (basic input/output system) software, the BIOS may issue a "recalibrate drive" command during system start-up. Generally, the recalibrate drive command causes the hard drive to perform a potentially time consuming calibration operation that may modify the drive. Embedded processor **330** may handle the recalibrate drive command by accepting the command and reporting a successful completion of the command to the host. However, embedded processor **330** does not pass the command to the disk drive. In this manner, the system is able to successfully initialize.

SUMMARY

As described above, a blocking device is inserted between a host computer system and a storage device. The blocking device blocks certain commands, such as write commands, from being sent to the storage device. An embedded processor within the blocking device controls functionality of the blocking device. The functionality of the embedded processor can be programmably modified to allow for a number of different possible blocking options.

Although the blocking device has been primarily described as blocking write commands, one of ordinary skill in the art will appreciate that the blocking device could instead or additionally block read commands.

It will be apparent to one of ordinary skill in the art that the embodiments as described above may be implemented in many different forms of software, firmware, and hardware in the implementations illustrated in the figures. The actual software code or specialized control hardware used to implement aspects consistent with the present invention is not limiting of the present invention. Thus, the operation and

US 6,813,682 B2

13

behavior of the embodiments were described without specific reference to the specific software code, it being understood that a person of ordinary skill in the art would be able to design software and control hardware to implement the embodiments based on the description herein.

The foregoing description of preferred embodiments of the present invention provides illustration and description, but is not intended to be exhaustive or to limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practice of the invention.

No element, act, or instruction used in the description of the present application should be construed as critical or essential to the invention unless explicitly described as such. Also, as used herein, the article "a" is intended to include one or more items. Where only one item is intended, the term "one" or similar language is used.

The scope of the invention is defined by the claims and their equivalents.

What is claimed:

1. A blocking device comprising:

an interface emulator configured to emulate an interface presented by a storage device and configured to connect to a host;

an interface for connecting to the storage device; and
a processor coupled to the interface emulator and the interface, the processor examining commands received through the interface emulator that are generated by the host and intended for the storage device, the processor allowing only those of the commands that match a predetermined set of commands to pass to the storage device via the interface, the predetermined set of commands being commands that are known to not permanently modify a state of the storage device, wherein the blocking device is transparent to normal operation of the host and the storage device.

2. The blocking device of claim 1, wherein the interface is an integrated device electronics (IDE) interface for a disk drive.

3. The blocking device of claim 1, wherein the processor receives data back from the storage device in response to the commands passed to the storage device and forwards the received data to the host through the interface emulator.

4. The blocking device of claim 3, wherein, when the commands include a capabilities request command relating to the storage device, the processor modifies data received from the storage device relating to the capabilities request command to reflect the capability of the storage device as affected by the presence of the blocking device.

5. The blocking device of claim 1, wherein the processor drops those of the commands that do not match the predetermined set of commands, and, after dropping one of the commands, returns status information to the host that indicates that the dropped command was successfully completed.

6. The blocking device of claim 1, further comprising: additional interfaces for connecting to additional storage devices.

7. The blocking device of claim 6, wherein each of the interfaces is independently coupled to the processor.

8. The blocking device of claim 1, further including light emitting diodes (LEDs) coupled to the processor and configured to transmit status information relating to the status of the blocking device.

9. The blocking device of claim 1, further including:
a temporary storage device coupled to the processor, the processor storing data from the host corresponding to at

14

least one command that does not match the predetermined set of commands in the temporary storage device.

10. The blocking device of claim 9, wherein when read commands are received from the host that refer to data stored in the temporary storage device, the processor returns the data from the temporary storage device to the host.

11. The blocking device of claim 1, wherein the processor examines feature information from the storage device that relate to features supported by the storage device and the processor zeroes any features not supported by the processor before making the feature information available to the host.

12. The blocking device of claim 1, wherein the processor supports a removable drive feature set with the host and the processor returns a write protected error code to the host when the processor drops one of the commands.

13. A device comprising:

an IDE emulator component, the IDE emulator component including a physical interface designed to engage a first cable that connects to a host that controls an IDE storage device;

an IDE interface configured to engage a second cable that connects to the IDE storage device; and

a logic circuit connecting the IDE emulator component to the IDE interface and configured to: compare commands received at the IDE emulator component to a predetermined set of commands that are known to not modify a state of the IDE storage device, and to allow transmission of the commands from the IDE emulator component to the IDE interface when the comparison indicates that the received command is in the predetermined set of commands, wherein

the device operates transparently to normal operation of the host and the IDE storage device.

14. The device of claim 13, wherein the logic circuit includes:

an embedded processor,

a computer memory connected to the embedded processor, the embedded processor loading program instructions from the computer memory during device initialization, and

a programmable logic device (PLD) coupled to the embedded processor, the IDE emulator component, and the IDE interface.

15. The device of claim 14, wherein the PLD includes:

a bus driver component configured to transfer data between the embedded processor, the IDE emulator component, and the IDE interface,

a first dual port memory buffer connected between the bus driver and the IDE interface,

a first set of communication lines connecting the bus driver directly to the IDE interface and indirectly to the IDE interface through the first dual port memory buffer,

a second dual port memory buffer connected between the bus driver and the IDE emulator component, and

a second set of communication lines connecting the bus driver directly to the IDE emulator component and indirectly to the IDE emulator component through the second dual port memory buffer.

16. The device of claim 13, wherein when the logic circuit receives data back from the IDE storage device the logic circuit forwards the received data to the host through the IDE emulator component.

17. The device of claim 16, wherein, when the comparison indicates the command includes a capabilities request

US 6,813,682 B2

15

command relating to the IDE storage device, the logic circuit modifies data received from the IDE storage device relating to the capabilities request command to reflect the capability of the IDE storage device as affected by the presence of the device.

18. The device of claim 13, wherein the logic circuit commands not in the predetermined set of commands and, after blocking transmission of one of the commands, returns status information to the host that indicates that the blocked command was successfully executed.

19. The device of claim 13, further comprising:

a second interface for connecting to a second IDE storage device.

20. The device of claim 19, wherein each of the interfaces is independently coupled to the logic circuit.

21. The device of claim 13, further including light emitting diodes (LEDs) coupled to the logic circuit and configured to transmit status information relating to the status of the device.

22. The device of claim 13, further including:

a temporary storage device coupled to the logic circuit, the logic circuit storing data corresponding to commands that are not allowed to be transmitted to the IDE interface in the temporary storage device.

23. The device of claim 22, wherein when read commands are received from the host that refer to data stored in the temporary storage device, the logic circuit returns the data from the temporary storage device to the host.

24. The device of claim 13, wherein the logic circuit examines feature information from the IDE storage device that relates to features supported by the IDE storage device and removes any feature information not supported by the device before making the feature information available to the host.

25. A method comprising:

intercepting communications between a computer motherboard and a local non-volatile storage device for the motherboard;

comparing commands in the communications between the motherboard and the storage device to a predetermined set of commands;

forwarding selected ones of the commands to the storage only when, based on the comparison, the commands are determined to be commands that are in a predetermined set of commands known to not permanently modify a state of the storage device; and

blocking other commands from being received by the storage device, wherein

the intercepting communications, comparing commands, forwarding selected ones of the commands, and blocking selected other ones of the commands is transparent to normal operation of the computer motherboard and the storage device.

26. The method of claim 25, further comprising:

forwarding data from the storage device to the motherboard in response to a read command received from the motherboard and forwarded to the storage device.

27. The method of claim 25, wherein the storage device is an integrated device electronics (IDE) disk drive.

28. The method of claim 25, wherein the commands forwarded to the storage device include a capabilities request command, the method further comprising:

modifying data received from the storage device relating to the capabilities request command to reflect the capability of the storage device as modified by operation of the method.

16

29. The method of claim 28, further comprising, after blocking a command:

returning status information to the motherboard that indicates that the blocked command was successfully executed by the storage device.

30. A computer system comprising:

a host computer;

a long-term storage device; and

a blocking device coupled between the host computer and the storage device, the blocking device configured to: intercept commands from the host to the storage device,

pass commands to the storage device only when the commands are in a predetermined set of commands that are known to not permanently modify a state of the storage device, and

block other commands from reaching the storage device,

wherein the intercepting commands, blocking commands, and passing commands are performed by the blocking device transparently to the host computer and the long-term storage device.

31. The computer system of claim 30, wherein the blocking device further includes:

an interface emulator configured to emulate the storage device to the host; and

an interface configured to connect the blocking device to the storage device.

32. The computer system of claim 31, wherein the interface emulator emulates an Integrated Device Electronics (IDE) interface and the storage device is an IDE disk drive.

33. The computer system of claim 30, wherein the blocking device receives data back from the storage device in response to one of the passed commands and forwards the received data to the host.

34. The computer system of claim 30, wherein, when the passed commands include a capabilities request command relating to the storage device, the blocking device modifies data received from the storage device relating to the capabilities request command to reflect the capability of the storage device as affected by the presence of the blocking device.

35. The computer system of claim 30, wherein the blocking device, after blocking one of the commands, returns status information to the host that indicates that the blocked command was successfully completed.

36. The computer system of claim 30, wherein the blocking device further includes light emitting diodes (LEDs) configured to transmit status information relating to the status of the blocking device.

37. The computer system of claim 30, wherein the blocking device further includes:

a temporary storage device, the blocking device storing data from the host corresponding to blocked commands in the temporary storage device.

38. The computer system of claim 37, wherein when read commands are received from the host that refer to data stored in the temporary storage device, the blocking device returns the data from the temporary storage device to the host.

39. The computer system of claim 30, wherein the blocking device further includes:

a user configurable memory, the user configurable memory storing instructions that define protected areas on the storage device, the blocking device dropping those of the commands that would otherwise modify the protected areas on the storage device.

US 6,813,682 B2

17

40. A blocking device comprising:
 means for intercepting communications between a host
 and a storage device;
 means for comparing commands in the communications
 between the host and the storage device to a predeter-
 mined set of commands;
 means for forwarding selected ones of commands in the
 intercepted communications to the storage device only
 when, based on the comparison, the commands that are
 in a predetermined set of commands are determined to
 be commands that are known to not permanently
 modify a state of the storage device; and
 means for blocking other ones of the commands from
 being received by the storage device based on the
 comparison, wherein
 the blocking device operates transparently to normal
 operation of the host and the storage device.
 41. The blocking device of 40, wherein the storage device
 is an integrated device electronics (IDE) disk drive.

18

42. The blocking device of 40, wherein the commands
 forwarded to the storage device include a capabilities
 request command, and the means for forwarding further
 comprises:

means for modifying data received from the storage
 device relating to the capabilities request command to
 reflect the capabilities of the blocking device.

43. The blocking device of 40, further comprising:

means for returning status information to the host that
 indicates that the blocked command was successfully
 executed by the storage device.

44. The blocking device of claim 2, wherein the interface
 emulator is configured to emulate an IEEE 1394 connection.

45. The computer system of claim 31, wherein the inter-
 face emulator emulates an IEEE 1394 connection and the
 storage device is an IDE disk drive.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,813,682 B2
DATED : November 2, 2004
INVENTOR(S) : Steven Bress et al.

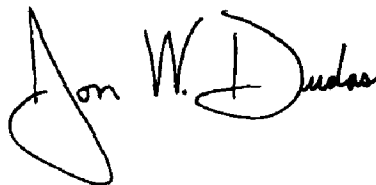
Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 15,
Line 6, insert -- blocks -- after "circuit."

Signed and Sealed this

Eighteenth Day of January, 2005

A handwritten signature in black ink, reading "Jon W. Dudas". The signature is stylized, with a large, looped initial "J" and a cursive "Dudas".

JON W. DUDAS
Director of the United States Patent and Trademark Office

EXHIBIT B

(12) **United States Patent**
Bress et al.

(10) **Patent No.:** **US 7,159,086 B2**
(45) **Date of Patent:** **Jan. 2, 2007**

(54) **SYSTEMS AND METHODS FOR CREATING EXACT COPIES OF COMPUTER LONG-TERM STORAGE DEVICES**

(76) Inventors: **Steven Bress**, 17917 Wheatridge Dr., Germantown, MD (US) 20874; **Mark Joseph Menz**, 114 Rawlings Ct., Folsom, CA (US) 95630

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 323 days.

(21) Appl. No.: **10/765,343**

(22) Filed: **Jan. 27, 2004**

(65) **Prior Publication Data**

US 2004/0186969 A1 Sep. 23, 2004

Related U.S. Application Data

(60) Provisional application No. 60/443,387, filed on Jan. 29, 2003.

(51) **Int. Cl.**
G06F 12/16 (2006.01)
G06F 12/14 (2006.01)

(52) **U.S. Cl.** **711/162; 711/170**

(58) **Field of Classification Search** **711/162, 711/165**

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,516,431 A * 6/1970 Wagner et al. 137/266

OTHER PUBLICATIONS

McLean, Pete. "Device Configuration Overlay Proposal". Oct. 24, 2000, Maxtor Corporation, pp. 1-16.*

* cited by examiner

Primary Examiner—Matthew Kim
Assistant Examiner—Lev Iwashko

(57) **ABSTRACT**

A Simple Device for creating exact copies of computer long-term memory devices such as hard drives and compact flash memory. Our current invention is a stand-alone device. A user connects a long-term memory device he desires to make a copy of (source) to our device. The user also connects a long-term memory device to receive this copy (destination). Our device contains logic and circuitry, which perform operations on both the source and destination device to make an exact copy of the Source Data to the Destination Device, while protecting the Source device from any changes. Our device has a very simple user interface. This simplified user interface makes it difficult and unlikely to use our device incorrectly.

21 Claims, 7 Drawing Sheets

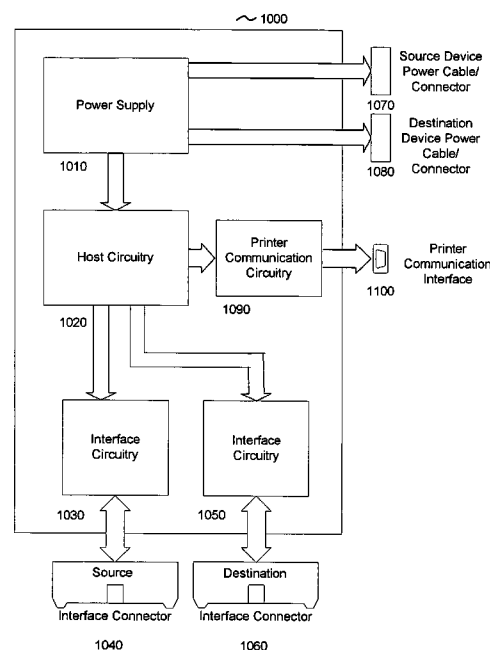
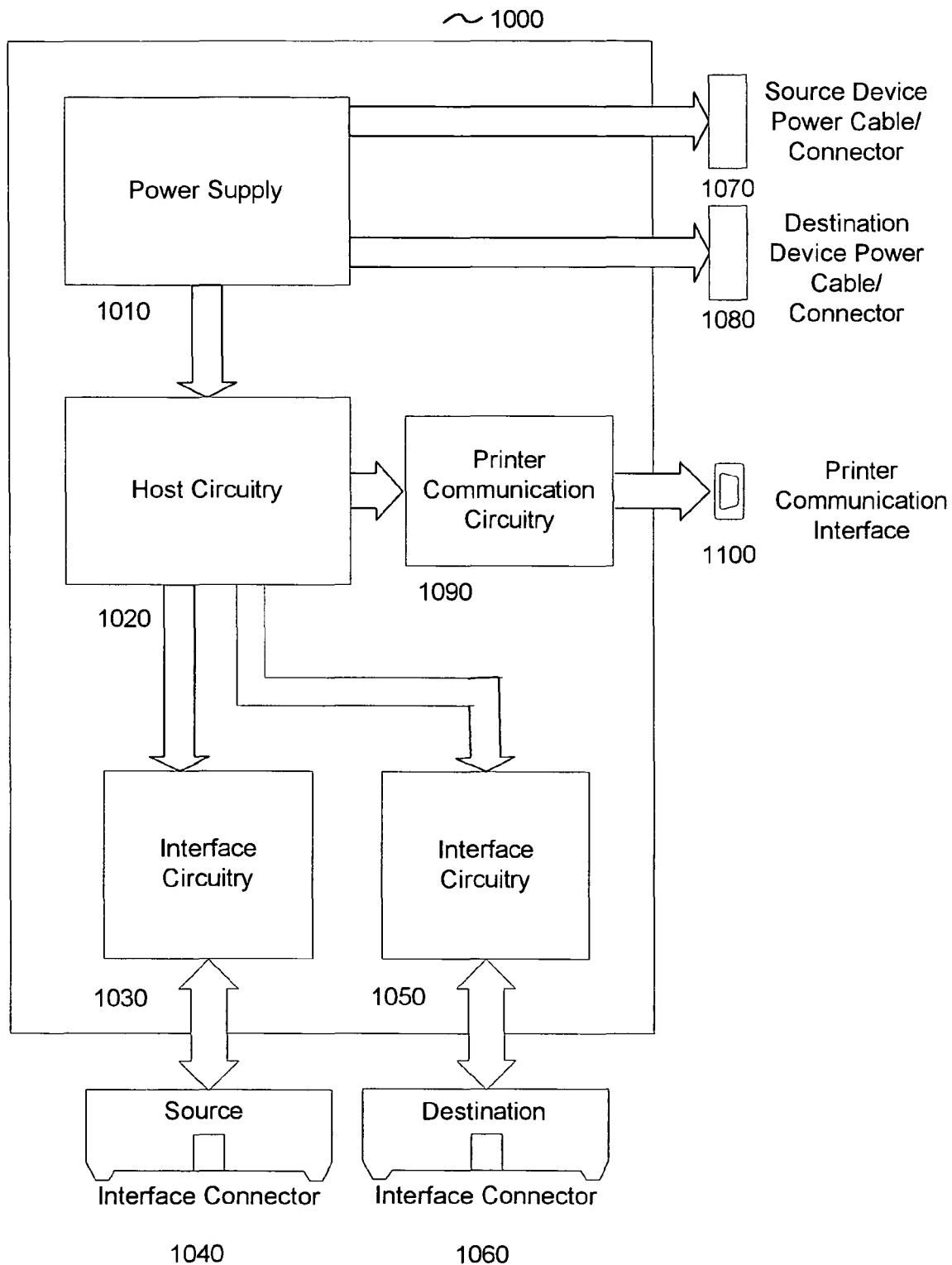


Figure 1



U.S. Patent

Jan. 2, 2007

Sheet 2 of 7

US 7,159,086 B2

Figure 2

Printout Example

```
-----  
  
Source Drive  
  
Model: WDC1103  
S/N: WD-1524305  
Firmware: 2.00.34C  
Size: 45.2 Gigs  
  
Destination Drive  
  
Model: WDC1406  
S/N: WD-3258741  
Firmware: 2.10.21  
Size: 59.6 Gigs  
  
-----  
  
Copy Results  
  
Bad Sector: 1,324,544  
Bad Sector: 6,564,974  
  
---Copy Complete---
```

Figure 3
Drive Copy Logic

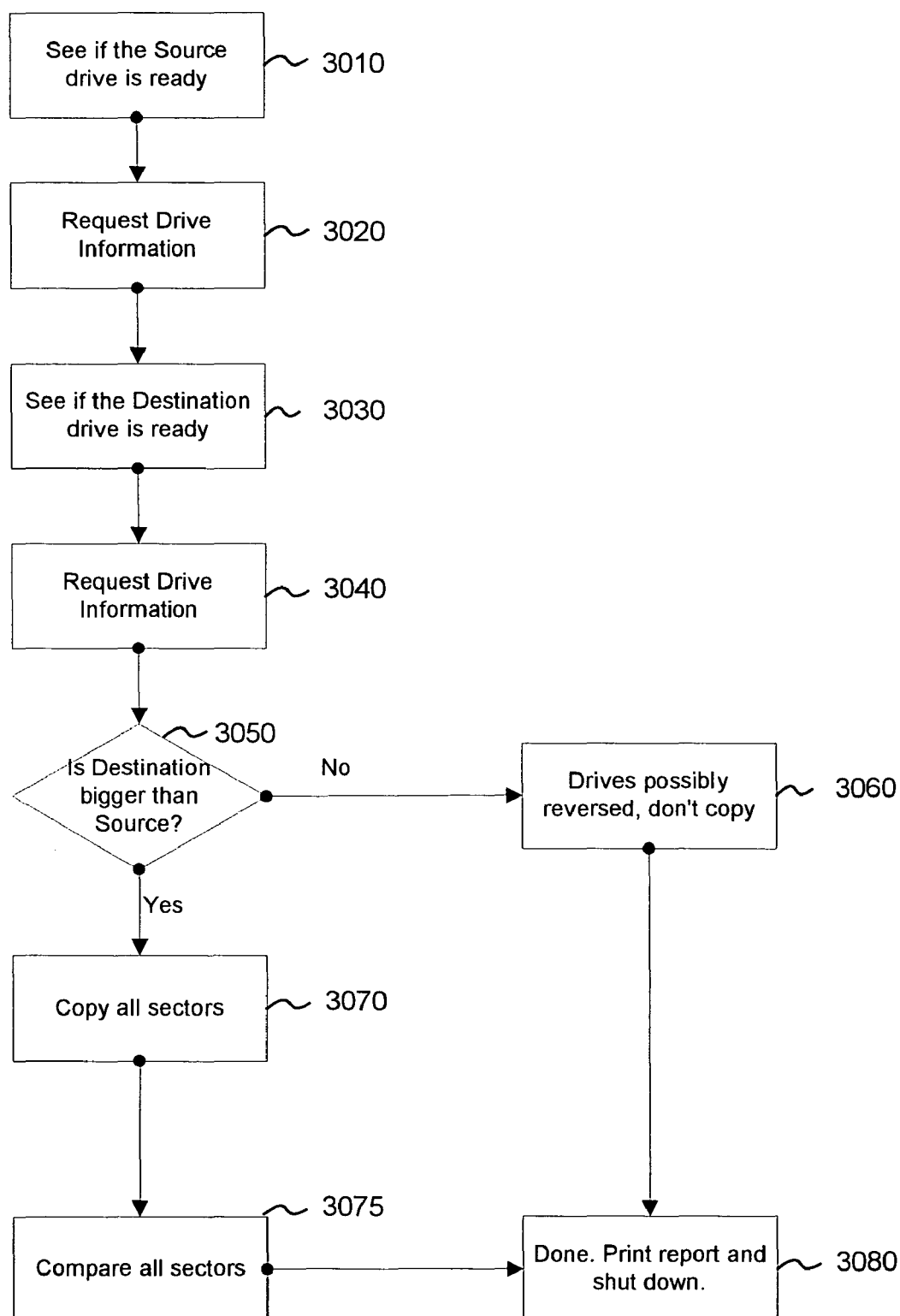


Figure 4

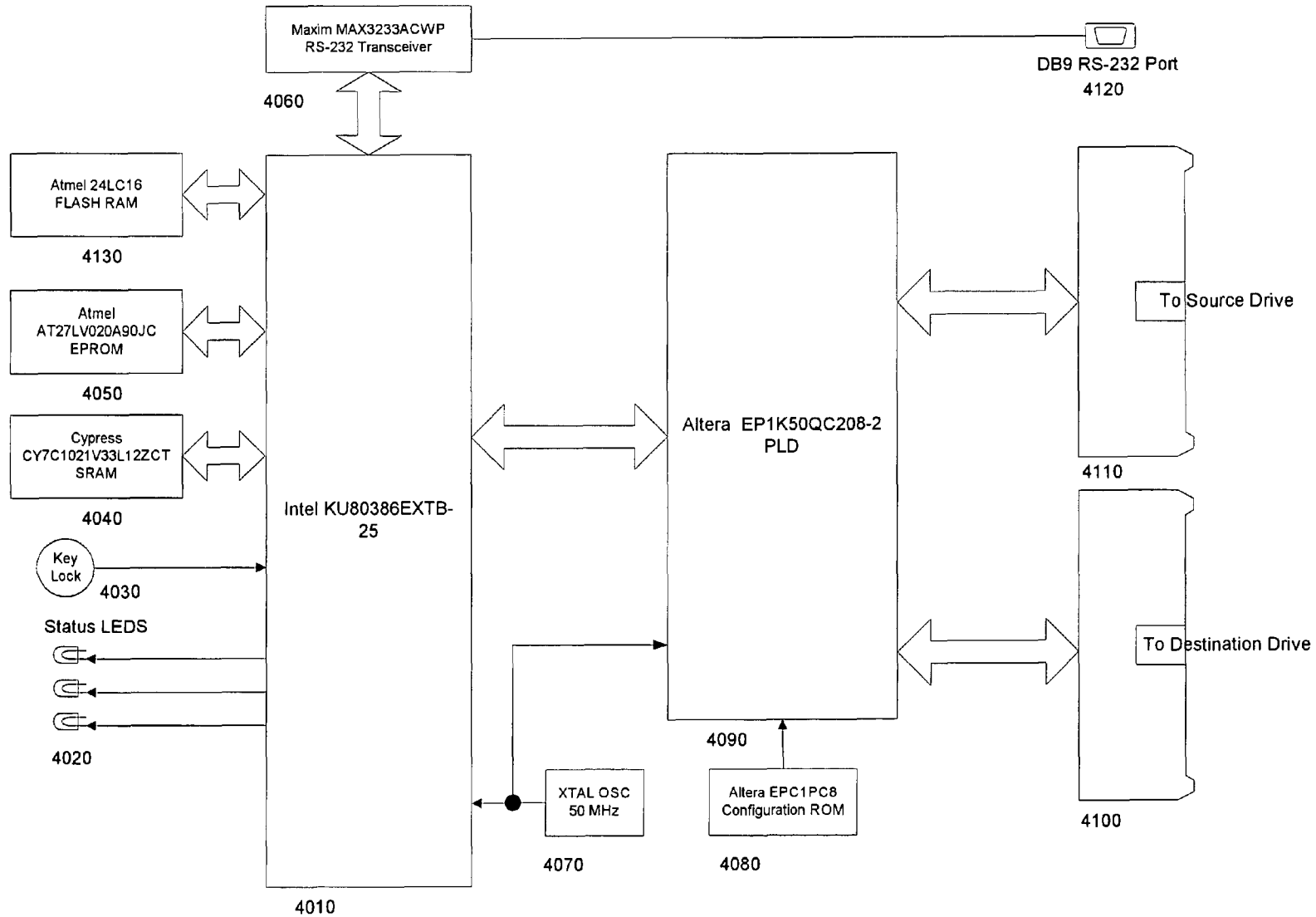


Figure 5

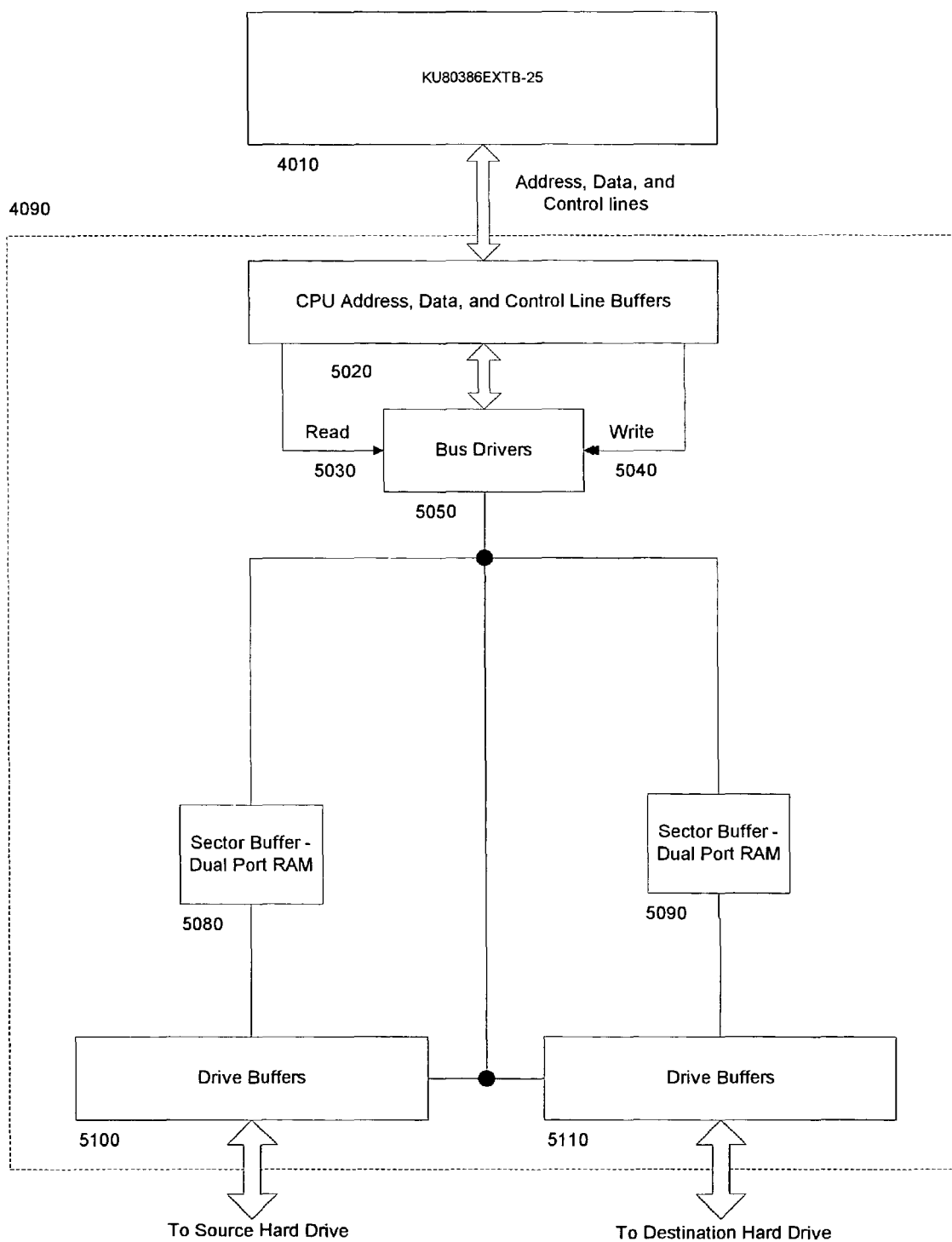


Figure 6

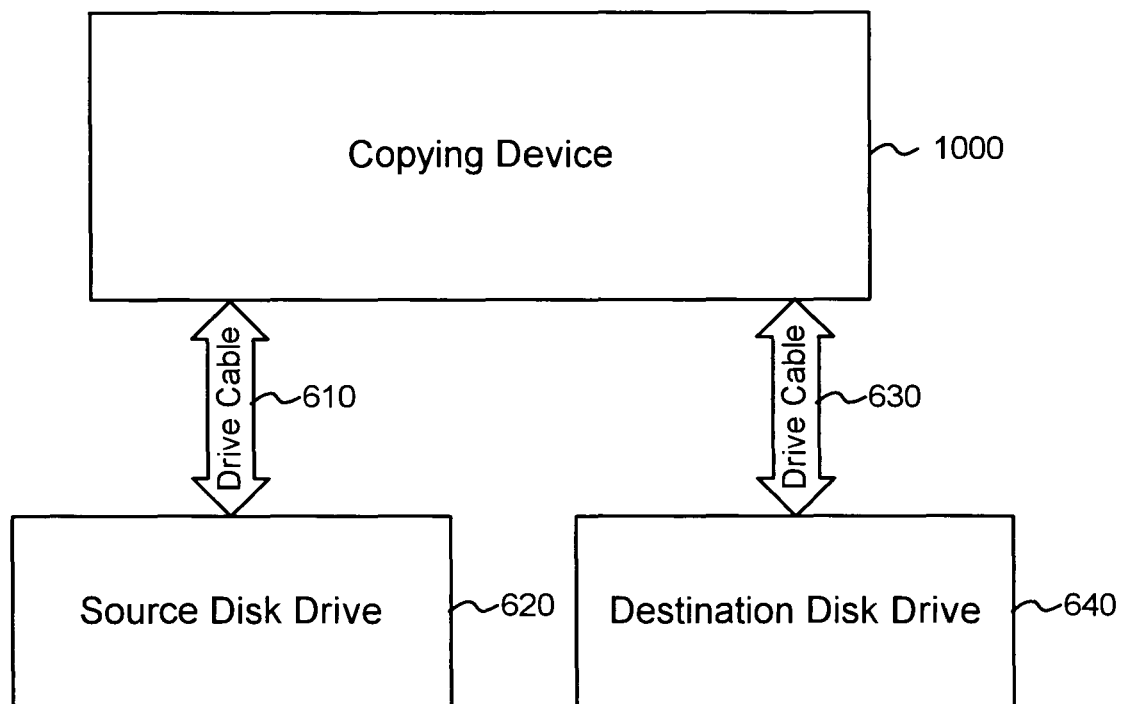
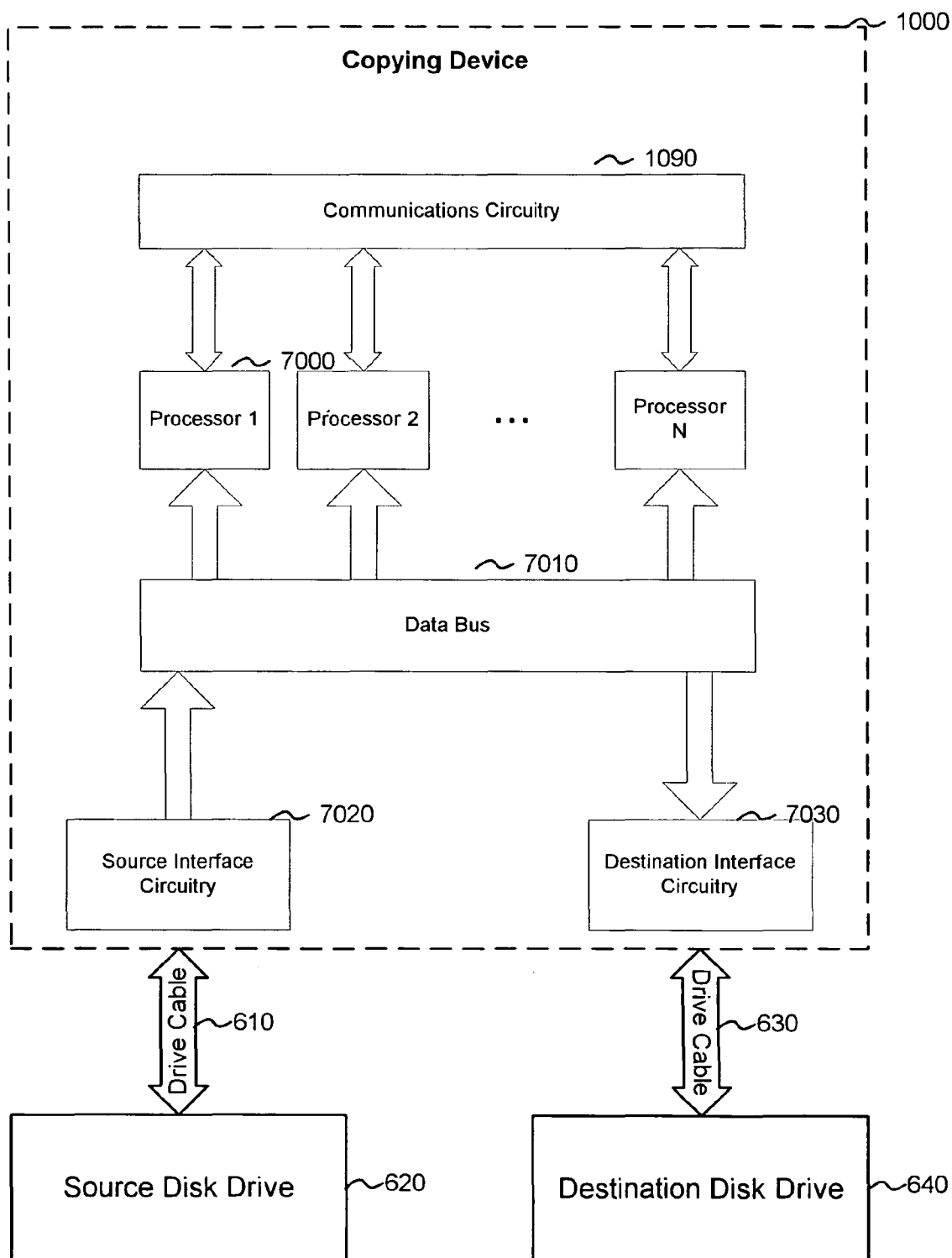


Figure 7



US 7,159,086 B2

1

SYSTEMS AND METHODS FOR CREATING EXACT COPIES OF COMPUTER LONG-TERM STORAGE DEVICES

RELATED APPLICATION

This application claims priority under 35 U.S.C. § 119 based on U.S. Provisional Application No. 60/443,387, filed Jan. 29, 2003, the disclosure of which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

A. Field of the Invention

The present invention relates to computer memory devices and, more specifically, to mechanisms for making exact copies of these devices.

B. Description of Related Art

There are many situations in which it is desirable to make exact copies of long-term memory storage device, such as computer hard drives and flash memory. For example, law enforcement officials have occasion to confiscate long-term memory storage devices. Once confiscated, the law enforcement officials need to be able to examine the storage device without changing the storage state of the device. In addition, other court officials such as defense attorneys have a need to examine confiscated long-term memory storage devices. These examinations are typically performed on an exact copy of a long-term memory storage device, so there is no danger of changing the storage state of the original device.

An example of another situation in which it is desirable to make exact copies of long-term memory storage device is in the area of computer security. Corporate Security and Corporate IT personnel are frequently tasked with making exact copies of long-term memory storage devices. Typically this involves a wide variety of devices running under an assortment of operating systems.

There are a number of known conventional techniques for making exact copies of long-term memory storage devices. One class of early techniques revolved around the concept of simply using software on a PC. These techniques are neither safe, nor easy to implement. To make a software copy, an operator must know technical information about the drive to be copied. Secondly the computer that is to be used to make a copy must be configured correctly. Thirdly, the simple act of running a drive under an operating system, such as Microsoft Windows® may change the source drive.

A second class of techniques for making exact copies of long-term memory storage devices involves dedicated stand-alone devices. A company named Logicube produces such a device. This device has numerous operating modes and options that must be specified before making a copy. Options are selected through the use of a number of buttons and a small display. One of the options is to delete the contents of what will be the destination drive. This device requires a trained operator and it has enough options to cause confusion or errors on the part of its user.

Current stand-alone devices have several limitations. They do not copy hidden areas (HPA and DCO) automatically. They do not restore hidden areas automatically. They do not perform a read and compare verification after making a copy. They do not set the size of the copied drive to be the same as the original. They do not comply fully with TWGEDE/NIJ guidelines.

Accordingly, there is a need in the art for an improved mechanism for making exact copies of long-term memory storage devices.

2

SUMMARY OF THE INVENTION

Systems and methods consistent with the present invention address these and other needs by providing for an operating system independent copying device that is physically connected to a source storage device and one or more destination storage devices.

One aspect of the invention is directed to a copying device including a plurality of elements. Specifically, the copying device includes an interface emulator configured to emulate an interface presented by a storage device (source storage device) and one or more interfaces for connecting to storage devices (destination storage device). Additionally, the copying device includes control circuitry coupled to the interface emulator and the interface(s). A user controllable switch, when actuated by a user, causes the control circuit to perform the copying procedure. The control circuitry makes an exact copy of a source device and subsequently verifies the copy was performed correctly. The copying device is operating system independent.

A method consistent with aspects of the invention includes connecting a power supply to a copying device, connecting an interface cable associated with the copying device to a long-term memory component (source) in a computer and powering up the computer. The method further includes connecting an interface cable and a power cable associated with the copying device to another long-term memory component (destination). The method further includes activating the copying device via a switch attached to the copying device, making an exact copy, and signaling completion of the copy process.

Another method consistent with aspects of the invention includes connecting a power supply to a copying device, connecting an interface cable and power cable associated with the copying device to a long-term memory component (source). The method further includes connecting an interface cable and a power cable associated with the copying device to another long-term memory component (destination). The method further includes activating the copying device via a switch attached to the copying device, making an exact copy, and signaling completion of the copy process.

Yet another aspect of the invention is directed to a device for making an exact copy of a long-term storage device. The device includes LEDs configured to provide feedback relating to an operational status of the device to a user, a user settable switch and interfaces and power cables for connecting to long-term memory components. The device further includes control circuitry coupled to the LEDs, the user settable switch and the interfaces, the control circuitry configured, when the switch is actuated by the user to make an exact copy. The control circuitry is enclosed in a portable casing and the LEDs, the user controllable switch, and the interface are mounted on an external portion of the casing.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate the invention and, together with the description, explain the invention. In the drawings,

FIG. 1 is an illustration of a functional overview of our invention.

FIG. 2 is a sample printout from our invention.

FIG. 3 is an illustration of the logic flow of our invention.

FIG. 4 is an illustration showing major electronic components and their connections, for the preferred embodiment

US 7,159,086 B2

3

FIG. 5 is an illustration of the functions of a program-
mable logic device (PLD).

FIG. 6 is a block diagram of our invention

FIG. 7 is an illustration of our invention using multiple
processors

4

DETAILED DESCRIPTION

In the following detailed description, numerous specific
details are set forth in order to provide a thorough under-
standing of the present invention. However, it will be
understood by those skilled in the art that the present
invention may be practiced without these specific details.
The use of specific electronic components is disclosed to
provide a thorough understanding of the preferred embod-
iment. However, one skilled in the art will appreciate that
there are many electronic components with similar function-
ality, and the present invention is not limited to the compo-
nents disclosed. In other instances, well known methods,
procedures, components, and circuits have not been
described in detail so as not to obscure aspects of the present
invention.

For the sake of clarity, this detailed description of the
preferred embodiment will describe an Integrated Drive
Electronics (IDE) hard drive. One skilled in the art would
appreciate that the method and system described here can be
applied to other interfaces and other long-term storage
devices.

COPYING DEVICE

Our device is able to make an exact copy of a long-term
memory storage device on another long-term memory stor-
age device of the same capacity or larger.

Our invention **1000** is a physical device that connects to
two long-term storage devices such as hard drives **620** and
640. The long-term storage devices are connected to our
device through cables **610** and **630**. One of the long-term
storage devices is designated as a Source device **620** and the
other is designated as a Destination device **640**. Our device
creates an exact copy of the data on long-term storage device
(source) **620** on long-term storage device (destination) **640**.

In a standard configuration, our device is connected to two
standard IDE hard drives. Please refer to FIG. 1. A drive
containing the Source data is connected through a standard
cable to Interface Connector **1040**. Power for this drive is
provided through an industry standard drive power con-
nector **1070**. A drive that will receive the copy of the source
data, or Destination Drive, is connected through a standard
IDE cable to Interface Connector **1060**. Power for this drive
is provided through an industry standard drive power con-
nector **1080**.

To a drive, our device appears to be a host computer. The
source and destination drives are electrically isolated from
each other through the use of separate Interface Circuitry for
each drive, **1030** for the Source, and **1050** for the Destina-
tion. In a standard PC, two IDE drives can share the same
interface circuitry. This has three disadvantages that are
overcome by our invention.

The first is that with two drives using the same interface
circuit, there is only half of the bandwidth available in the
circuit for each drive. When making a copy of one drive to
the other on the same circuit, the speed of the copy will be
no more than $\frac{1}{2}$ of the maximum bandwidth of the interface
circuit, as the data must be read from one drive, then written
to the other.

The second major problem with two drives using the same
interface circuit is that it increases the chances that the
source drive will inadvertently have data written to it. In
order to read or write from an IDE drive, a command must
be issued to the drive. Both drives receive the command, but
only one is supposed to operate on it. A single data bit in one
of the drive's command registers determines which drive
should respond to the command. This means that should a
source drive misinterpret this bit when a write command was
being issued to the destination drive, it would be possible to
write data to the source drive. The likelihood of this hap-
pening increases if there is a mismatch in the performance
capabilities of the Source and Destination drives.

The last problem is that if two IDE drives share the same
interface circuit, they must be configured by the user cor-
rectly, typically by using small jumper blocks, so that one
drive is a master and the other a slave. This is a procedure
that is fairly simple to a trained operator, assuming that
proper documentation is handy for the drive. However, since
improper jumper settings may cause one or both drives to
temporarily malfunction, it is best to get the settings correct.
By having separate Interface Circuitry for each drive,
jumper settings will not cause a drive to malfunction. In
addition, our device may detect the current drive settings and
communicate with them properly.

One advantage that our invention has over simply trying
to make a copy of a drive using a computer is that our device
is operating system independent. It does not care what kind
of data is on the drive, it simply makes a copy of a drive.
This allows it to make a perfect copy of a drive regardless
of the type of system that the drive was previously in.

Please refer to FIG. 6. Our invention **1000** is a physical
device that connects to two long-term storage devices such
as hard drives (FIG. 1.) Our device is connected to a Source
storage device through a standard cable **610** and to a
Destination storage device through a standard cable **630**. To
each drive, our device appears to be a host computer.

In order to meet the goal of creating a perfect copy of a
Source drive on a Destination drive, all of the data that
resides on the Source must be copied. This would not seem
to be a problem, but drives have functions that allow for data
to be hidden. In order to make a copy, the hidden data must
be made available to our device. There are two primary
methods for hiding data on an IDE hard drive. One is by
setting a Host Protected Area (HPA) and the other is by using
the Device Configuration Overlay function.

Both of these methods hide data by causing the drive to
report its size as smaller than it truly is. For instance, a 40
gigabyte drive could be instructed to report its size as 30
gigabytes. From that point on, any computer would see it as
a 30 gigabyte drive. Any data stored in the area between 30
gigabytes and 40 gigabytes would be effectively hidden
from the system.

If the data is hidden using the HPA method, the drive may
be instructed to make the hidden area accessible temporarily.
When a temporary change command is issued, the drive
resorts to its previous state the next time that the drive is
powered off and on again. If, however, the DCO method is
used, there is no such temporary change command. Making
a change to DCO settings is permanent until changed again.

This has the potential to cause a problem, especially when
working with sensitive information, such as a police evi-
dence drive. Any permanent change to the drive has the
potential to make the drive unreadable. Under normal cir-
cumstances, our device would make the change, copy the
data, and restore the DCO to its original configuration. This
is fine in theory, but doesn't take into account events such an

US 7,159,086 B2

5

unforeseen power failure. Should the power fail at a point in time when the DCO has been set away from its original settings, potentially the drive could be left in an unreadable state.

It is still vitally important to be able to copy a Source drive, even with the risk of a change to the drive. Our invention uses the method as described in the U.S. Provisional Patent No. 60/443,388 entitled "Systems and Methods for Restoring Critical Data to Computer Long-Term Memory Device Controllers" to provide a method for restoring a drive to its original configuration. This keeps the risk down to an acceptable level.

Although our device automatically copies and then verifies the copy, there are situations where a user may be interested in just performing the copy operation. For this eventuality, one embodiment of our device allows a user to cancel the verification process through a user control. This control only affects the verification process, and has no effect during the copy process.

In most cases, a Destination drive will be larger than a Source drive. One of the objectives of our invention is to make a copy indistinguishable from a original. If both drives were the same size, and especially if they were the same make and model, this would not be too difficult. Once all of the sectors were copied, the drives would be identical. Having a larger Destination drive causes a problem simply by virtue of it being larger. If a comparison were done of the two drives, there would be a mismatch starting at the first sector beyond the end of the Source drive. This can lead to much confusion as seen in the case: "United States of America vs Zacarias Moussaoui" <http://notablecases.vaed.uscourts.gov/1:01-cr-00455/docs/68092/0.pdf>

Our invention provides a solution to this problem by taking advantage of the ability of a drive to hide data. Using either the industry standard HPA method or the DCO method of hiding data, our device can make the Destination drive appear to be exactly the same size as the host. As such any comparison of the two drives will show them to have identical data.

The DCO function is very powerful. In addition to changing the size that the drive reports itself to be, it can also be used to change other drive parameters, such as the speed at which the drive can operate. Should the need arise, our device can use this function to more accurately match the reported capabilities of a Destination drive to a Source.

OPERATION OF THE COPYING DEVICE

Source drive interface **1030** allows connection to a drive through Interface Connector **1040**. Similarly, Destination drive interface **1050** allows connection to a drive through Interface Connector **1060**. Industry standard cables connect our device to the Source drive and the Destination drive. Our device, including cables, is physically designed to encourage an untrained user to connect it properly. When Drive Cable **610** is connected to Source Drive **620** and Drive Cable **630** is connected to Destination Drive **640**, our device is ready to operate.

A DETAILED LOOK AT LOGIC FLOW

Please refer to FIG. 3, which details the logic flow in a preferred embodiment. On power up, our device detects the presence of a Source Drive and waits for it to become ready for operations **3010**. Once the Source drive is determined to be ready, our device requests information about the drive **3020**, such as its size and capabilities.

6

In a similar manner, our device detects the presence of a Destination Drive and waits for it to become ready for operations **3030**. Once the Destination drive is determined to be ready, our device requests information about the drive **3040**, such as its size and capabilities.

A comparison is made **3050** to determine if the Destination drive is big enough to hold all of the contents of the Source drive. If the Destination drive is too small **3060**, there is the possibility that the Source and Destination drives have been reversed. The copy procedure is aborted and the user notified **3080** through status indicators and alternatively a printout.

If the Destination drive is the same size as the source or larger, then the copy procedure may begin. All of the sectors are then copied from the Source drive to the Destination **3070**.

Once the data has been copied to the Destination drive, a data comparison **3075** is initiated. This comparison checks every byte in every sector of the Source drive to be sure that the copy has completed with no errors. When complete, the user is notified through status indicators and possibly a printout **3080**.

A DETAILED LOOK AT AN ELECTRONIC BLOCK DIAGRAM

FIG. 5 is an illustration of a block diagram of our device. Our device uses an Intel 80386 EX embedded processor **5010** for its main control and logic function. This version of Intel's 80386 processor is optimized for embedded applications. The highly integrated design of this processor means that very little additional circuitry is necessary to create a small, dedicated computer system. Some static RAM is provided in **5040**, which is used for temporary program and data storage. **5050** is an EPROM that holds the code necessary to initialize and run the processor. Timing is generated by a 50 MHz crystal oscillator, **5070**.

The 80386EX **5010** has I/O pins available that may be used to connect additional devices. Three of these pins are used to power LED status indicators **5020** in order to provide feedback to a user. Another of these I/O pins, used as an input, may be used to connect a key lock **5030** to the processor.

A Programmable Logic Device, or PLD, is used to integrate numerous logical devices into a single chip **5090**. Configuration data for the PLD is stored in a configuration ROM **5080**. Timing for the PLD is generated by a 50 MHz crystal oscillator, **5070**. When the PLD is reset, it automatically tries to load configuration information from its ROM. When configuration is complete, this single chip **5090** can be viewed as having all of the functionality shown in FIG. 6.

FIG. 5 shows a breakdown of the functions implemented by the PLD **4090**. The address, data, and control lines from the processor **4010** are routed to the PLD. They are then buffered and latched in **4020** as necessary to reduce the electrical load on the processor and to stabilize the signal timing. Buffered read **4030** and write **4040** signals control the direction of the bus drivers shown in **4050**. These buffers help control the data flow and distribution of the address and data busses from the processor to other functions in the PLD.

Buffering and signal conditioning for the Source Drive is provided by the Drive Buffers in **5100**, making this the Drive Interface. Through the Bus drivers **5050** the processor can directly read and write to the Drive Interface. Another way that the processor and the Drive may communicate is through the Dual Ported RAM Sector Buffer **5080**. This allows the drive to write one sector's worth of data to RAM

US 7,159,086 B2

7

at high speed, while the processor performs other tasks. By allowing the operations to overlap in this fashion, the processor is not restricted to running at the speed of the drive, and is free to handle other functions until it needs the data in the Sector Buffer.

Similarly, buffering and signal conditioning for the Destination Drive is provided by the Drive Buffers in **5110**, making this the Drive Interface. Through the Bus drivers **5050** the processor can directly read and write to the Drive Interface. Another way that the processor and the Drive may communicate is through the Dual Ported RAM Sector Buffer **5090**. This allows the drive to read one sector's worth of data from RAM at high speed, while the processor performs other tasks. By allowing the operations to overlap in this fashion, the processor is not restricted to running at the speed of the drive, and is free to handle other functions until it needs the data in the Sector Buffer.

The 80386EX processor **4010** has a UART built in that may be used for serial communications. For versions of our device that require such communication capabilities, the UART is connected to an RS-232 transceiver **4060**. This part not only buffers the signals, it also generates the necessary voltages required for RS-232 communications. A standard DB9 connector **4120** allows our device to be connected to a computer using a standard DB9 male to female serial cable.

In the preferred embodiment, the goal of making this device foolproof for use by an untrained person is accomplished in a number of ways. The first is that the device is controlled by a single switch, such as Key Lock **4030**. This switch has only two choices, on and off. When switched on, the device starts operation and provides any required feedback to the user through one or more indicators, such as LEDs **4020**. Under normal circumstances, the only indicator that the user need be concerned with is the "Operation Complete" indicator. Additional indicators may be available for such error conditions as "Destination Drive Too Small" or "Copy in Progress." These indicators may be as simple as LEDs.

For additional detailed status, a printer may be connected to the device through communication port **4120**. Information sent to the printer may include Drive identification information to uniquely identify the drive being copied. Should any errors occur, such as a bad sector on the Source disk, the sector number may be printed.

Should an unreadable sector be identified on the Source drive, something still must be written to the Destination drive. In the case of this type of error, our device writes a standard, predefined bit pattern to the corresponding area on the Destination drive.

ADDITIONAL FEATURES

When our device is used for law enforcement forensics work, a detailed analysis is typically performed on the Destination drive after the copy has been completed. This eliminates the need to perform any work on the Source drive, which is typically an evidence drive.

An additional embodiment of our device helps to speed up the analysis process. As can be seen from the earlier technical description, at some point during the copy operation, all of the data from the Source drive passes through PLD **4090**. The data paths described allow for the Processor **4010** to have access to the data. With this ability, one skilled in the art could see how the processor would have the ability to perform an analysis of the data during the copy process.

For this embodiment, the user would communicate with our device through the RS-232 data port **4120**. The bit

8

pattern being sought would be sent to our device. As the data is transferred from the Source drive to the Destination, the processor scans the data, looking for the specified bit pattern. Should the bit pattern be found, our device can report it to an attached printer or PC.

Ideally, the processor should be fast enough to scan for a specific bit pattern without causing a slowdown in the copy process. There are cases where it is desirable to scan for more than one specific bit pattern. One solution is to simply select a faster processor. This is a reasonable solution up to a point, but depending on how many bit patterns must be scanned during the copy operation, it may not be practical to select such a processor. Assuming that such a processor was available, its price could easily prohibit its use.

A more cost effective solution than just trying to use one incredibly fast processor is to use multiple lower performance, less expensive processors as shown in FIG. 7. Each processor would be able to scan for one or more bit patterns, up to their performance limits. In this case, as data is transferred from the Source Disk Drive **620** to Data Bus **7010**, Processors **1** through **N 7000** each have access to the data arriving on Data Bus **7010**. Each processor may scan for one or more bit patterns within the data. The desired bit patterns may be built in to our device or may be communicated to the device through Communications Circuitry **1090**. If the bit pattern is detected by one of the processors, this fact is communicated to the User or another computer through Communications Circuitry **1090**.

A way to reduce the cost and complexity of a multiprocessor system is to take advantage of recent advances in Programmable Logic Devices. Newer PLDs, such as the Stratex devices from Altera Corporation of San Jose, Calif., are large enough to have processors embedded in them. The logic to create Processors to embed in PLDs is available from many sources, including Altera. Logic for other processors may even be downloaded from the Web from such sites as OpenCores.org.

Using such off-the-shelf logic, one skilled in the art would understand the benefit of embedding the processor in the PLD. Not only does it reduce the complexity of the physical PC board, but by reducing component count, the reliability of the resulting product is increased. This is especially true of a multiprocessor system. The interconnects between the data bus and each processor are shorter and more reliable if they are all within a single chip. In addition, processor support logic, such as program memory and RAM, may also be created within the PLD. The result is a single chip with multiple processors, each having the ability to examine the same set of data.

Like many functions performed by a processor, there are multiple ways of accomplishing the same task. In this case, rather than writing code to examine each byte of data for a bit pattern, the PLD could be configured to directly test the incoming data from a Source drive against the bit pattern. When this comparison is done in the PLD, it is done at hardware speeds. This method has an advantage over a processor based scan in that it is more likely to be able to keep up with advances in data transfer speeds of long-term storage devices.

By definition, if the PLD is moving the data, it is fast enough to compare the data to a bit pattern as it moves. This may not always be the case when the PLD moves the data and a processor tries to examine it. The PLD may have to slow down so that the processors can keep up.

The PLD solution is a little more complex to implement. If the bit patterns to be scanned are known at the time of design, the device can be configured from the factory. It

US 7,159,086 B2

9

would be quite rare for this to happen. In order to accommodate varying numbers of bit patterns to scan and different scanning requirements for each customer, the PLD would have to be reconfigured on-the-fly to meet the customer requirements. This raises the complexity level of the design, yet would be transparent to the user.

While having the PLD do the bit pattern comparison directly has a lot of advantages, it is not without its drawbacks. The primary one of these is that as more bit patterns need to be scanned, the complexity of the device may have to be increased. Such an increase typically comes with a higher price tag. This brings the decision as to the preferred method for scanning to a decision that can easily be made on a price/performance basis. As technology evolves, the preferred embodiment of this invention is likely to as well.

In an additional embodiment, our invention may be able to scan a long-term storage device without copying data. In the simplest case, this would be the default operation if no Destination device is detected. For a slightly more complex version of our invention, a control would be available for the user to request the scanning function.

USER COMMUNICATION

In the implementations described above, copying device 1000 signals its operational status to a user through LEDs 4020. For example, LEDs may be used to signal that: (1) copying device 1000 is performing, (2) copying device 1000 has finished copying the source device, and (3) an error was encountered.

In alternate embodiments, copying device 1000 may include additional display devices such as a graphical display or a connection to a printer. With these output devices, additional status information such as percentage of operations complete, time until operations are complete, and information about the target device may be displayed. The printer could be used to generate a continuous written record of the operations performed on the target device, including information about the devices themselves. This written record may include, for example, type of devices, time and date of data copy, and read/write errors encountered. In a similar implementation, copying device 1000 could interact with a host computer through interface 4120 to upload status information to the host computer. In another implementation, interface 4120 could be used to receive bit patterns to scan from a host computer.

Although the copying device discussed above was primarily described as copying an IDE device, in other implementations, long-term storage devices having other interfaces, such as FireWire, USB2, or SCSI could be copied using concepts similar to those discussed herein.

In addition, the copying device has been primarily described as copying from an IDE device to an IDE device. It will be apparent to one of ordinary skill in the art that the copying device could instead or additionally have different interfaces for source and copy devices. For example, the copying device may copy from an ATA device and copy to a Firewire device.

CONCLUSION

As described above, a copying device makes exact copies of a long-term storage device, while protecting the source device from inadvertent state changes. The copying device is portable, provides easy to understand user feedback, has a simple user interface and could thus be effectively used by non-technical people.

10

The copying device has a number of advantages. It is operating system independent. The copying device can operate while the source device is still in the host computer system or when it has been removed from the host computer system. It is a stand alone device that can replace more complicated and more expensive devices or systems. Additionally, the copying device does not require that the operator have any particular knowledge of the source device; it detects the capabilities and settings of the source device and adjusts the operating parameters for optimal performance without user intervention.

Still further, the copying device copies data from all partitions on the source device, regardless of the data's format, and removes any reserved or protected data areas, so that all of the data storage areas on the target device are available for copying without requiring user intervention. Additionally, the copying device can reset the source device to its original state. The copying device provides feedback to a user that it has either performed operations correctly or has run into an error. The copying device verifies the copy was performed correctly.

Additionally, the copying device may be configured to display a report of the copying process and information about the devices involved. The copying device may be configured to accept one or more bit patterns to scan during the copying process. The copying device may be further configured to perform a scan only.

It will be apparent to one of ordinary skill in the art that the embodiments as described above may be implemented in many different forms of software, firmware, and hardware in the implementations illustrated in the figures. The actual software code or specialized control hardware used to implement aspects consistent with the present invention is not limiting of the present invention. Thus, the operation and behavior of the embodiments were described without specific reference to the specific software code, it being understood that a person of ordinary skill in the art would be able to design software and control hardware to implement the embodiments based on the description herein.

The foregoing description of preferred embodiments of the present invention provides illustration and description, but is not intended to be exhaustive or to limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practice of the invention.

No element, act, or instruction used in the description of the present application should be construed as critical or essential to the invention unless explicitly described as such. Also, as used herein, the article "a" is intended to include one or more items. Where only one item is intended, the term "one" or similar language is used.

The scope of the invention is defined by the claims and their equivalents.

What is claimed:

1. A stand-alone, dedicated function device for making exact copies of long-term memory devices comprising:
 - an interface for connecting to a storage device (source);
 - one or more interfaces for connecting to the storage device(s) (destination);
 - wherein the interfaces are electronically isolated from each other through the use of separate interface circuitry for each interface;
 - a user controllable switch that, when actuated by a user, causes the device to commence a copy;
 - and a control circuit coupled to the interface (source) and the interface(s) (destination), the control circuit issuing commands to: compare the size of the source device to

US 7,159,086 B2

11

the size of a destination device and communicate result to a user, open hidden areas on the source device, make an exact copy of the storage device connected to the interface (source), read and compare data on the source and destination devices and communicate result to a user, restore the source drive to its original condition, wherein the copying device is operating system independent.

2. The copying device of claim 1, wherein a user controllable switch is connected to the control circuit to interrupt the verification procedure.

3. The copying device of claim 1, wherein the interface is an integrated device electronics (IDE) interface for a disk drive.

4. The copying device of claim 1, wherein the hidden area the control circuit removes or modifies is a Host Protected Area (HPA).

5. The copying device of claim 1, wherein the hidden area the control circuit removes or modifies is Device Configuration Overlay settings (DCO).

6. The copying device of claim 1, wherein the control circuit sets the size of a copied device to the size of a source device.

7. The copying device of claim 1, further comprising: one or more additional interfaces for connecting to display and/or output devices, to produce a report.

8. The copying device of claim 1, wherein the control circuit writes a standard bit pattern on a copy device to indicate unreadable data on the source device.

9. The copying device of claim 1, further including light emitting diodes (LEDs) coupled to the control circuit and configured to transmit status information relating to the status of the copying device.

10. The copying device of claim 1, wherein the control circuit scans the source device for one or more specific bit patterns, during the copy procedure.

11. The copying device of claim 10, further including: a user controllable switch is connected to the control circuit to enable scanning functions only.

12. The copying device of claim 10, further including: one or more additional interfaces for connecting to input devices, to receive specific bit patterns to scan.

13. The copying device of claim 1, further comprising: a casing configured to contain the control circuit and the

12

interface, the user controllable switch being mounted on the casing, the casing being of a size that is portable by the user.

14. The copying device of claim 13, further comprising: cables emanating from the casing and connected to the interfaces, the cables being configured to connect to long-term memory devices.

15. The copying device of claim 14, wherein the cables are Integrated Device Electronics (IDE) cables.

16. The copying device of claim 1, further comprising: a power supply configured to supply power to the control circuit.

17. The device of claim 16, further comprising: drive power cords emanating from the casing and configured to supply power from the power supply to the long-term memory components.

18. A stand-alone, dedicated function copying device comprising:

means for interfacing with a source drive,

wherein the source device is protected from accidental state changes;

means for interfacing with one or more destination devices;

means initiating the copying procedure;

means for comparing the size of the source device to the size of a destination device and communicating result to a user;

means for opening hidden areas on the source device;

means for making an exact copy;

means for reading and comparing the data on the source and destination devices and communicating result to a user;

means for restoring the source device to its original condition,

wherein the copy device is operating system independent.

19. The copying device of 18, further comprising: means for scanning for one or more specific bit patterns.

20. The copying device of 18, further comprising: means for indicating areas on a source device that were unreadable.

21. The copying device of 1, wherein the source and destination devices have different interfaces.

* * * * *

EXHIBIT C

(12) **United States Patent**
Bress et al.

(10) **Patent No.:** **US 7,228,379 B2**
(45) **Date of Patent:** **Jun. 5, 2007**

(54) **SYSTEMS AND METHODS FOR REMOVING DATA STORED ON LONG-TERM MEMORY DEVICES**

(76) Inventors: **Steven Bress**, 17917 Wheatridge Dr., Germantown, MD (US) 20874; **Dan Bress**, 17917 Wheatridge Dr., Germantown, MD (US) 20874; **Mike Menz**, 1335 Champagne Cir., Roseville, CA (US) 95747; **Mark Joseph Menz**, 114 Rawlings Ct., Folsom, CA (US) 95630

5,630,093 A * 5/1997 Holzhammer et al. 711/115
5,721,665 A 2/1998 Schultz 361/149
5,777,811 A * 7/1998 Bodo 360/15
5,861,873 A * 1/1999 Kikinis 345/157
5,953,199 A * 9/1999 Owens 361/212
5,966,732 A * 10/1999 Assaf 711/170
5,969,933 A 10/1999 Schultz et al. 361/149
6,629,184 B1 * 9/2003 Berg et al. 710/306
6,727,894 B1 * 4/2004 Karidis et al. 345/174
2004/0078514 A1 * 4/2004 Kung et al. 711/105

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 442 days.

(21) Appl. No.: **10/174,984**

(22) Filed: **Jun. 20, 2002**

(65) **Prior Publication Data**

US 2002/0196572 A1 Dec. 26, 2002

Related U.S. Application Data

(60) Provisional application No. 60/299,783, filed on Jun. 21, 2001.

(51) **Int. Cl.**

G06F 12/06 (2006.01)

G06F 12/08 (2006.01)

(52) **U.S. Cl.** 711/112; 711/105; 711/115; 711/170

(58) **Field of Classification Search** 711/112, 711/115; 345/174, 157; 361/212; 710/306
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,717,975 A 1/1988 Ogura et al. 360/66

OTHER PUBLICATIONS

Friedhelm Schmidt, The SCSC BUS and IDE Interface, Addison-Wesley, 1995.*

<http://en.wikipedia.org/wiki/univac> univac history.*

Microsoft computer dictionary 1997.*

* cited by examiner

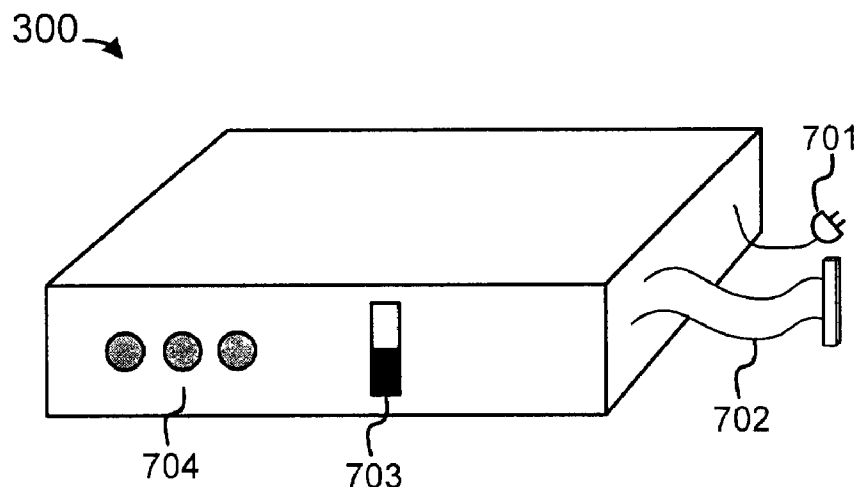
Primary Examiner—Kevin L. Ellis

Assistant Examiner—Duc T Doan

(57) **ABSTRACT**

An application specific device for erasing data from a long-term storage device includes a power supply, a control circuit, and an interface to the storage device. The control circuit controls the long-term storage device to irretrievably remove data from the storage device. The storage device may be, for example, a hard disk drive or compact flash memory. The application specific device is physically small, is operating system independent, and has simple interface that is useable by non-computer professionals.

3 Claims, 10 Drawing Sheets



U.S. Patent

Jun. 5, 2007

Sheet 1 of 10

US 7,228,379 B2

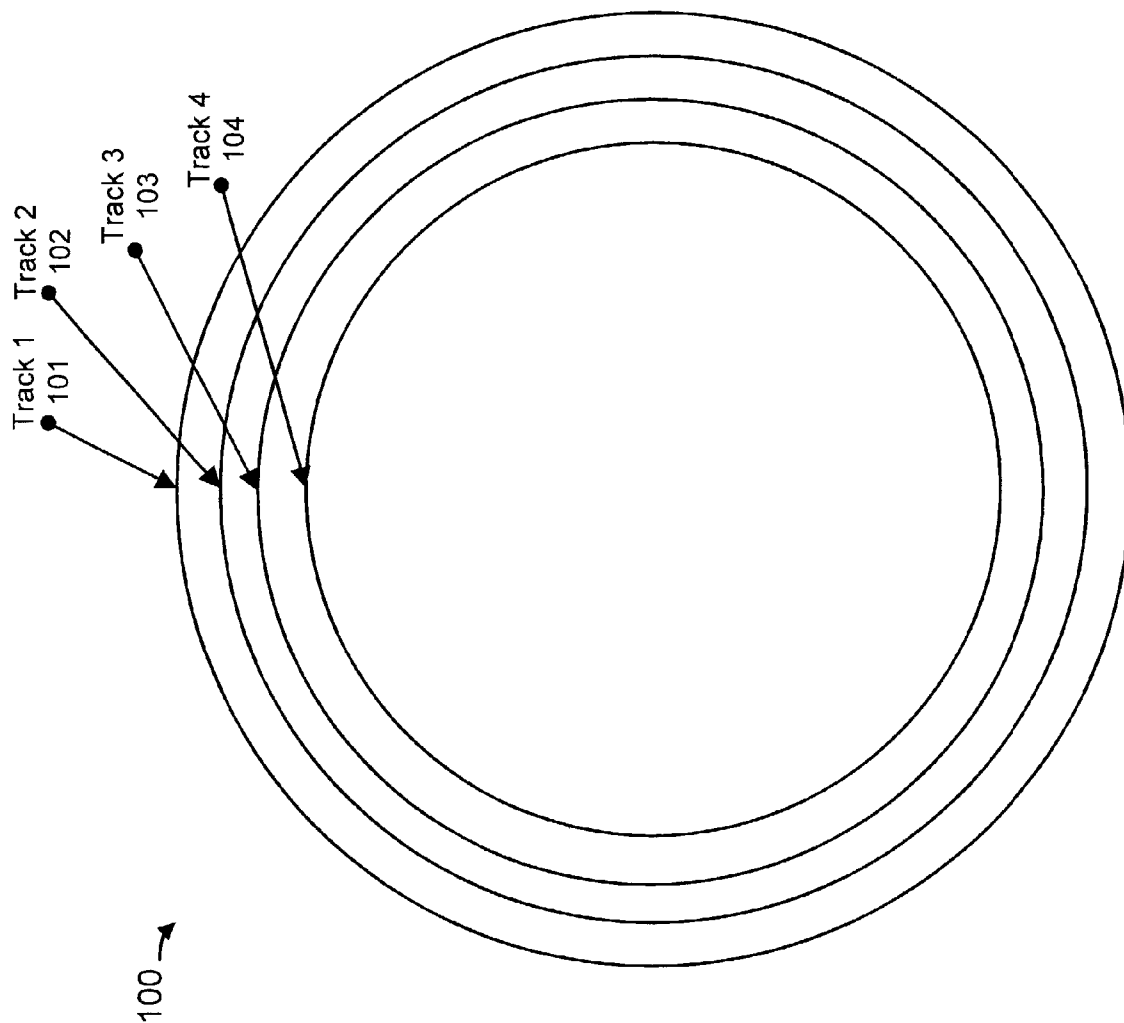


Fig. 1

U.S. Patent

Jun. 5, 2007

Sheet 2 of 10

US 7,228,379 B2

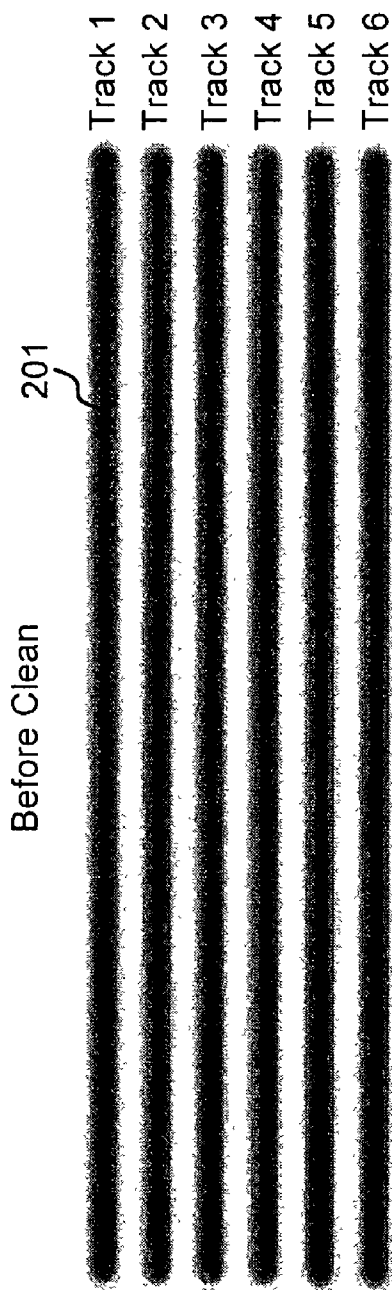


Fig. 2A

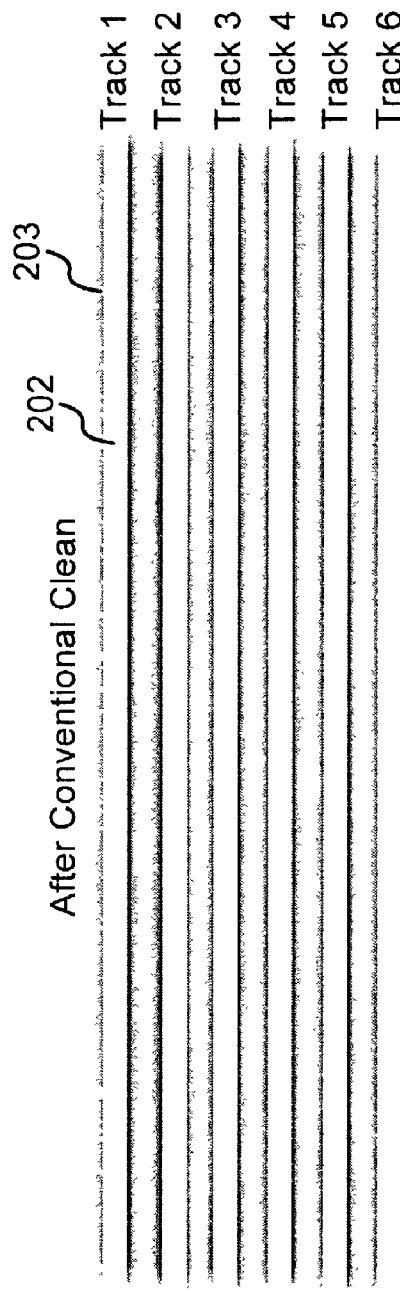
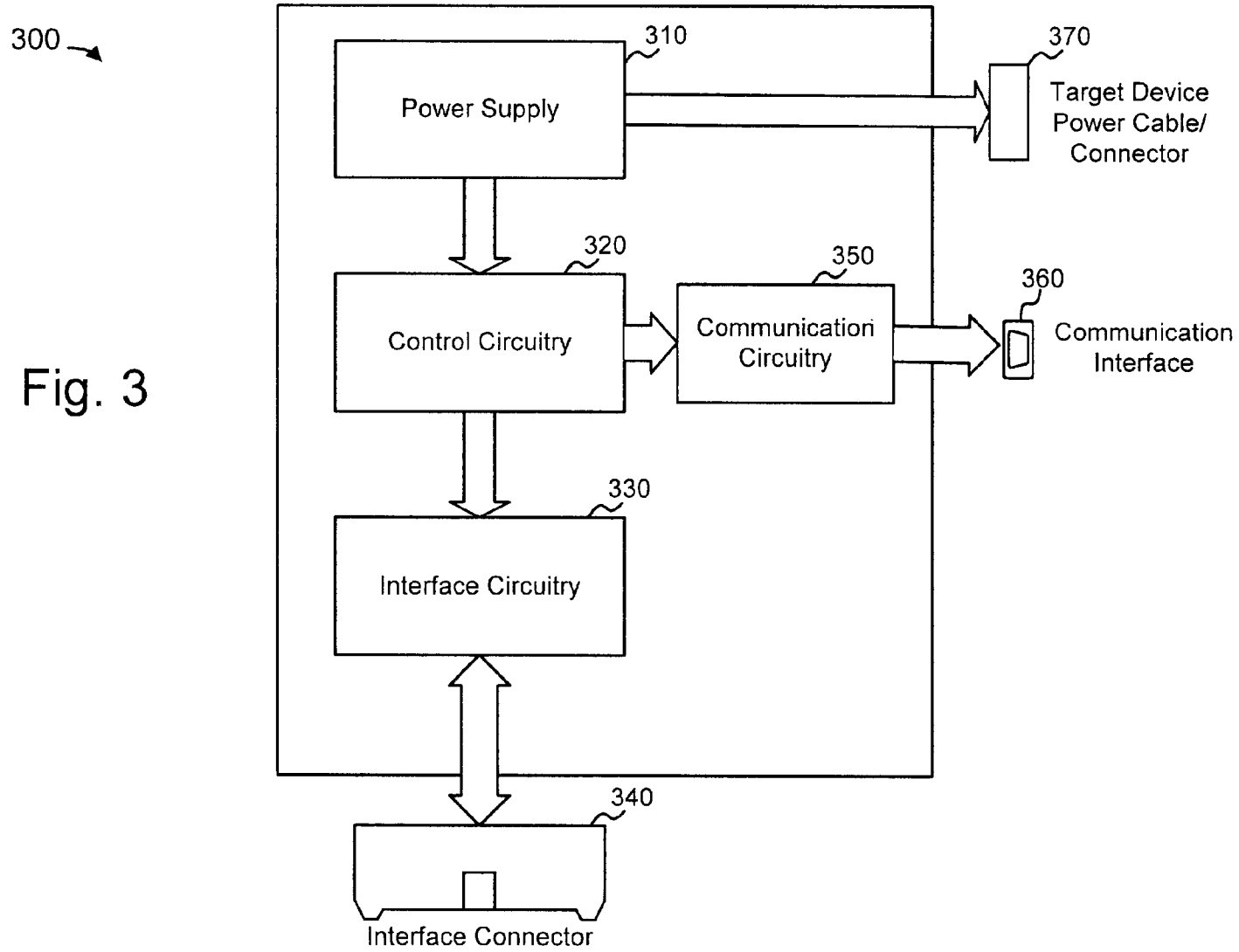


Fig. 2B



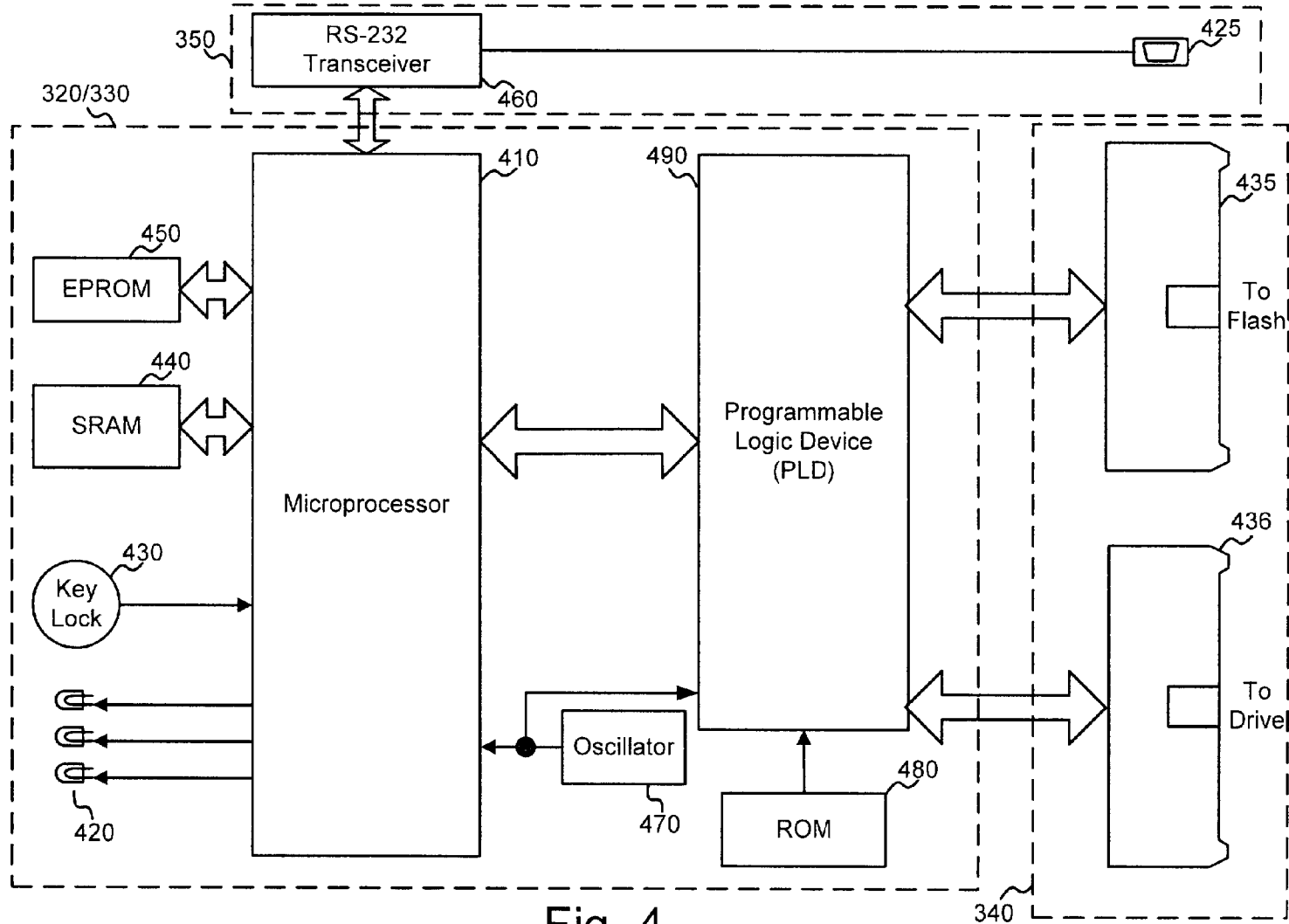


Fig. 4

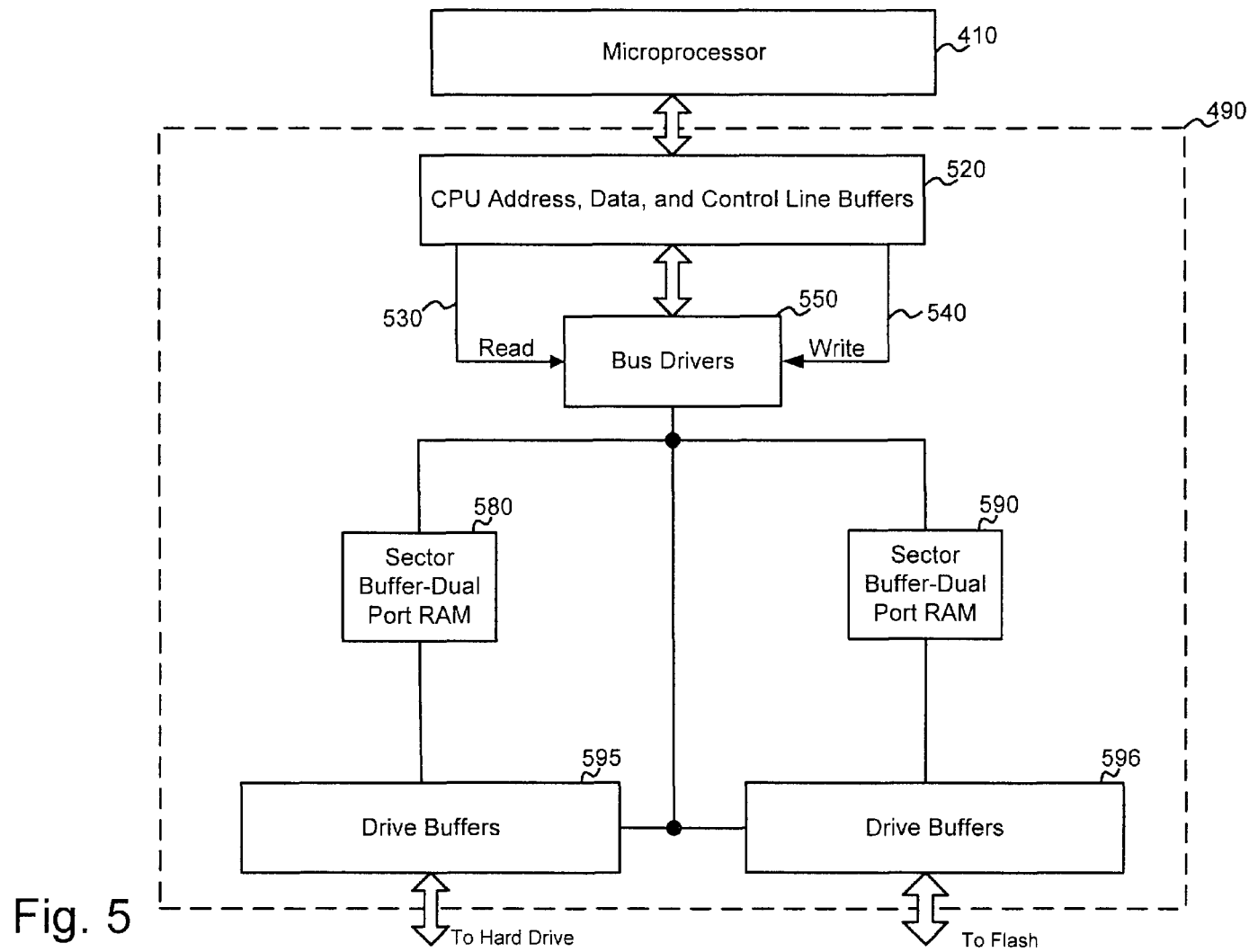
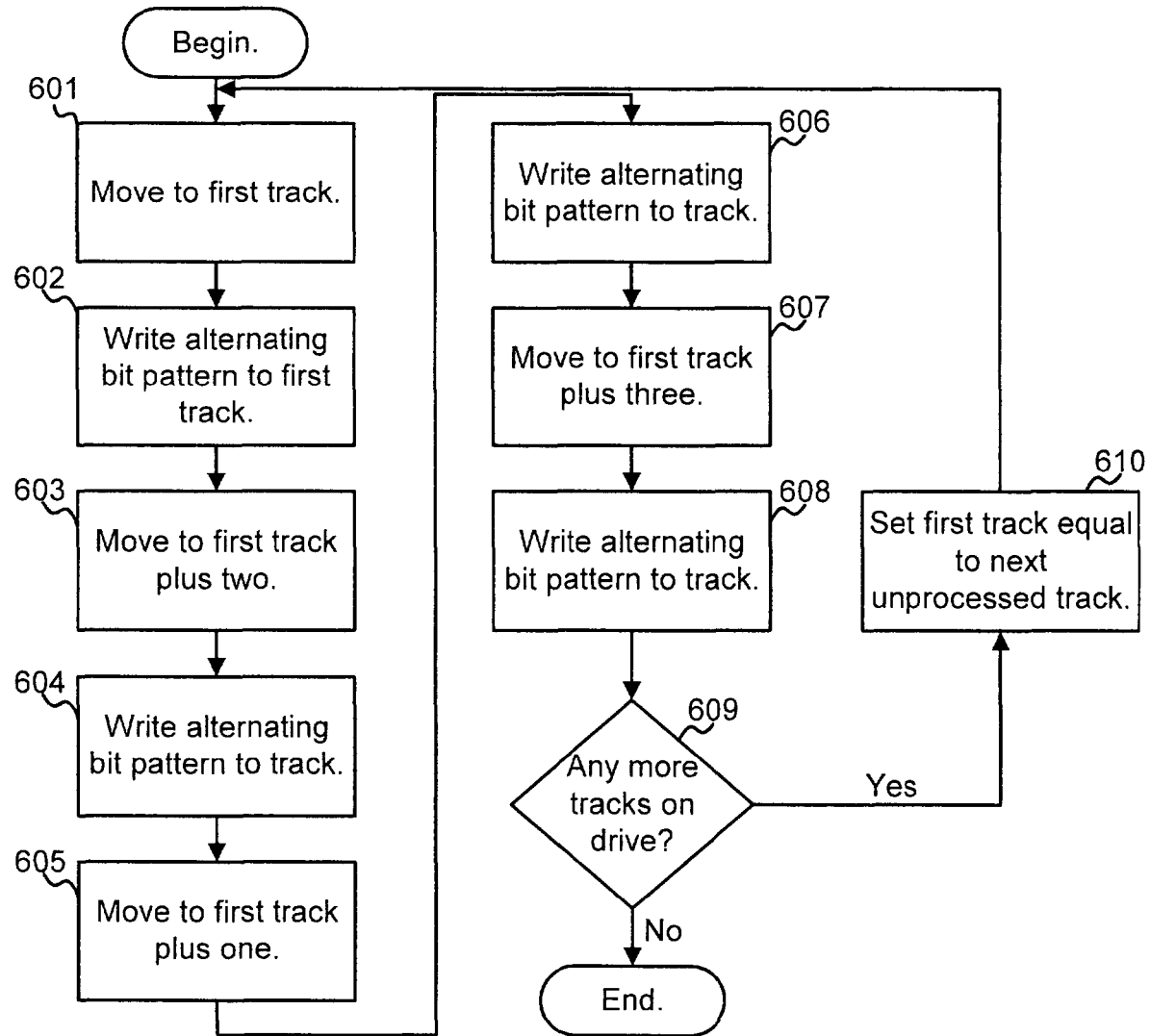


Fig. 6



U.S. Patent

Jun. 5, 2007

Sheet 7 of 10

US 7,228,379 B2

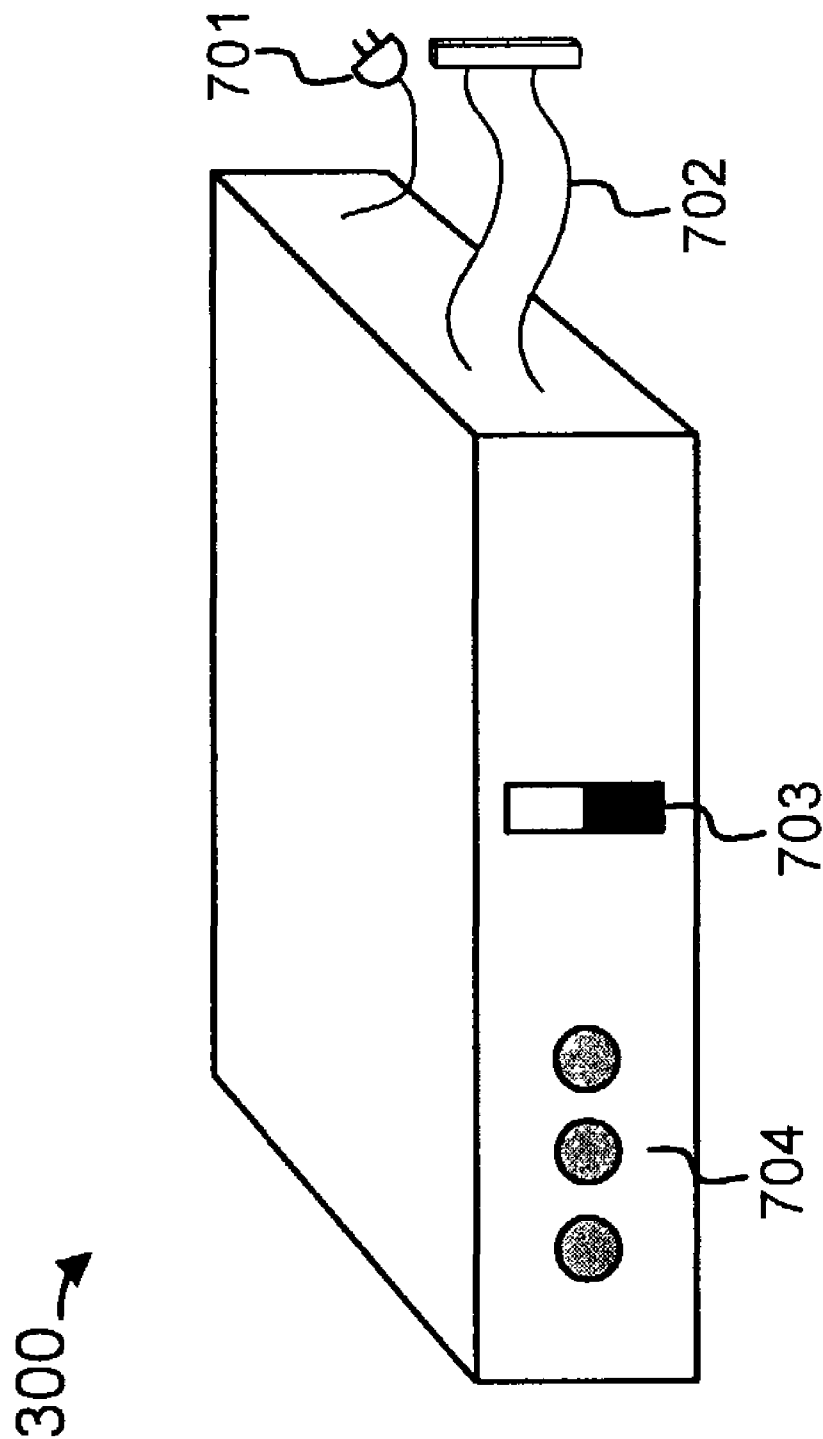


Fig. 7

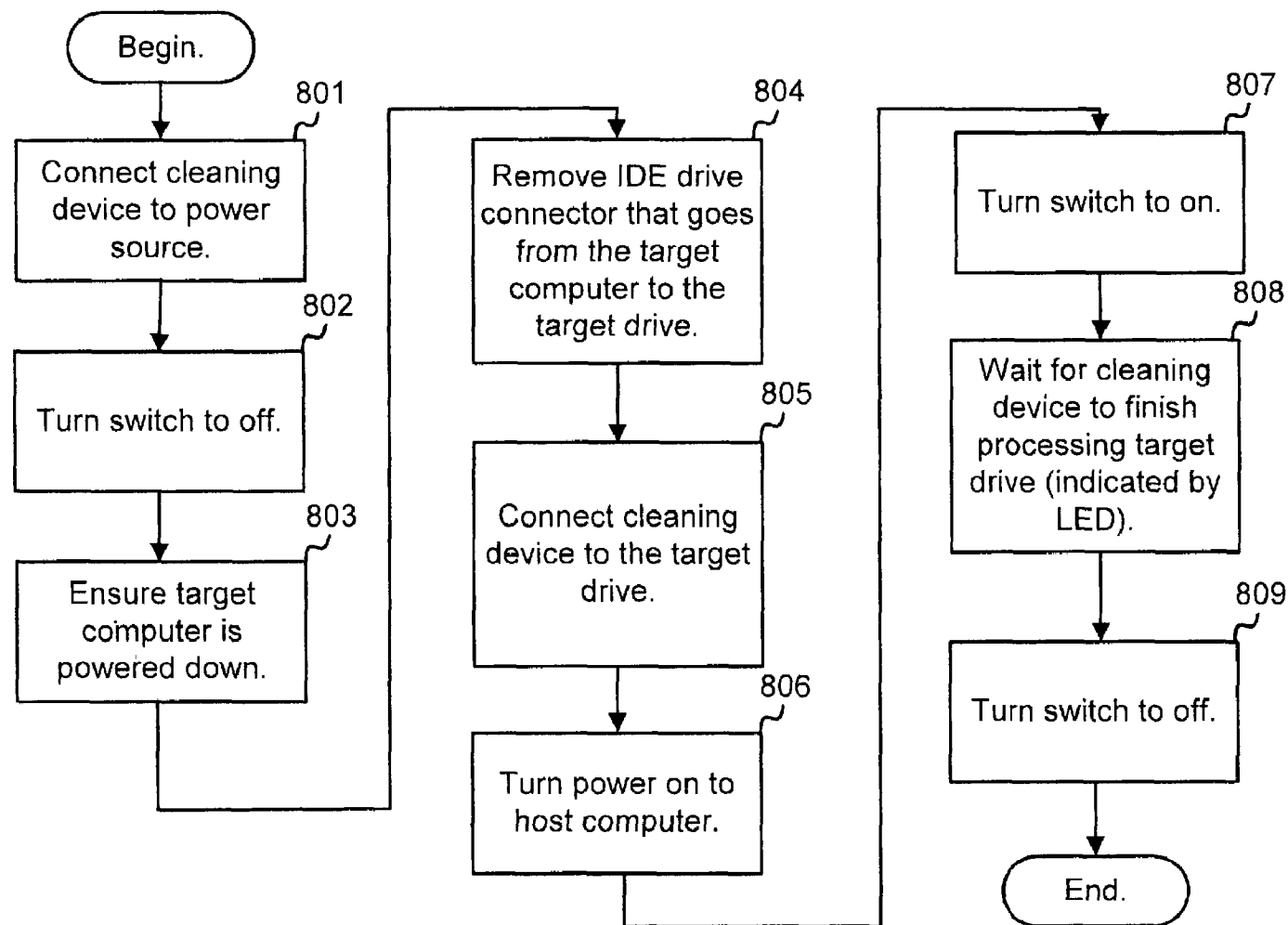


Fig. 8

U.S. Patent

Jun. 5, 2007

Sheet 9 of 10

US 7,228,379 B2

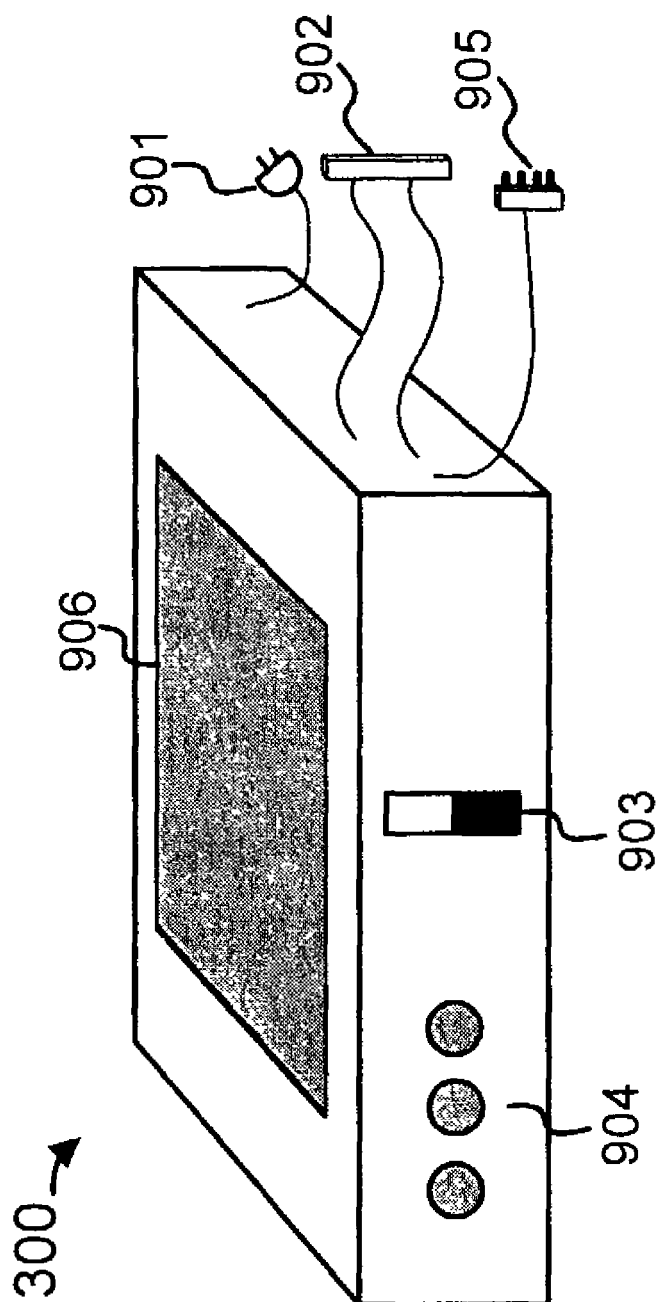


Fig. 9

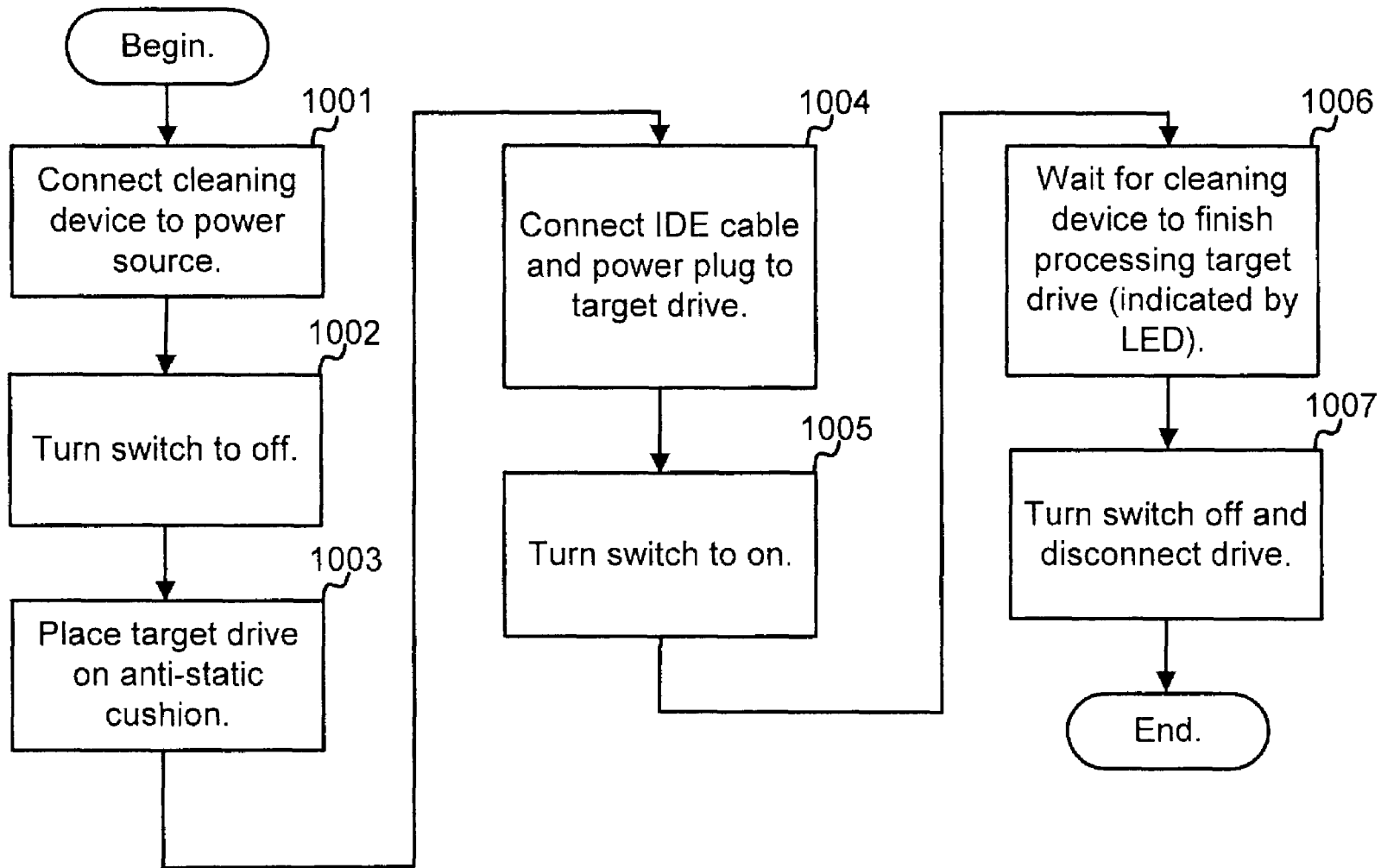


Fig. 10

US 7,228,379 B2

1

SYSTEMS AND METHODS FOR REMOVING DATA STORED ON LONG-TERM MEMORY DEVICES

RELATED APPLICATION

This application claims priority under 35 U.S.C. §119 based on U.S. Provisional Application No. 60/299,783, filed Jun. 21, 2001, the disclosure of which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

A. Field of the Invention

The present invention relates to computer memory devices and, more specifically, to mechanisms for removing data from memory devices.

B. Description of Related Art

Confidential information, such as credit card numbers, passwords, and personal account login information is often stored on computer hard drives. Confidential information stored on paper can be shredded when it is no longer required. Computer long-term memory devices, such as hard drives, however, are not as easy to destroy. Even if physically crushed with a hammer, the hard drive is likely to still have recoverable data left on its magnetic media.

Not only is it difficult to physically destroy long-term memory storage devices, these devices have intrinsic value and could be reused. For example, some users replace their long-term memory storage devices for larger and/or faster devices. A graphic artist may need a bigger/faster long-term memory storage device than say a writer. While a device may no longer have value to the graphic artist, the writer may find value in that device. Some users routinely replace their entire functioning computer, which includes one or more long-term memory storage devices. These replaced computers will often have value. However, in order to be able to securely reuse these long-term memory storage devices, it is desirable to first securely remove the stored data.

Simply deleting files on a drive does not normally remove the data from the drive media. Instead, pressing the delete key on most computers simply causes a change to be made to the FAT (File Allocation Table) that keeps track of where data is stored on the hard drive. The data itself remains on the hard drive. It is trivial to recover "deleted" data using off-the-shelf software.

One conventional software method of irretrievably deleting files is by "formatting" the media through a software format command. The format command typically rewrites a small area of the drive that contains its table of contents with a blank table of contents. It may also rewrite the file allocation table to indicate that all of the space on the disk is available. While a novice computer user would see a formatted drive as clean, one skilled in the art would have no problem recovering data or even complete files from a disk formatted in this manner.

Another method of implementing the format command consists of writing the value "0" to the entire hard drive. This is much stronger protection from a casual attempt to recover data from a drive treated in this fashion. However, even with this type of format command, data can still be recovered.

In order to assist the understanding of how data can be recovered from a hard drive that has been overwritten with zeroes, a basic description of hard drive operation will now be given. A hard drive includes three main components: (1) magnetic media, (2) media control electronics, and (3) a

2

read/write head. A hard drive is commonly conceptualized as pockets of magnetic media surrounded by null media. In practice, however, it is often impractical to manufacture hard drives in this fashion. A hard drive may consist of contiguous magnetic media, any part of which may be magnetized. The read/write head is directed by the media control electronics to move to a specific location over the magnetic media and read and/or write at that location. For each bit of data, the media control electronics reserves a location on the magnetic media that is larger than the read/write head.

The hard drive uses changing patterns of magnetism to represent digital data. For example, if the magnetic field of an area is polarized in one direction, the data is declared to be a one. If the polarization is in the other direction, the data is a zero. In the real world, the strength of the polarization may be quite different from one data area to the next.

As mentioned, polarization is over an area. The size of the area is determined by a number of factors, but the principal ones are the size of the recording head, the type of recording media, and the speed of the magnetic platter. As the magnetic platter spins under the recording head, the head traces out one track. An actuator in the hard drive moves the head in discrete steps. Each one of these steps defines another track on the drive. FIG. 1 is a diagram illustrating multiple tracks **101–104** on a platter of a hard drive **100**. The term "step time" describes the time the head takes to get from one track, such as track **101**, to a next track, such as track **102**. The term "seek time" describes the time that it takes for the head to get from any one track to any other track.

In the real world, there is no guarantee that the head will be aligned over a track in precisely the same location each time the head moves, although it will typically be close enough to the track to read and write data reliably. Reasons for this potential miss-alignment include thermal expansion and contraction of the disk platters. If the operating temperature of the drive changes, the platters may shrink or expand. The actuator that moves the head may also change its performance over time. On any given motion, there may be a slight overshoot, undershoot, or even a small oscillation in the head as it moves. Drive manufacturers may recommend reformatting drives regularly so that these changes do not cause any data loss.

Thus, a track of data on a disk drive is generally not a perfect circle with a fixed width. The track may be circular, but its width may vary greatly from the theoretically ideal. In normal operation, this is mostly irrelevant. However, when trying to irretrievably delete data by overwriting the data with "zeros," there is likely to be remnants of the old data on the edges of the track. This data may potentially be recovered.

FIGS. 2A and 2B illustrate the above-discussed concept of erased data remaining on the edges of a track. FIG. 2A illustrates a number of tracks (labeled as tracks **1–6**) on a magnetic platter in which darker areas **201** on each track represent magnetic changes that can be interpreted as data. FIG. 2B illustrates the same tracks after a conventional clean implemented by rewriting zeros over each bit. The central dark area is now a uniform white area **202** with a fuzzy area **203** to either side. Fuzzy area **203** includes residual magnetism that could be interpreted by someone with the correct equipment.

Simply deleting files on a hard disk or formatting the hard disk may fail to remove all of the data for another reason. The disk may be set up with more than one partition. When a hard disk is set up for use by a computer, the allocation of storage space must be specified. In many cases, the entire hard drive is assigned to be in a single partition. In other

US 7,228,379 B2

3

cases, there may be reason to create two or more partitions. Each partition can be used to hold data in different formats, such as one partition used for Microsoft Windows data, and another partition used for Linux data. Or, in the case of Windows, multiple partitions may be set up so that the drive appears to be a number of small drives, rather than one large drive.

This common scenario presents a couple of problems. In the case where different partitions hold different formats of data, it may not be possible to access one of the partitions for file deletion or formatting. Windows may not make the Linux partition visible, so there would be nothing to delete. Even with multiple partitions of the same data type, the data in the partitions may all be visible to the user, but it would be up to the user to know enough to delete all of the data in all of the partitions assigned to a single drive. For untrained personnel, this might be difficult to determine, so for them, the data may appear "hidden."

Software solutions exist that attempt to overcome the above problems in permanently removing data from hard drives. These solutions involve running software on a single computer to eliminate data on that computer's hard drive. These solutions are not without disadvantages. First, these solutions tend to be operating system dependent, and the operating system must be running properly. Second, at least a moderately trained computer operator must select the proper software, install the software, run the software, and then verify the results. This can be technically difficult, as the operating system files are being eliminated along with all other files. Third, these software solutions can be time consuming. For example, assume a computer professional is given seven computer systems by her company and is tasked with removing data on the drives before recycling the computer systems. In order to use this system of data removal, the computer professional would have to hook up seven monitors, seven keyboards, and seven mice to seven computers. She would then have to turn on seven computers, identify the operating system of each computer, and choose the proper software to run on each one. If there were a power interruption during this process, she may have to format the drives, reinstall the operating system, and then continue with data removal. Thus, this can be a time intensive removal method.

Other conventional software approaches to permanently removing data involve using a first computer as a data removal station for a second computer or drive. The first computer is set up to run data removal software. The user connects the drive to be "cleaned" to this computer. There are a number of potential problems with this approach. First, an expensive PC must be taken from productive service, along with the desk space needed to support its keyboard, monitor and mouse. Second, a trained user must attach the target drive to the host machine. During this process, the host machine must be shut down, the target drive properly installed and configured with a valid address or master/slave status, and finally the host machine rebooted. This can be a time consuming process requiring a trained user. Finally, there is a potential system problem. Most PCs are not designed for hard drives being swapped in an out on a regular basis. A static charge, or cable failure while installing a target drive can damage not only the target drive but also the host machine.

Accordingly, there is a need in the art to more efficiently delete data from long-term memory devices such as hard drives.

4

SUMMARY OF THE INVENTION

Systems and methods consistent with the principles of this invention provide for an easy to use and portable long-term memory cleaning device.

One aspect of the invention is directed to a device for removing data from a long-term memory component. The device includes an interface for connecting the device to the long-term memory component and a control circuit configured to control the long-term memory component through the interface to permanently remove data from the long-term memory component. A user controllable switch, when actuated by a user, causes the control circuit to commence permanently removing the data from the long-term memory component.

A method consistent with aspects of the invention includes connecting a power supply to a cleaning device, connecting a cable associated with the cleaning device to a long-term memory component in a computer, and powering-up the computer. The method further includes activating the cleaning device via a switch attached to the cleaning device, permanently removing data from the long-term memory component, and signaling completion of the data removal from the long-term memory component.

Another method consistent with aspects of the invention includes connecting a power supply to a cleaning device, connecting a cable associated with the cleaning device to a long-term memory component, and connecting a power cable attached to the cleaning device to the long-term memory component. The method further includes activating the cleaning device via a switch attached to the cleaning device, permanently removing data from the long-term memory component, and signaling completion of the data removal from the long-term memory component.

Yet another aspect of the invention is directed to a device for removing data from a long-term memory component. The device includes LEDs configured to provide feedback relating to an operational status of the device to a user, a user settable switch, and an interface for connecting to a long-term memory component. The device further includes circuitry coupled to the LEDs, the user settable switch, and the interface, the circuitry configured, when the switch is actuated by the user, to communicate with the long-term memory component through the interface and control the long-term memory component to permanently remove the data therefrom. The circuitry is enclosed in a portable casing and the LEDs, the user controllable switch, and the interface are mounted on an external portion of the casing.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate the invention and, together with the description, explain the invention. In the drawings,

FIG. 1 is a diagram illustrating tracks on a disk drive;

FIGS. 2A and 2B illustrate data stored in the magnetic media of a disk drive;

FIG. 3 is a block diagram illustrating a long-term memory cleaning device consistent with an aspect of the present invention;

FIG. 4 is a diagram illustrating portions of the cleaning device shown in FIG. 3 in additional detail;

FIG. 5 is a diagram illustrating portions of the cleaning device shown in FIGS. 3 and 4 in additional detail;

FIG. 6 is a flow chart illustrating operation of the cleaning device consistent with an aspect of the invention;

US 7,228,379 B2

5

FIG. 7 is a diagram of an external view of an implementation of the cleaning device;

FIG. 8 is a flow chart illustrating operation of the cleaning device of FIG. 7 when cleaning a drive located within a host computer;

FIG. 9 is a diagram of external view of another implementation of the cleaning device; and

FIG. 10 is a flow chart illustrating operation of the cleaning device of FIG. 9 when cleaning a detached drive.

DETAILED DESCRIPTION

The following detailed description of the invention refers to the accompanying drawings. The same reference numbers in different drawings identify the same or similar elements. Also, the following detailed description does not limit the invention. Instead, the scope of the invention is defined by the appended claims and equivalents.

A cleaning device for erasing long-term memory devices, such as hard drives, includes a host circuit that directs a target drive's read/write head to perform specific actions relating to deleting the data on the target drive. The cleaning device is physically compact, is relatively simple to operate, and is operating system independent.

Storage devices discussed herein may be any type of long-term nonvolatile memory device. For example, the storage device may be a hard disk drive or compact flash memory. In one implementation, the storage device uses an Integrated Drive Electronics (IDE) interface. An IDE interface is a well-known electronic interface that is frequently used to connect a computer's motherboard and disk drive.

Although concepts consistent with the present invention are primarily described herein in relation to an IDE magnetic hard disk drive, these concepts may be implemented with other types of IDE media, such as flash memory with an IDE interface. Flash memories are a special type of semiconductor random access memory that retains its data after power has been removed from the system. Other types of media useable with an IDE interface include magnetic tape. In addition to the IDE interface, concepts consistent with the invention may be applied in a straightforward manner to other types of high level storage interfaces, such as the well known Small Computer System Interface (SCSI) standard or a hard drive connected through an IEEE 1394 (Firewire) connection.

For the sake of clarity the remaining description herein will be described with reference to an IDE magnetic hard disk drive, although, as mentioned above, the concepts of the invention are not limited to such drives. One skilled in the art would appreciate that other modern long-term storage device interfaces share similar functionality that could be incorporated into the concepts described herein.

Cleaning Device

FIG. 3 is a block diagram illustrating a cleaning device 300 consistent with an aspect of the present invention. Cleaning device 300 includes power supply 310, control circuitry 320, interface circuitry 330, an interface connector 340, and communication circuitry 350. Power supply 310 supplies power through a power cable connector 370 and communication circuitry 350 may connect to a communication interface 360.

A hard disk drive or compact flash drive (target drive) attaches to interface connector 340. As will be described in more detail below, control circuitry 320 detects the capabilities of the target drive and issues commands that result in

6

permanent data removal from the target drive. In other words, the target drive is thoroughly "cleaned" by cleaning device 300. Interface circuitry 330 passes instructions through connector 340 to the target device. Power supply 310 provides power to both control circuitry 320 and interface circuitry 330.

Communication circuitry 350 allows cleaning device 300 to communicate with a host computer, such as a personal computer. Communication interface 360 may be a serial interface that may be connected to a personal computer. Communication circuitry 350 may then handle the serial communication protocols between the personal computer and cleaning device 300. Additionally, instead of connecting to a personal computer, communication interface 360 may be used to interface directly to a stand-alone printer, through which status information may be output. Communication circuitry 350 and communication interface 360 are optional. In one implementation, cleaning device is implemented as a stand-alone device that may be used to permanently remove data from hard drives without using a separate computer.

Cleaning device 300 may be designed as a relatively small, lightweight, and easily portable device. In one implementation, cleaning device 300 is embodied in a case approximately 8"x10"x1.5".

Power cables allow power to be supplied to the target disk drive through power connector 370. This allows cleaning device 300 to process a target drive whether or not the target drive is still connected to a working computer or similar device (host).

FIG. 4 is a diagram illustrating portions of cleaning device 300 in additional detail. Control circuitry 320 and interface circuitry 330 of cleaning device 300 includes microprocessor 410 and programmable logic device (PLD) 490. Microprocessor 410 may be an embedded processor, such as the 80386 EX embedded processor manufactured by Intel Corporation, of Santa Clara, Calif. The integrated design of microprocessor 410 allows relatively little additional circuitry to be used to create a small, dedicated computer system. PLD 490 complements microprocessor 410 by performing logical operations required by the microprocessor 410 or other circuitry of cleaning device 300. ROM 480 stores configuration data that is initially loaded into PLD 490 on start-up. Similarly, EPROM 450 stores the initial code necessary to initialize and run microprocessor 410. Static RAM (SRAM) 440 is also connected to microprocessor 410, and is used for temporary program and data storage. Crystal oscillator 470 provides clocking signals to microprocessor 410 and PLD 490. In one implementation, crystal oscillator 470 generates a 50 MHz clock signal.

Microprocessor 410 may control a number of external devices, such as LED status indicators 420 and a processor key lock 430. Through LED status indicators 420, microprocessor 410 may provide easily understandable feedback to a user. For example, one of LEDs 420 may be a green LED that is powered by microprocessor 410 when it finishes deleting data from a drive. Alternatively, microprocessor 410 may cause an audible sound to be produced when it finishes deleting data from a drive. Processor key lock 430 is a physical interface through which a user must insert a physical key to enable microprocessor 410.

Interface 340 may include a hard drive interface 436. Drive interface 436 may be a standard IDE drive interface that connects cleaning device 300 to the target drive. Interface 340 may also include a compact flash interface 435 which, in a similar manner, allows cleaning device 300 to connect to and erase compact flash memory devices.

US 7,228,379 B2

7

In addition to connecting to the target drive through interface circuitry **340**, microprocessor **410** may be connected to external devices via RS-232 port **425** and RS-232 transceiver **460**. RS-232 port **425** may be a standard DB9 male to female serial cable.

One of ordinary skill in the art will recognize that the components shown in FIG. **4** may be selected from a wide variety of commercially available components. In one implementation, the components in FIG. **4** may be selected as follows: PLD **490**, part number EP1K50QC208-2, available from Altera Corporation of San Jose, Calif.; ROM **480**, part number EPC1PC8, available from Altera Corporation; EPROM **450**, part number AT27LV020A90JC, available from Atmel Corporation, of San Jose, Calif.; and SRAM **440**, part number CY7C1021V33L12ZCT, available from Cypress Corporation, of San Jose, Calif.

FIG. **5** is diagram that graphically illustrates the functionality of PLD **490** in additional detail. Address, data, and control lines from the microprocessor **410** are routed to PLD **490** where their information is buffered and latched as necessary in buffers **520**. Buffers **520** serve to reduce the electrical load on the processor and to stabilize the signal timing. Buffer read and write signals **530** and **540** control the direction of the bus drivers **550**. Thus, bus drivers **550** may write data into buffers **520** when read signal **530** is active and read data out of buffers **520** when write signal **540** is active. Buffers **520** and bus drivers **550** help control the data flow and distribution of the address and data buses from the microprocessor **410** to other portions of PLD **490**.

Buffering and signal conditioning for the target disk drive is provided by drive buffers **595**, which form the drive interface with the target disk drive. Buffering and signal conditioning for compact flash is provided by drive buffers **596**, which form the interface with the compact flash. Through the bus drivers **550**, the microprocessor **510** can directly read and write to the drive interface associated with the target disk drive and compact flash.

Instead of directly communicating with drive buffers **595** and **596**, bus drivers **550** may indirectly communicate with drive buffers **595** and **596** through dual ported RAM sector buffers **580** and **590**. Sector buffer **580** provides an additional layer of buffering between the microprocessor **410** and the disk drive and/or compact flash. This allows the target drive to write one sector's worth of data to RAM at high speed, while the microprocessor **410** reads a previous sector's worth of data. By allowing the operations to overlap in this fashion, the microprocessor **410** is not restricted to running at the speed of the target drive or compact flash, and is free to handle other functions until it needs the data in the sector buffers **580** or **590**.

Referring back to FIG. **4**, microprocessor **410** may include a UART that may be used for serial communications. Microprocessor **410** connects to an RS-232 transceiver **425**, which buffers the signals and generates the necessary voltages required for RS-232 communications. DB9 connector **425** connects to a corresponding DB9 connector on a host device such as a personal computer. Although communication circuitry **350** is shown as implementing a serial connection, in other implementations, other types of connections, such as parallel or USB connections, may be used.

Operation of the Cleaning Device

Cleaning device **300** operates on a target drive inserted into interface connector **340** to permanently erase the con-

8

tents of the drive. In general, cleaning device **300** removes data from a hard drive by writing to all data tracks on a drive multiple times with multiple data patterns. Data tracks are not written to sequentially. Instead, they are written in a pattern that forces the head to move from track to track in such a way as to introduce jitter in the absolute head position. Ideally, the head would be positioned a bit to one side of the track in one pass, and a bit to the other in the next pass. To accomplish this, cleaning device **300** controls the drive head to move a minimum of two tracks between writes. For some passes it is moved from an inner track to an outer one, while in other passes it is moved from an outer track to an inner one.

This process of writing with major jumps in head position may be repeated numerous times with different data patterns. By the end of the cleaning process, the motion of the recording heads and the changing data patterns renders the original data unrecoverable. One skilled in the art would appreciate that the longer this process is carried on, the more securely data is removed.

FIG. **6** is a flow chart illustrating the operation of cleaning device **300** consistent with an aspect of the present invention in additional detail. Cleaning device **300** begins, through microprocessor **410** and PLD **490**, by moving the read/write head of the target disk drive to a first track of the disk drive (e.g., track one) (Act **601**). An alternating pattern of bits (i.e., 010101 etc.) is then written to this track (Act **602**). Once complete, the disk drive head is moved to the first track plus two (i.e., track three), (Act **603**), and an alternating pattern of bits is written to this track (Act **604**). Cleaning device **300** then instructs the head to move to the first track plus one (e.g., track two) (Act **605**). An alternating bit pattern is written to this track (Act **606**). Cleaning device **300** then instructs the head to move to the first track plus three (i.e., track four), (Act **607**), and an alternating bit pattern is written to this track (Act **608**). This process is repeated until all of the tracks have been written (Acts **609** and **610**). For example, tracks five through eight may be the next set of tracks that are processed.

In one embodiment, the process shown in FIG. **6** is repeated multiple times. For a secure erase, each track may be written a minimum of seven times.

By alternating the overwriting of tracks as described in FIG. **6**, cleaning device **300** introduces jitter into the read/write head of the disk drive. This jitter increases the likelihood that data at the fringe of the track will be overwritten, thus providing for secure data erasing.

The alternating system of erase shown in FIG. **6** is one of many potential cleaning pattern that specifically applies to moving magnetic media. In alternate implementations, cleaning device **300** may perform more simple cleans for fast "good enough" cleaning, or more complex patterns may be used that are specific to other types of media.

In an alternate cleaning scheme, some target disk drives have an internal command that will erase the drive. Cleaning device **300** may use these internal erase commands when cleaning the drive.

External Structure and Operation of the Cleaning Device

As previously mentioned, cleaning device **300** may be constructed in a relatively small case, such as a case as small or smaller than 8.5"×10"×1.5". FIG. **7** is a diagram illustrating an external view of one implementation of cleaning device **300** consistent with an aspect of the invention.

US 7,228,379 B2

9

As shown in FIG. 7, the external portion of cleaning device 300 may include a power cord with a wall socket plug 701, an IDE drive cable 702 that is long enough to lead from cleaning device 300 to a computer, an on/off switch 703, and LED status lights 704. Alternatively, cleaning device 300

FIG. 8 is a flow chart illustrating operation of cleaning device 300, as shown in FIG. 7, from the perspective of a user when cleaning a hard disk drive located within a host computer.

A user begins by connecting cleaning device 300 to a power supply (Act 801) and ensuring that switch 703 is set to the "off" state (Act 802). In one implementation, this may involve plugging socket plug 701 into a wall power outlet. The user also ensures that the computer system with the target disk drive is powered down (Act 803). The user may then remove any IDE cables that are in the target drive (Act 804). This may involve removing the cover of the host computer and removing the IDE drive connector that connects the host computer and the drive. The user may then connect IDE drive cable 702 to the drive, turn power on to the host computer, and turn switch 703 to the "on" position (Acts 805, 806, and 807). In response, cleaning device 300 will power on and control the target drive to permanently delete its data. The cleaning may proceed as previously discussed with reference to FIG. 6. When cleaning device 300 has finished cleaning the drive, it signals cleaning completion via LED status lights 704 (Act 808). For example, one of LED status lights 704 may flash on and off while cleaning device 300 is operating. When cleaning device 300 completes operating, the LED status light may remain steadily on. The user may then turn switch 703 to the "off" position and disconnect the cleaning device from the host computer (Act 809).

FIG. 9 is a diagram illustrating an external view of another implementation of cleaning device 300 consistent with an aspect of the invention.

As shown in FIG. 9, the external portion of cleaning device 300 may include a power cord with a wall socket plug 901, an IDE drive cable 902, an on/off switch 903, LED status lights 904, a drive power cord 905, and an antistatic cushion 906. A target drive may be placed on the anti-static cushion and connected to the drive power cord 905 and the drive cable 902.

FIG. 10 is a flow chart illustrating operation of the cleaning device 300, shown in FIG. 9, from the perspective of a user when cleaning a detached hard disk drive.

A user begins by plugging socket plug 701 into a power supply (Act 1001) and ensuring that switch 703 is set to the "off" state (Act 1002). The target drive is then placed on the anti-static cushion 906, the IDE drive cable 902 plugged into the target drive, and the drive power cord 905 plugged into the target drive (Acts 1003 and 1004). The user may then turn switch 903 to the "on" position (Act 1005). In response, cleaning device 300 will power on and clean the drive in a manner similar to that described above with respect to FIG. 6. When cleaning device 300 has finished cleaning the drive, it signals cleaning completion via LED status lights 904 (Act 1006). The user may then turn switch 903 to the "off" position and disconnect the cleaning device and the target drive (Act 1007).

As can be appreciated, the operation of cleaning device 300, from the perspective of the user, is relatively simple. Accordingly, cleaning device 300 can be operated by only moderately trained technicians. Additionally, the operation simplicity of cleaning device 300 makes it unlikely that a

10

user will improperly use the cleaning device in a manner that only partially erases a disk drive.

User Communication

In the implementations described above, cleaning device 300 signals its operational status to a user through LEDs 704 or 904. For example, LEDs may be used to signal that: (1) cleaning device 300 is performing operations of a target device, (2) cleaning device 300 has finished cleaning the target device, and (3) an error was encountered.

In alternate embodiments, cleaning device 300 may include additional display devices such as a graphical display or a connection to a printer. With these output devices, additional status information such as percentage of operations complete, time until operations are complete, and information about the target device may be displayed. The printer could be used to generate a continuous written record of the operations performed on the target device. This written record may include, for example, type of data removal performed, time and date of data removal, and information about the target device. In a similar implementation, cleaning device 300 could interact with a host computer through serial interface 330 to upload status information to the host computer. The status information may be uploaded in real-time as the cleaning device operates or in a batch mode in which information relating to multiple target drives may be uploaded at one time.

Additional Features

In the implementations described above, cleaning device 300 has its level of data removal security set in hardware. The level of data removal security may be determined by the number of passes (i.e., the number of times the process shown in FIG. 6 is repeated) cleaning device 300 makes over the target device. In an alternate embodiment, cleaning device 300 may include a switch that a user can set to vary the cleaning level. Cleaning device 300 would then vary its number of cleaning passes based on the switch setting.

Some target drives may contain data that the drive has been instructed to classify as "hidden" data. For example, a command supported by many commercially available hard drives allows a skilled programmer to reserve a portion of the storage media. Once reserved, this area of the media is effectively hidden from the computer by the drive. Standard queries about the amount of storage on the drive return the full amount of storage minus the reserved amount. This prevents standard data removal methods from working as the computer is unaware that there is hidden data.

In one implementation, cleaning device 300 may issue appropriate commands to target drives that contain hidden data to release the hidden data. Cleaning device 300 may automatically issue these commands without the need for human intervention, thus making the entire contents of the drive accessible for data removal. In situations where the hidden data is protected by a password, the operator of cleaning device 300 may be informed that the device is protected by a password and that additional steps are necessary to complete the data removal.

In yet another implementation, cleaning device 300 could have multiple drive interface cables, through which it could simultaneously clean two or more target devices.

Still further, cleaning device 300, after erasing a target drive, may partition and/or format the target drive. For example, cleaning device 300, after erasing a target drive, may then automatically partition the drive into a single

US 7,228,379 B2

11

partition and format the drive under FAT32 drive format. In additional embodiments, cleaning device 300 may proceed, after formatting the target drive, to copy an image to the drive that defines an operating system. This may help to speed up the process of reusing drives in refurbished computer systems.

Although the cleaning device discussed above was primarily described as cleaning an IDE device, in other implementations, long-term storage devices having other interfaces, such as FireWire, USB2, or SCSI could be cleaned using concepts similar to those discussed herein.

CONCLUSION

As described above, a cleaning device permanently removes data from a target long-term memory device. The cleaning device is portable, provides easy to understand user feedback, has a simple user interface and could thus be effectively used by non-technical people.

The cleaning device has a number of advantages. It is operating system independent. The cleaning device can operate while the target device is still in the host computer system or when it has been removed from the host computer system. It is a stand alone device that can replace more complicated and more expensive devices or systems. Additionally, the cleaning device does not require that the operator have any particular knowledge of the target device; it detects the capabilities and settings of the target device and adjusts the operating parameters for optimal performance without user intervention.

Still further, the cleaning device removes data from all partitions on the target device, regardless of the data's format, and removes any reserved or protected data areas, so that all of the data storage areas on the target device are available for cleaning without requiring user intervention. Additionally, the cleaning device can provide different levels of data removal. The longer the cleaning device performs operations on a target device, the more securely it is able to remove data. The cleaning device provides feedback to a user that it has either performed operations correctly or has run into an error.

It will be apparent to one of ordinary skill in the art that the embodiments as described above may be implemented in many different forms of software, firmware, and hardware in the implementations illustrated in the figures. The actual software code or specialized control hardware used to implement aspects consistent with the present invention is not limiting of the present invention. Thus, the operation and behavior of the embodiments were described without specific reference to the specific software code, it being understood that a person of ordinary skill in the art would be able

12

to design software and control hardware to implement the embodiments based on the description herein.

The foregoing description of preferred embodiments of the present invention provides illustration and description, but is not intended to be exhaustive or to limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practice of the invention. Moreover, while a series of acts have been presented with respect to FIGS. 6, 8, and 10, the order of the acts may be different in other implementations consistent with the present invention. Moreover, non-dependent acts may be performed in parallel.

No element, act, or instruction used in the description of the present application should be construed as critical or essential to the invention unless explicitly described as such. Also, as used herein, the article "a" is intended to include one or more items. Where only one item is intended, the term "one" or similar language is used.

The scope of the invention is defined by the claims and their equivalents.

What is claimed:

1. A stand-alone, dedicated function device for removing data from a long-term memory component, comprising:

an interface for connecting the stand-alone, dedicated function device to the long-term memory component;

a control circuit configured to control the long-term memory component through the interface to irretrievably remove data from the long-term memory component without regard to data content or data storage format of the data on the long-term memory component by overwriting the data of the long-term memory component;

a user controllable switch that, when actuated by a user, causes the control circuit to commence irretrievably removing all the data from the long-term memory component;

a casing configured to contain the control circuit and the interface, the casing being of a size that is portable by the user;

a power supply configured to supply power to the control circuit;

wherein the control circuit is further configured to open a hidden storage area on the long-term memory component before irretrievably removing the data.

2. The device of claim 1, wherein the control circuit opens a hidden storage area without user intervention.

3. The device of claim 1, wherein the control circuit notifies a user when the opening a hidden storage area requires a password for release.

* * * * *