PTO/SB/05 (07-07)
Approved for use through 06/30/2010. OMB 0651-0032
U.S. Patent and Trademark Office. U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	CROSS1120-33
First Inventor	Geoffrey B. Hoese
Title	Storage Router and Method for...
Express Mail Label No.	N/A

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO:

Commissioner for Patents
P.O. Box 1450
Alexandria VA 22313-1450

1. ☑ **Fee Transmittal Form** (e.g., PTO/SB/17)
   (Submit an original and a duplicate for fee processing)
2. □ **Applicant claims small entity status.**
   See 37 CFR 1.27.
3. ☑ **Specification** [*Total Pages* 25]
   Both the claims and abstract must start on a new page
   (For information on the preferred arrangement, see MPEP 608.01(a))
4. ☑ **Drawing(s)** (*35 U.S.C. 113*) [*Total Sheets* 2]
5. **Oath or Declaration** [*Total Sheets* 4]
   a. □ Newly executed (original or copy)
   b. ☑ A copy from a prior application (37 CFR 1.63(d))
   (for continuation/divisional with Box 18 completed)
   i. □ DELETION OF INVENTOR(S)
   Signed statement attached deleting inventor(s) name in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
6. □ **Application Data Sheet.** See 37 CFR 1.76
7. □ **CD-ROM or CD-R** in duplicate, large table or Computer Program *(Appendix)*
   □ Landscape Table on CD
8. **Nucleotide and/or Amino Acid Sequence Submission**
   (if applicable, items a. – c. are required)
   a. □ Computer Readable Form (CRF)
   b. Specification Sequence Listing on:
   i. □ CD-ROM or CD-R (2 copies); or
   ii. □ Paper
   c. □ Statements verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

9. □ **Assignment Papers** (cover sheet & document(s))
   Name of Assignee____________________
10. □ **37 CFR 3.73(b) Statement** *(when there is an assignee)* □ **Power of Attorney**
11. □ **English Translation Document** *(if applicable)*
12. □ **Information Disclosure Statement** (PTO/SB/08 or PTO-1449)
    □ Copies of citations attached
13. □ **Preliminary Amendment**
14. □ **Return Receipt Postcard** (MPEP 503)
    (*Should be specifically itemized)*
15. □ **Certified Copy of Priority Document(s)**
    (if foreign priority is claimed)
16. □ **Nonpublication Request** under 35 U.S.C. 122(b)(2)(B)(i). Applicant must attach form PTO/SB/35 or equivalent.
17. ☑ Other: Cert. of Transmission and Identification of Change in Power of Attorney

18. If a CONTINUING APPLICATION, *check appropriate box, and supply the requisite information below and in the first sentence of the specification following the title, or in an Application Data Sheet under 37 CFR 1.76:*

☑ Continuation □ Divisional □ Continuation-in-part (CIP) of prior application No.: 12/552,885................

Prior application information: *Examiner* Unknown *Art Unit:* 2181

19. CORRESPONDENCE ADDRESS

☑ The address associated with Customer Number: 44654 *OR* □ Correspondence address below

Name					
Address					
City		State		Zip Code	
Country		Telephone		Email	

Signature		Date	January 20, 2010
Name (Print/Type)	John L. Adair	Registration No. (Attorney/Agent)	48,828

This collection of information is required by 37 CFR 1.53(b). The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE

TECHNICAL FIELD OF THE INVENTION

[0001] This application is a continuation of, and claims a benefit of priority under 35 U.S.C. 120 of the filing date of U.S. Patent Application Serial No. 12/552,885 entitled "Storage Router and Method for Providing Virtual Local Storage" filed 09/02/2009, which is a continuation of and claims the benefit of priority of U.S. Application Serial No. 11/851,724 entitled "Storage Router and Method for Providing Virtual Local Storage" filed 09/07/2007, which is a continuation of and claims the benefit of priority of U.S. Patent Application Serial No. 11/442,878 entitled "Storage Router and Method for Providing Virtual Local Storage" filed 09/07/2007, which is a continuation of and claims the benefit of priority of U.S. Patent Application Serial No. 11/353,826 entitled "Storage Router and Method for Providing Virtual Local Storage" filed on 02/14/2006, now U.S. Patent No. 7,340,549 issued 03/04/2008, which is a continuation of and claims the benefit of priority of U.S. Patent Application Serial No. 10/658,163 entitled "Storage Router and Method for Providing Virtual Local Storage" filed on 09/09/2003 now U.S. Patent No. 7,051,147 issued 05/23/2006, which is a continuation of and claims the benefit of benefit of priority of U.S. Patent Application Serial No. 10/081,110 by inventors Geoffrey B. Hoese and Jeffery T. Russell, entitled "Storage Router and Method for Providing Virtual Local Storage" filed on 02/22/2002, now U.S. Patent No. 6,789,152 issued on 09/07/2004, which in turn is a continuation of and claims benefit of priority of U.S. Application No. 09/354,682 by inventors Geoffrey B. Hoese and Jeffrey T. Russell, entitled "Storage Router and Method for Providing Virtual Local Storage" filed on 07/15/1999, now U.S. Patent No. 6,421,753 issued on 07/16/2002, which in turn is a continuation of and claims benefit of priority of U.S. Patent Application Serial No. 09/001,799, filed on 12/31/1997, now U.S. Patent No. 5,941,972 issued on 08/24/1999, and hereby incorporates these applications and patents by reference in their entireties as if they had been fully set forth herein.

[0002] This invention relates in general to network storage devices, and more particularly to a storage router and method for providing virtual local storage on remote SCSI storage devices to Fibre Channel devices.

BACKGROUND OF THE INVENTION

[0003] Typical storage transport mediums provide for a relatively small number of devices to be attached over relatively short distances. One such transport medium is a Small Computer System Interface (SCSI) protocol, the structure and operation of which is generally well known as is described, for example, in the SCSI-1, SCSI-2 and SCSI-3 specifications. High speed serial interconnects provide enhanced capability to attach a large number of high speed devices to a common storage transport medium over large distances. One such. serial interconnect is Fibre Channel, the structure and operation of which is described, for example, in Fibre Channel Physical and Signaling Interface (FC-PH), ANSI X3.230 Fibre Channel Arbitrated Loop (FC-AL), and ANSI X3.272 Fibre Channel Private Loop Direct Attach (FC-PLDA).

[0004] Conventional computing devices, such as computer workstations, generally access storage locally or through network interconnects. Local storage typically consists of a disk drive, tape drive, CD-ROM drive or other storage device contained within, or locally connected to the workstation. The workstation provides a file system structure that includes security controls, with access to the local storage device through native low level block protocols. These protocols map directly to the mechanisms used by the storage device and consist of data requests without security controls. Network interconnects typically provide access for a large number of computing devices to data storage on a remote network server. The remote network server provides file system structure, access control, and other miscellaneous capabilities that include the network interface. Access to data through the network server is through network protocols that the server must translate into low level requests to the storage device. A workstation with access to the server storage must translate its file system protocols into network protocols that are used to communicate with the server. Consequently, from the perspective of a workstation, or other computing device, seeking to access such server data, the access is much slower than access to data on a local storage device.

SUMMARY OF THE INVENTION

[0005] In accordance with the present invention, a storage router and method for providing virtual local storage on remote SCSI storage devices to Fibre Channel devices are disclosed that provide advantages over conventional network storage devices and methods.

[0006] According to one aspect of the present invention, a storage router and storage network provide virtual local storage on remote SCSI storage devices to Fibre Channel devices. A plurality of Fibre Channel devices, such as workstations, are connected to a Fibre Channel transport medium, and a plurality of SCSI storage devices are connected to a SCSI bus transport medium. The storage router interfaces between the Fibre Channel transport medium and the SCSI bus transport medium. The storage router maps between the workstations and the SCSI storage devices and implements access controls for storage space on the SCSI storage devices. The storage router then allows access from the workstations to the SCSI storage devices using native low level, block protocol in accordance with the mapping and the access controls.

[0007] According to another aspect of the present invention, virtual local storage on remote SCSI storage devices is provided to Fibre Channel devices. A Fibre Channel transport medium and a SCSI bus transport medium are interfaced with. A configuration is maintained for SCSI storage devices connected to the SCSI bus transport medium. The configuration maps between Fibre Channel devices and the SCSI storage devices and implements access controls for storage space on the SCSI storage devices. Access is then allowed from Fibre Channel initiator devices to SCSI storage devices using native low level, block protocol in accordance with the configuration.

[0008] A technical advantage of the present invention is the ability to centralize local storage for networked workstations without any cost of speed or overhead. Each workstation accesses its virtual local storage as if it were locally connected. Further, the centralized storage devices can be located in a significantly remote position even in excess of ten kilometers as defined by Fibre Channel standards.

[0009] Another technical advantage of the present invention is the ability to centrally control and administer storage space for connected users without limiting the speed with which the users can access local data. In addition, global access to data, backups, virus scanning and redundancy can be more easily accomplished by centrally located storage devices.

[0010] A further technical advantage of the present invention is providing support for SCSI storage devices as local storage for Fibre Channel hosts. In addition, the present invention helps to provide extended capabilities for Fibre Channel and for management of storage subsystems.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] A more complete understanding of the present invention and the advantages thereof may be acquired by referring to the following description taken in conjunction with the accompanying drawings, in which like reference numbers indicate like features, and wherein:

[0012] FIGURE 1 is a block diagram of a conventional network that provides storage through a network server;

[0013] FIGURE 2 is a block diagram of one embodiment of a storage network with a storage router that provides global access and routing;

[0014] FIGURE 3 is a block diagram of one embodiment of a storage network with a storage router that provides virtual local storage;

[0015] FIGURE 4 is a block diagram of one embodiment of the storage router of FIGURE 3; and

[0016] FIGURE 5 is a block diagram of one embodiment of data flow within the storage router of FIGURE 4.

DETAILED DESCRIPTION OF THE INVENTION

[0017] FIGURE 1 is a block diagram of a conventional network, indicated generally at 10, that provides access to storage through a network server. As shown, network 10 includes a plurality of workstations 12 interconnected with a network server 14 via a network transport medium 16. Each workstation 12 can generally comprise a processor, memory, input/output devices, storage devices and a network adapter as well as other common computer components. Network server 14 uses a SCSI bus 18 as a storage transport medium to interconnect with a plurality of storage devices 20 (tape drives, disk drives, etc.). In the embodiment of FIGURE 1, network transport medium 16 is a network connection and storage devices 20 comprise hard disk drives, although there are numerous alternate transport mediums and storage devices.

[0018] In network 10, each workstation 12 has access to its local storage device as well as network access to data on storage devices 20. The access to a local storage device is typically through native low level, block protocols. On the other hand, access by a workstation 12 to storage devices 20 requires the participation of network server 14 which implements a file system and transfers data to workstations 12 only through high level file system protocols. Only network server 14 communicates with storage devices 20 via native low level, block protocols. Consequently, the network access by workstations 12 through network server 14 is slow with respect to their access to local storage. In network 10, it can also be a logistical problem to centrally manage and administer local data distributed across an organization, including accomplishing tasks such as backups, virus scanning and redundancy.

[0019] FIGURE 2 is a block diagram of one embodiment of a storage network, indicated generally at 30, with a storage router that provides global access and routing. This environment is significantly different from that of FIGURE 1 in that there is no network server involved. In FIGURE 2, a Fibre Channel high speed serial transport 32 interconnects a plurality of workstations 36 and storage devices 38. A SCSI bus storage transport medium interconnects workstations 40 and storage devices 42. A storage router 44 then serves to interconnect these mediums and provide devices on either medium global, transparent access to devices on the other medium. Storage router 44 routes requests from initiator devices on one medium to target devices on the other medium and routes data between the target and the initiator. Storage router 44 can allow initiators and targets to be on either side. In this manner, storage router 44 enhances the functionality of Fibre Channel 32, by providing access, for example, to legacy SCSI storage devices on SCSI bus 34. In the embodiment of FIGURE 2, the operation of storage router 44 can be managed by a management station 46 connected to the storage router via a direct serial connection.

[0020] In storage network 30, any workstation 36 or workstation 40 can access any storage device 38 or storage device 42 through native low level, block protocols, and vice versa. This functionality is enabled by storage router 44 which routes requests and data as a generic transport between Fibre Channel 32 and SCSI bus 34. Storage router 44 uses tables to map devices from one medium to the other and distributes requests and data across Fibre Channel 32 and SCSI bus 34 without any security access controls. Although this extension of the high speed serial interconnect provided by Fibre Channel is beneficial, it is desirable to provide security controls in addition to extended access to storage devices through a native low level, block protocol.

[0021] FIGURE 3 is a block diagram of one embodiment of a storage network, indicated generally at 50, with a storage router that provides virtual local storage. Similar to that of FIGURE 2, storage network 50 includes a Fibre Channel high speed serial interconnect 52 and a SCSI bus 54 bridged by a storage router 56. Storage router 56 of FIGURE 3 provides for a large number of workstations 58 to be interconnected on a common storage transport and to access common storage devices 60, 62 and 64 through native low level, block protocols.

[0022] According to the present invention, storage router 56 has enhanced functionality to implement security controls and routing such that each workstation 58 can have access to a specific subset of the overall data stored in storage devices 60, 62 and 64. This specific subset of data has the appearance and characteristics of local storage and is referred to herein as virtual local storage. Storage router 56 allows the configuration and modification of the storage allocated to each attached workstation 58 through the use of mapping tables or other mapping techniques.

[0023] As shown in FIGURE 3, for example, storage device 60 can be configured to provide global data 65 which can be accessed by all workstations 58. Storage device 62 can be configured to provide partitioned subsets 66, 68, 70 and 72, where each partition is allocated to one of the workstations 58 (workstations A, B, C and D). These subsets 66, 68, 70 and 72 can only be accessed by the associated workstation 58 and appear to the associated workstation 58 as local storage accessed using native low level, block protocols. Similarly, storage device 64 can be allocated as storage for the remaining workstation 58 (workstation E).

[0024] Storage router 56 combines access control with routing such that each workstation 58 has controlled access to only the specified partition of storage device 62 which forms virtual local storage for the workstation 58. This access control allows security control for the specified data partitions. Storage router 56 allows this allocation of storage devices 60, 62 and 64 to be managed by a management station 76. Management station 76 can connect directly to storage router 56 via a direct connection or, alternately, can interface with storage router 56 through either Fibre Channel 52 or SCSI bus 54. In the latter case, management station 76 can be a workstation or other computing device with special rights such that storage router 56 allows access to mapping tables and shows storage devices 60, 62 and 64 as they exist physically rather than as they have been allocated.

[0025] The environment of FIGURE 3 extends the concept of single workstation having locally connected storage devices to a storage network 50 in which workstations

58 are provided virtual local storage in a manner transparent to workstations 58. Storage router 56 provides centralized control of what each workstation 58 sees as its local drive, as well as what data it sees as global data accessible by other workstations 58. Consequently, the storage space considered by the workstation 58 to be its local storage is actually a partition (i.e., logical storage definition) of a physically remote storage device 60, 62 or 64 connected through storage router 56. This means that similar requests from workstations 58 for access to their local storage devices produce different accesses to the storage space on storage devices 60, 62 and 64. Further, no access from a workstation 58 is allowed to the virtual local storage of another workstation 58.

[0026] The collective storage provided by storage devices 60, 62 and 64 can have blocks allocated by programming means within storage router 56. To accomplish this function, storage router 56 can include routing tables and security controls that define storage allocation for each workstation 58. The advantages provided by implementing virtual local storage in centralized storage devices include the ability to do collective backups and other collective administrative functions more easily. This is accomplished without limiting the performance of workstations 58 because storage access involves native low level, block protocols and does not involve the overhead of high level protocols and file systems required by network servers.

[0027] FIGURE 4 is a block diagram of one embodiment of storage router 56 of FIGURE 3. Storage router 56 can comprise a Fibre Channel controller 80 that interfaces with Fibre Channel 52 and a SCSI controller 82 that interfaces with SCSI bus 54. A buffer 84 provides memory work space and is connected to both Fibre Channel controller 80 and to SCSI controller 82. A supervisor unit 86 is connected to Fibre Channel controller 80, SCSI controller 82 and buffer 84. Supervisor unit 86 comprises a microprocessor for controlling operation of storage router 56 and to handle mapping and-security access for requests between Fibre Channel 52 and SCSI bus 54.

[0028] FIGURE 5 is a block diagram of one embodiment of data flow within storage router 56 of FIGURE 4. As shown, data from Fibre Channel 52 is processed by a Fibre Channel (FC) protocol unit 88 and placed in a FIFO queue 90. A direct memory access (DMA) interface 92 then takes data out of FIFO queue 90 and places it in buffer 84. Supervisor unit 86 processes the data in buffer 84 as represented by supervisor processing 93. This processing involves mapping between Fibre Channel 52 and SCSI bus 54 and applying access controls and routing functions. A DMA interface 94 then pulls data from buffer 84 and places it into a buffer 96. A SCSI protocol unit 98 pulls data from buffer 96 and communicates the data on SCSI bus 54. Data flow in the reverse direction, from SCSI bus 54 to Fibre Channel 52, is accomplished in a reverse manner.

[0029] The storage router of the present invention is a bridge device that connects a Fibre Channel link directly to a SCSI bus and enables the exchange of SCSI command set information between application clients on SCSI bus devices and the Fibre Channel links. Further, the storage router applies access controls such that virtual local storage can be established in remote SCSI storage devices for workstations on the Fibre Channel link. In one embodiment, the storage router provides a connection for Fibre Channel links running the SCSI Fibre Channel Protocol (FCP) to legacy SCSI devices attached to a SCSI bus. The Fibre Channel topology is typically an Arbitrated Loop (FC_AL).

[0030] In part, the storage router enables a migration path Fibre Channel based, serial SCSI networks by providing connectivity for legacy SCSI bus devices. The storage router can be attached to a Fibre Channel Arbitrated Loop and a SCSI bus to support a number of SCSI devices. Using configuration settings, the storage router can make the SCSI bus devices available on the Fibre Channel network as FCP logical units. Once the configuration is defined, operation of the storage router is transparent to application clients. In this manner, the storage router can form an integral part of the migration to new Fibre Channel based networks while providing a means to continue using legacy SCSI devices.

[0031] In one implementation (not shown), the storage router can be a rack mount or free standing device with an internal power supply. The storage router can have a Fibre Channel and SCSI port, and a standard, detachable power cord can be used, the FC connector can be a copper DB9 connector, and the SCSI connector can be a 68-pin type. Additional modular jacks can be provided for a serial port and an 802.3 10BaseT port, i.e. twisted pair Ethernet, for management access. The SCSI port of the storage router an support SCSI direct and sequential access target devices and can support SCSI initiators, as well. The Fibre Channel port can interface to SCSI-3 FCP enabled devices and initiators.

[0032] To accomplish its functionality, one implementation of the storage router uses: a Fibre Channel interface based on the HEWLETT-PACKARD TACHYON HPFC-5000 controller and a GLM media interface; an Intel 80960RP processor, incorporating independent data and program memory spaces, and associated logic required to implement a stand alone processing system; and a serial port for debug and system configuration. Further, this implementation includes a SCSI interface supporting Fast-20 based on the SYMBIOS 53C8xx series SCSI controllers, and an operating system based upon the WIND RIVERS SYSTEMS VXWORKS or IXWORKS kernel, as determined by design. In addition, the storage router includes software as required to control basic functions of the various elements, and to provide appropriate translations between the FC and SCSI protocols.

[0033] The storage router has various modes of operation that are possible between FC and SCSI target and initiator combinations. These modes are: FC Initiator to SCSI Target; SCSI Initiator to FC Target; SCSI Initiator to SCSI Target; and FC Initiator to FC Target. The first two modes can be supported concurrently in a single storage router device and are discussed briefly below. The third mode can involve two storage router devices back to back and can serve primarily as a device to extend the physical distance beyond that possible via a direct SCSI connection. The last mode can be used to carry FC protocols encapsulated on other transmission technologies (e.g. ATM, SONET), or to act as a bridge between two FC loops (e.g. as a two port fabric).

[0034] The FC Initiator to SCSI Target mode provides for the basic configuration of a server using Fibre Channel to communicate with SCSI targets. This mode requires that a host system have an FC attached device and associated device drivers and software to generate SCSI-3 FCP requests. This system acts as an initiator using the storage router to communicate with SCSI target devices. The SCSI devices supported can include SCSI-2 compliant direct or sequential access (disk or tape) devices. The storage router serves to translate command and status information and transfer data between SCSI-3 FCP and SCSI-2, allowing the use of standard SCSI-2 devices in a Fibre Channel environment.

[0035] The SCSI Initiator to FC Target mode provides for the configuration of a server using SCSI-2 to communicate with Fibre Channel targets. This mode requires that a host system has a SCSI-2 interface and driver software to control SCSI-2 target devices. The storage router will connect to the SCSI-2 bus and respond as a target to multiple target IDs. Configuration information is required to identify the target IDs to which the bridge will respond on the SCSI-2 bus. The storage router then translates the SCSI-2 requests to SCSI-3 FCP requests, allowing the use of FC devices with a SCSI host system. This will also allow features such as a tape device acting as an initiator on the SCSI bus to provide full support for this type of SCSI device.

[0036] In general, user configuration of the storage router will be needed to support various functional modes of operation. Configuration can be modified, for example, through a serial port or through an Ethernet port via SNMP (simple network management protocol) or the Telnet session. Specifically, SNMP manageability can be provided via a B02.3 Ethernet interface. This can provide for configuration changes as well as providing statistics and error information. Configuration can also be performed via TELNET or RS-232 interfaces with menu driven command interfaces. Configuration information can be stored in a segment of flash memory and can be retained across resets and power off cycles. Password protection can also be provided.

[0037] In the first two modes of operation, addressing information is needed to map from FC addressing to SCSI addressing and vice versa. This can be 'hard' configuration data, due to the need for address information to be maintained across initialization and partial reconfigurations of the Fibre Channel address space. In an arbitrated loop configuration, user configured addresses will be needed for AL_PAs in order to insure that known addresses are provided between loop reconfigurations.

[0038] With respect to addressing, FCP and SCSI 2 systems employ different methods of addressing target devices. Additionally, the inclusion of a storage router means that a method of translating device IDs needs to be implemented. In addition, the storage router can respond to commands without passing the commands through to the opposite interface. This can be implemented to allow all generic FCP and SCSI commands to pass through the storage router to address attached devices, but allow for configuration and diagnostics to be performed directly on the storage router through the FC and SCSI interfaces.

[0039] Management commands are those intended to be processed by the storage router controller directly. This may include diagnostic, mode, and log commands as well as other vendor-specific commands. These commands can be received and processed by both the FOP and SCSI interfaces, but are not typically bridged to the opposite interface. These commands may also have side effects on the operation of the storage router, and cause other storage router operations to change or terminate.

[0040] A primary method of addressing management commands though the FCP and SCSI interfaces can be through peripheral device type addressing. For example, the storage router can respond to all operations addressed to logical unit (LUN) zero as a controller device. Commands that the storage router will support can include INQUIRY as well as vendor-specific management commands. These are to be generally consistent with SCC standard commands.

[0041] The SCSI bus is capable of establishing bus connections between targets. These targets may internally address logical units. Thus, the prioritized addressing scheme used by SCSI subsystems can be represented as follows: BUS:TARGET:LOGICAL UNIT. The BUS identification is intrinsic in the configuration, as a SCSI initiator is attached to only one bus. Target addressing is handled by bus arbitration from information provided to the arbitrating device. Target addresses are assigned to SCSI devices directly through some means of configuration, such as a hardware jumper, switch setting, or device specific software configuration. As such, the SCSI protocol provides only logical unit addressing within the Identify message. Bus and target information is implied by the established connection.

[0042] Fibre Channel devices within a fabric are addressed by a unique port identifier. This identifier is assigned to a port during certain well-defined states of the FC protocol. Individual ports are allowed to arbitrate for a known, user defined address. If such an address is not provided, or if arbitration for a particular-user address fails, the port is assigned a unique address by the FC protocol. This address is generally not guaranteed to be unique between instances. Various scenarios exist where the AL-PA of a device will change, either after power cycle or loop reconfiguration.

[0043] The FC protocol also provides a logical unit address field within command structures to provide addressing to devices internal to a port. The FCP_CMD payload specifies an eight byte LUN field. Subsequent identification of the exchange between devices is provided by the FQXID (Fully Qualified Exchange ID).

[0044] FC ports can be required to have specific addresses assigned. Although basic functionality is not dependent on this, changes in the loop configuration could result in disk targets changing identifiers with the potential risk of data corruption or loss. This configuration can be straightforward, and can consist of providing the device a loop-unique ID (AL_PA) in the range of " 01h" to "EFh." Storage routers could be shipped with a default value with the assumption that most configurations will be using single storage routers and no other devices requesting the present ID. This would provide a minimum amount of initial configuration to the system administrator. Alternately, storage routers could be defaulted to assume any address so that configurations requiring multiple storage routers on a loop would not require that the administrator assign a unique ID to the additional storage routers.

[0045] Address translation is needed where commands are issued in the cases FC Initiator to SCSI Target and SCSI Initiator to FC Target. Target responses are qualified by the FQXID and will retain the translation acquired at the beginning of the exchange. This prevents configuration changes occurring during the course of execution of a command from causing data or state information to be inadvertently misdirected. Configuration can be required in cases of SCSI Initiator to FC Target, as discovery may not effectively allow for FCP targets to consistently be found. This is due to an FC arbitrated loop supporting addressing of a larger number of devices than a SCSI bus and the possibility of FC devices changing their AL-PA due to device insertion or other loop initialization.

[0046] In the direct method, the translation to BUS:TARGET:LUN of the SCSI address information will be direct. That is, the values represented in the FCP LUN field will directly map to the values in effect on the SCSI bus. This provides a clean translation and does not require SCSI bus discovery. It also allows devices to be dynamically added to the SCSI bus without modifying the address map. It may not allow for complete discovery by FCP initiator devices, as gaps between device addresses may halt the discovery process. Legacy SCSI device drivers typically halt discovery on a target device at the first unoccupied LUN, and proceed to the next target. This would lead to some devices not being discovered. However, this allows for hot plugged devices and other changes to the loop addressing.

[0047] In the ordered method, ordered translation requires that the storage router perform discovery on reset, and collapses the addresses on the SCSI bus to sequential FSP LUN values. Thus, the FCP LUN values 0-N can represent N+1 SCSI devices, regardless of SCSI address values, in the order in which they are isolated during the SCSI discovery process. This would allow the FCP initiator discovery process to identify all mapped SCSI devices without further configuration. This has the limitation that hot-plugged devices will not be identified until the next reset cycle. In this case, the address may also be altered as well.

[0048] In addition to addressing, according to the present invention, the storage router provides configuration and access controls that cause certain requests from FC Initiators to be directed to assigned virtual local storage partitioned on SCSI storage devices. For example, the same request for LUN 0 (local storage) by two different FC Initiators can be directed to two separate subsets of storage. The storage router can use tables to map, for each initiator, what storage access is available and what partition is being addressed by a particular request. In this manner, the storage space provided by SCSI storage devices can be allocated to FC initiators to provide virtual local storage as well as to create any other desired configuration for secured access.

[0049] Although the present invention has been described in detail, it should be understood that various changes, substitutions, and alterations can be made hereto without departing from the spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A storage router for providing virtual local storage on remote storage devices, comprising:

a first controller operable to interface with a first transport medium, wherein the first medium is a serial transport media; and

a processing device coupled to the first controller, wherein the processing device is configured to:

maintain a map to allocate storage space on the remote storage devices to devices connected to the first transport medium by associating representations of the devices connected to the first transport medium with representations of storage space on the remote storage devices, wherein each representation of a device connected to the first transport medium is associated with one or more representations of storage space on the remote storage devices;

control access from the devices connected to the first transport medium to the storage space on the remote storage devices in accordance with the map; and

allow access from devices connected to the first transport medium to the remote storage devices using native low level block protocol.

2. The storage router of Claim 1, wherein the map associates a representation of storage space on the remote storage devices with multiple devices connected to the first transport medium.

3. The storage router of Claim 1, wherein the storage space on the remote storage devices comprises storage space on multiple remote storage devices.

4. The storage router of Claim 1, wherein the map associates a representation of a device connected to the first transport medium with a representation of an entire storage space of at least one remote storage device.

5. The storage router of Claim 1, wherein the map resides at the storage router and is maintained at the storage router.

6. The storage router of Claim 1, wherein the native low level block protocol is received at the storage router via the first transport medium and the processing device uses the received native low level block protocol to allow the devices connected to the first transport medium access to storage space specifically allocated to them in the map.

7. The storage router of Claim 1, wherein the storage router is configured to receive commands according to a first low level block protocol from the device connected to the first transport medium and forward commands according to a second low level block protocol to the remote storage devices.

8. The storage router of Claim 7, wherein the first low level block protocol is an FCP protocol and the second low level block protocol is a protocol other than FCP.

9. The storage router of Claim 1, wherein the map comprises one or more tables.

10. The storage router of Claim 1, wherein the virtual local storage is provided to the devices connected to the first transport medium in a manner that is transparent to the devices and wherein the storage space allocated to the devices connected to the first transport medium appears to the devices as local storage.

11. The storage router of Claim 1, wherein the storage router provides centralized control of what the devices connected to the first transport medium see as local storage.

12. The storage router of Claim 1, wherein the representations of storage space comprise logical unit numbers that represent a subset of storage on the remote storage devices.

13. The storage router of Claim 12, wherein the storage router is operable to route requests to the same logical unit number from different devices connected to the first transport medium to different subsets of storage space on the remote storage devices.

14. The storage router of Claim 1, wherein the representations of devices connected to the first transport medium are unique identifiers.

15. The storage router of Claim 14, wherein the unique identifiers are world wide names.

16. The storage router of Claim 1, wherein the storage router is configured to allow modification of the map in a manner transparent to and without involvement of the devices connected to the first transport medium.

17. The storage router of Claim 1, wherein the processing device is a microprocessor.

18. The storage router of Claim 1, wherein the processing device is a microprocessor and associated logic to implement a stand-alone processing system.

19. The storage router of Claim 1, wherein the first transport medium is a fibre channel transport medium and further comprising a second transport medium connected to the remote storage devices that is a fibre channel transport medium.

20. A storage network comprising:

a set of devices connected a first transport medium, wherein the first transport medium;

a set of remote storage devices connected to a second transport medium;

a storage router connected to the serial transport medium;

a storage router connected to the first transport medium and second transport medium to provide virtual local storage on the remote storage devices, the storage router configured to:

maintain a map to allocate storage space on the remote storage devices to devices connected to the first transport medium by associating representations of the devices connected to the first transport medium with representations of storage space on the remote storage devices, wherein each representation of a device connected to the first transport medium is associated with one or more representations of storage space on the remote storage devices;

control access from the devices connected to the first transport medium to the storage space on the remote storage devices in accordance with the map; and allow access from devices connected to the first transport medium to the remote storage devices using native low level block protocol.

21. The storage network of Claim 20, wherein the map associates a representation of storage space on the remote storage devices with multiple devices connected to the first transport medium.

22. The storage network of Claim 20, wherein the storage space on the remote storage devices comprises storage space on multiple remote storage devices.

23. The storage network of Claim 20, wherein the map associates a representation of a device connected to the first transport medium with a representation of an entire storage space of at least one remote storage device.

24. The storage network of Claim 20, wherein the map resides at the storage router and is maintained at the storage router.

25. The storage network of Claim 20, wherein the native low level block protocol is received at the storage router via the first transport medium and the storage router uses the received native low level block protocol to allow the devices connected to the first transport medium access to storage space specifically allocated to them in the map.

26. The storage router of Claim 20, wherein the storage router is configured to receive commands according to a first low level block protocol from the device connected to the first transport medium and forward commands according to a second low level block protocol to the remote storage devices.

27. The storage network of Claim 20, wherein the first low level block protocol is an FCP protocol and the second low level block protocol is a protocol other than FCP.

28. The storage network of Claim 20, wherein the map comprises one or more tables.

29. The storage network of Claim 20, wherein the virtual local storage is provided to the devices connected to the first transport medium in a manner that is transparent to the devices and wherein the storage space allocated to the devices connected to the first transport medium appears to the devices as local storage.

30. The storage network of Claim 20, wherein the storage router provides centralized control of what the devices connected to the first transport medium see as local storage.

31. The storage network of Claim 20, wherein the representations of storage space comprise logical unit numbers that represent a subset of storage on the remote storage devices.

32. The storage network of Claim 31, wherein the storage router is operable to route requests to the same logical unit number from different devices connected to the first transport medium to different subsets of storage space on the remote storage devices.

33. The storage network of Claim 20, wherein the representations of devices connected to the first transport medium are unique identifiers.

34. The storage network of Claim 33, wherein the unique identifiers are world wide names.

35. The storage network of Claim 20, wherein the storage router is configured to allow modification of the map in a manner transparent to and without involvement of the devices connected to the first transport medium.

36. The storage network of Claim 20, wherein the first transport medium is a fibre channel transport medium and the second transport medium is a fibre channel transport medium.

37. A method for providing virtual local storage on remote storage devices comprising:

connecting a storage router between a set of devices connected to a first transport medium and a set of remote storage devices, wherein the first transport medium is a serial transport medium;

maintaining a map at the storage router to allocate storage space on the remote storage devices to devices connected to the first transport medium by associating representations of the devices connected to the first transport medium with representations of storage space on the remote storage devices, wherein each representation of a device connected to the first transport medium is associated with one or more representations of storage space on the remote storage devices;

controlling access from the devices connected to the first transport medium to the storage space on the remote storage devices in accordance with the map; and

allowing access from devices connected to the first transport medium to the remote storage devices using native low level block protocol.

38. The method of Claim 37, wherein the map associates a representation of storage space on the remote storage devices with multiple devices connected to the first transport medium.

39. The method of Claim 37, wherein the storage space on the remote storage devices comprises storage space on multiple remote storage devices.

40. The method of Claim 37, wherein the map associates a representation of a device connected to the first transport medium with a representation of an entire storage space of at least one remote storage device.

41. The method of Claim 37, wherein the map resides at the storage router and is maintained at the storage router.

42. The method of Claim 37, further comprising:

receiving the native low level block protocol at the storage router via the first transport medium;

using the received native low level block protocol at the storage router to allow the devices connected to the first transport medium access to storage space specifically allocated to them in the map.

43. The method of Claim 37, further comprising receiving commands at the storage router according to a first low level block protocol from the device connected to the first transport medium and forwarding commands according to a second low level block protocol to the remote storage devices.

44. The method of Claim 43, wherein the first low level block protocol is an FCP protocol and the second low level block protocol is a protocol other than FCP.

45. The method of Claim 37, wherein the map comprises one or more tables.

46. The method of Claim 37, wherein the virtual local storage is provided to the devices connected to the first transport medium in a manner that is transparent to the devices and wherein the storage space allocated to the devices connected to the first transport medium appears to the devices as local storage.

47. The method of Claim 37, wherein the storage router provides centralized control of what the devices connected to the first transport medium see as local storage.

48. The method of Claim 37, wherein the representations of storage space comprise logical unit numbers that represent a subset of storage on the remote storage devices.

49. The method of Claim 48, wherein the storage router is operable to route requests to the same logical unit number from different devices connected to the first transport medium to different subsets of storage space on the remote storage devices.

50. The method of Claim 37, wherein the representations of devices connected to the first transport medium are unique identifiers.

51. The method of Claim 50, wherein the unique identifiers are world wide names.

52. The method of Claim 51, wherein the storage router is configured to allow modification of the map in a manner transparent to and without involvement of the devices connected to the first transport medium.

53. The method of Claim 1 wherein connecting the storage router between a set of devices connected to a first transport medium and a set of remote storage devices further comprises connecting the storage router between a first fibre channel transport medium and a second fibre channel transport medium.

STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE

ABSTRACT OF THE DISCLOSURE

[0050] A storage router and storage network provide virtual local storage on remote storage devices. A plurality of devices are connected to a first transport medium. In one embodiment, a storage router maintains a map to allocate storage space on the remote storage devices to devices connected to the first transport medium by associating representations of the devices connected to the first transport medium with representations of storage space on the remote storage devices. The storage router controls access from the devices connected to the first transport medium to the storage space on the remote storage devices in accordance with the map and allows access from devices connected to the first transport medium to the remote storage devices using native low level block protocol.

Electronic Patent Application Fee Transmittal

Application Number:	
Filing Date:	
Title of Invention:	STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE
First Named Inventor/Applicant Name:	Geoffrey B. Hoese
Filer:	John L. Adair/Delia Narvaiz
Attorney Docket Number:	CROSS1120-33

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Utility application filing	1011	1	330	330
Utility Search Fee	1111	1	540	540
Utility Examination Fee	1311	1	220	220
Pages:				
Claims:				
Claims in excess of 20	1202	33	52	1716
Miscellaneous-Filing:				
Petition:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Miscellaneous:				
		Total in USD ($)		**2806**

Electronic Acknowledgement Receipt

EFS ID:	6845953
Application Number:	12690592
International Application Number:	
Confirmation Number:	8115
Title of Invention:	STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE
First Named Inventor/Applicant Name:	Geoffrey B. Hoese
Customer Number:	44654
Filer:	John L. Adair/Delia Narvaiz
Filer Authorized By:	John L. Adair
Attorney Docket Number:	CROSS1120-33
Receipt Date:	20-JAN-2010
Filing Date:	
Time Stamp:	16:24:15
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$2806
RAM confirmation Number	3127
Deposit Account	503183
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Miscellaneous Incoming Letter	CROSS1120-33_Cert_Transmission.pdf	23572 7d03f1dc62be0e2f8fd6d2793fc0af2a9d51a966	no	1
Warnings:					
Information:					
2	Miscellaneous Incoming Letter	CROSS1120-33_Id_chg_POA.pdf	153882 c7162bfeeceb5044f313e4c768dc96c5d5951a60	no	6
Warnings:					
Information:					
3	Oath or Declaration filed	CROSS1120-33_Declaration_fr_parent.pdf	125807 c84a7ce304d738dabf74ae001585566f64a8a261	no	4
Warnings:					
Information:					
4	Drawings-only black and white line drawings	CROSS1120-33_Drawings.pdf	36044 f06f4160ea41d04338498da87a96713f1652c97b	no	2
Warnings:					
Information:					
5	Transmittal of New Application	CROSS1120-33_Transmittal.pdf	75251 bcbbc251aa07bf22f000918791a0e6c8517fdac9	no	1
Warnings:					
Information:					
6		CROSS1120-33_Application.pdf	89877 516836a9c3cb832e9c0eee0634652871cfb45e3a	yes	25

Multipart Description/PDF files in .zip description

Document Description	Start	End
Specification	1	16
Claims	17	24
Abstract	25	25

Warnings:

Information:					
7	Fee Worksheet (PTO-875)	fee-info.pdf	36748 13f98da56b3ebe3eaa50c6aeefcc73a79b310d2f	no	2
Warnings:					
Information:					
			Total Files Size (in bytes):	541181	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

Electronic Acknowledgement Receipt

EFS ID:	6845953
Application Number:	12690592
International Application Number:	
Confirmation Number:	8115
Title of Invention:	STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE
First Named Inventor/Applicant Name:	Geoffrey B. Hoese
Customer Number:	44654
Filer:	John L. Adair/Delia Narvaiz
Filer Authorized By:	John L. Adair
Attorney Docket Number:	CROSS1120-33
Receipt Date:	20-JAN-2010
Filing Date:	
Time Stamp:	16:24:15
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$2806
RAM confirmation Number	3127
Deposit Account	503183
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Miscellaneous Incoming Letter	CROSS1120-33_Cert_Transmission.pdf	23572 7d03f1dc62be0e2f8fd6d2793fc0af2a9d51a966	no	1
Warnings:					
Information:					
2	Miscellaneous Incoming Letter	CROSS1120-33_Id_chg_POA.pdf	153882 c7162bfeeceb5044f313e4c768dc96c5d5951a60	no	6
Warnings:					
Information:					
3	Oath or Declaration filed	CROSS1120-33_Declaration_fr_parent.pdf	125807 c84a7ce304d738dabf74ae001585566f64a8a261	no	4
Warnings:					
Information:					
4	Drawings-only black and white line drawings	CROSS1120-33_Drawings.pdf	36044 f06f4160ea41d04338498da87a96713f1652c97b	no	2
Warnings:					
Information:					
5	Transmittal of New Application	CROSS1120-33_Transmittal.pdf	75251 bcbbc251aa07bf22f000918791a0e6c8517fdac9	no	1
Warnings:					
Information:					
6		CROSS1120-33_Application.pdf	89877 516836a9c3cb832e9c0eee0634652871cfb45e3a	yes	25

Multipart Description/PDF files in .zip description

Document Description	Start	End
Specification	1	16
Claims	17	24
Abstract	25	25

Warnings:

Information:					
7	Fee Worksheet (PTO-875)	fee-info.pdf	36748 13f98da56b3ebe3eaa50c6aeefcc73a79b310d2f	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	541181		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF TRANSMISSION VIA EFS-WEB SYSTEM	Atty Docket No. **CROSS1120-33**

In the Application of:
Geoffrey B. Hoese

Date Filed:
Herewith

Title:
Storage Router and Method for Providing Virtual Local Storage

Mail Stop: Patent Application
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

I hereby certify that the attached Utility Patent Application Transmittal Form, Declaration (copy from parent), Identification of Change in Power of Attorney Under 37 CFR 1.63(d)(4), Continuation Patent Application and copies of Drawings (2 sheets) are being deposited electronically using the United States Patent Office EFS-Web System on January 20, 2010

Respectfully submitted,
Sprinkle IP Law Group

Delia Narvaiz
Delia Narvaiz

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE	
IDENTIFICATION OF CHANGE IN POWER OF ATTORNEY UNDER 37 C.F.R. 1.63(d)(4)	Atty. Docket No. **CROSS1120-33**

Applicant **Geoffrey B. Hoese, et al.**	
Application Number **Unknown**	Date Filed **January ___, 2010**
Title **Storage Router and Method for Providing Virtual Local Storage**	
Confirmation Number: **Unknown**	Group Art Unit **Unknown**

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Certificate of Transmission Under 37 C.F.R. § 1.8

I hereby certify that this correspondence is being transmitted to the U.S. Patent and Trademark Office via the EFS-Web filing system on January 20, 2010

Delia Narvaiz
Delia Narvaiz

Dear Sir:

The above-referenced application is a continuation application of and claims priority from U.S. Patent Application No. 12/552,885 filed on 09/02/2009 ("Prior Application"). The power of attorney and correspondence address were changed during the prosecution of the Prior Application. 37 C.F.R. 1.63(d)(4) states:

> Where the power of attorney or correspondence address was changed during the prosecution of the prior application, the change in power of attorney or correspondence address must be identified in the continuation or divisional application. Otherwise, the Office may not recognize in the continuation or divisional application the change of power of attorney or correspondence address during the prosecution of the prior application.

As evidenced by Exhibit A submitted herewith, during the prosecution of Prior Application, the power of attorney was changed to attorneys under Customer No. 44654, all of the firm of Sprinkle IP Law Group, and the correspondence address was changed to:

Customer No. 44654
Sprinkle IP Law Group
1301 W. 25th Street, Suite 408
Austin, Texas 78705

Please recognize these changes in U.S. instant application. Please call the undersigned with any question you may have regarding this matter.

Respectfully submitted,

Sprinkle IP Law Group
Attorneys for Applicant

John L. Adair
Reg. No. 48,828

Dated: Jan 20, 2010

1301 W. 25th Street, Suite 408
Austin, Texas, 78705
Tel. (512) 637-9220
Fax. (512) 317-9088

EXHIBIT “A”

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371(C) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
12/552,885	09/02/2009	Geoffrey B. Hoese	CROSS1120-30

CONFIRMATION NO. 5484

POA ACCEPTANCE LETTER

44654
SPRINKLE IP LAW GROUP
1301 W. 25TH STREET
SUITE 408
AUSTIN, TX 78705

OC000000037859989

Date Mailed: 09/22/2009

NOTICE OF ACCEPTANCE OF POWER OF ATTORNEY

This is in response to the Power of Attorney filed 09/02/2009.

The Power of Attorney in this application is accepted. Correspondence in this application will be mailed to the above address as provided by 37 CFR 1.33.

/bcao/

Office of Data Management, Application Assistance Unit (571) 272-4000, or (571) 272-4200, or 1-888-786-0101

IN T... ...ITED STATES PATENT AND TRADE... ...K OFFICE

REVOCATION AND POWER OF ATTORNEY AND CHANGE OF MAILING ADDRESS	Atty. Docket No. (Opt.) CROSS1120-13
Applicants Geoffrey B Hoese, et. al.	
Application Number 10/658,163	Filed 9/9/2003
For STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE	
Group Art Unit 2186	Examiner Unknown
Confirmation No. 5675	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Certification Under 37 C.F.R. §1.8

I hereby certify that this document is being transmitted to COMMISSIONER FOR PATENTS via facsimile on 5-17, 2004.

Reynetto DeVeau
Signed Name

Reynetto DeVeau
Printed Name

Crossroads Systems, Inc., 100% owner of the above-identified patent application, as evidenced by the Assignment recorded on December 31, 1997 on Reel/Frame: 8929/0290, hereby revokes all previous Powers of Attorney and appoints the following attorneys under Customer No. 44654, all of the firm of SPRINKLE IP LAW GROUP, to prosecute the above-identified Patent and to transact all business in the Patent and Trademark Office connected therewith.

STEVEN R. SPRINKLE	Registration No. 40,825
JOHN ADAIR	Registration No. 48,828
ARI AKMAL	Registration No. 51,388

Direct all telephone calls and correspondence to:

Customer No. 44654
SPRINKLE IP LAW GROUP
P.O. Box 684767
Austin, TX 78768-4767
Attn: Steven Sprinkle
Tel. (512) 637.9220 / Fax (512) 371.9088

I hereby state I am authorized to act on behalf of CROSSROADS SYSTEMS, INC.

Respectfully submitted,

Crossroads Systems, Inc.

Dated: 8/11, 2004

By:
Robert Sims, President & CEO

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371 (c) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
10/658,163	09/09/2003	Geoffrey B. Hoese	CROSS1120-13

CONFIRMATION NO. 5675

OC000000018039068

OC000000018039068

44654
SPRINKLE IP LAW GROUP
1301 W. 25TH STREET
SUITE 408
AUSTIN, TX 78705

Date Mailed: 02/10/2006

NOTICE OF ACCEPTANCE OF POWER OF ATTORNEY

This is in response to the Power of Attorney filed 07/26/2005.

The Power of Attorney in this application is accepted. Correspondence in this application will be mailed to the above address as provided by 37 CFR 1.33.

ALBERTHA L JACKSON
2100 (571) 272-3594

ATTORNEY/APPLICANT COPY

DECLARATION AND POWER OF ATTORNEY

As the below named inventor, I declare that:

My residence, post office address and citizenship are as stated below next to my name, that I believe I am the original, first and joint inventor of the subject matter which is claimed and for which a patent is sought on the invention or design entitled STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE, the specification of which (check one):

__X__ is attached hereto; or

_____ was filed on _______________ as
Application Serial No. ____________ and was
amended on _______________ (if applicable);

that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above; and that I acknowledge the duty to disclose to the U.S. Patent and Trademark Office all information known to me to be material to patentability as defined in 37 C.F.R. § 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. § 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application(s) for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Number	Country	Date Filed	Priority Claimed (Yes) (No)

None.

I hereby claim the benefit under 35 U.S.C. § 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application(s) in the manner provided by the first paragraph of 35 U.S.C. § 112, I acknowledge the duty to disclose to the U.S. Patent and Trademark Office all information known to me to be material to patentability as defined in 37 C.F.R. § 1.56 which became available between the filing date of the prior application(s) and the national or PCT international filing date of this application:

Application Serial Number	Date Filed	Status

None.

I hereby appoint:

Jerry W. Mills	Reg. No. 23,005
Robert M. Chiaviello, Jr.	Reg. No. 32,461
Ann C. Livingston	Reg. No. 32,479
William N. Hulsey III	Reg. No. 33,402
Thomas R. Felger	Reg. No. 28,842
Charles S. Fish	Reg. No. 35,870
Wei Wei Jeang	Reg. No. 33,305
Kevin J. Meek	Reg. No. 33,738
Anthony E. Peterman	Reg. No. 38,270
Barton E. Showalter	Reg. No. 38,302
David G. Wille	Reg. No. 38,363
Philip W. Woo	Reg. No. 39,880
Bradley P. Williams	Reg. No. 40,227
Terry J. Stalford	Reg. No. 39,522
Christopher W. Kennerly	Reg. No. 40,675
Daniel P. Stewart	Reg. No. 41,332
Roger J. Fulghum	Reg. No. 39,678
Rodger L. Tate	Reg. No. 27,399
Scott F. Partridge	Reg. No. 28,142
James B. Arpin	Reg. No. 33,470
James Remenick	Reg. No. 36,902

Jay B. Johnson	Reg. No. 38,193
Christopher C. Campbell	Reg. No. 37,291
Stacy B. Margolies	Reg. No. 39,760
Robert W. Holland	Reg. No. 40,020
Steven R. Sprinkle	Reg. No. 40,825

all of the firm of Baker & Botts, L.L.P., my attorneys with full power of substitution and revocation, to prosecute this application and to transact all business in the United States Patent and Trademark Office connected therewith, and to file and prosecute any international patent applications filed thereon before any international authorities.

Send Correspondence To:

Baker & Botts, L.L.P.
2001 Ross Avenue
Dallas, Texas 75201-2980

Direct Telephone Calls To:

Anthony E. Peterman
at (512) 322-2599
Atty. Docket No. 064113.0103

I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

Full name of the first inventor

Geoffrey B. Hoese

Inventor's signature

Date 12/22/97

Residence (City, County, State) Austin, Travis County,
Texas

Citizenship United States of America

Post Office Address 1904 Ann Arbor Avenue
Austin, Texas 78704

Full name of the second inventor

Jeffry T. Russell

Inventor's signature Jeffry T Russell

Date December 22, 1997

Residence (City, County, State) Cibolo, Guadalupe County,
Texas

Citizenship United States of America

Post Office Address 205 Kariba Cove
Cibolo, Texas 78108

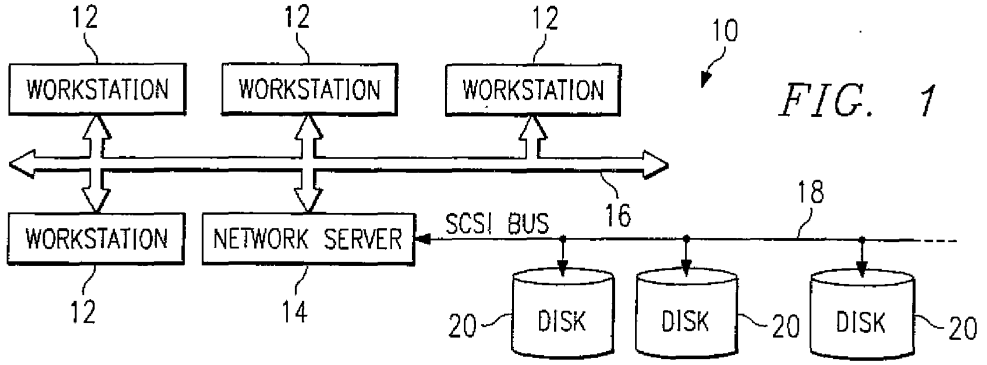


FIG. 1

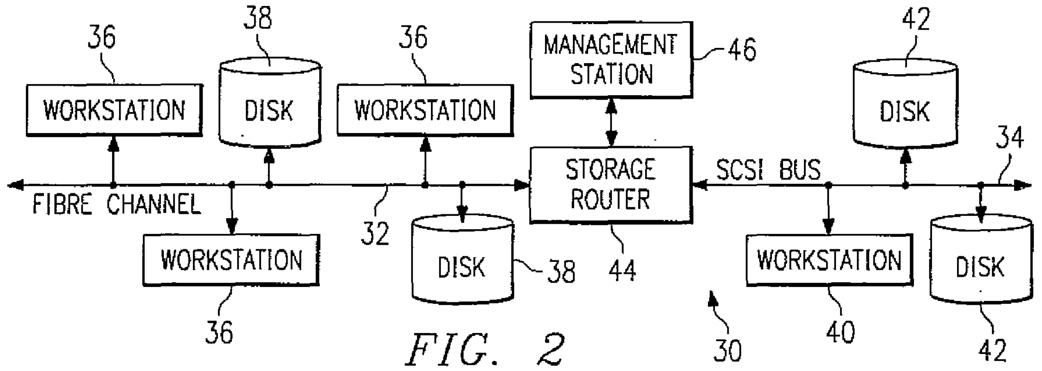


FIG. 2

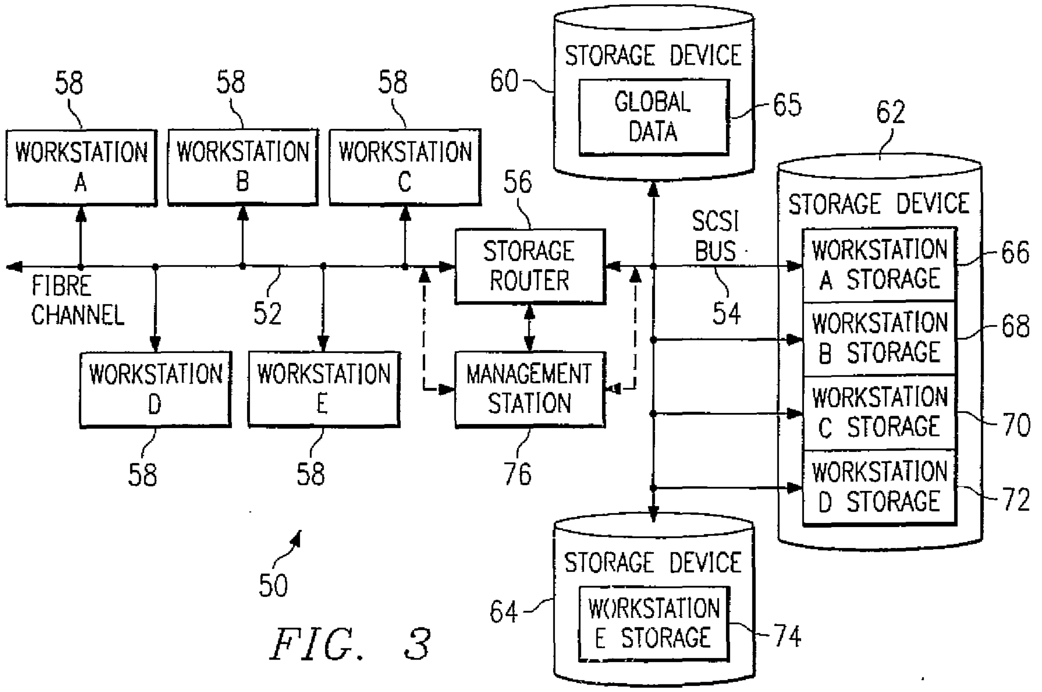


FIG. 3

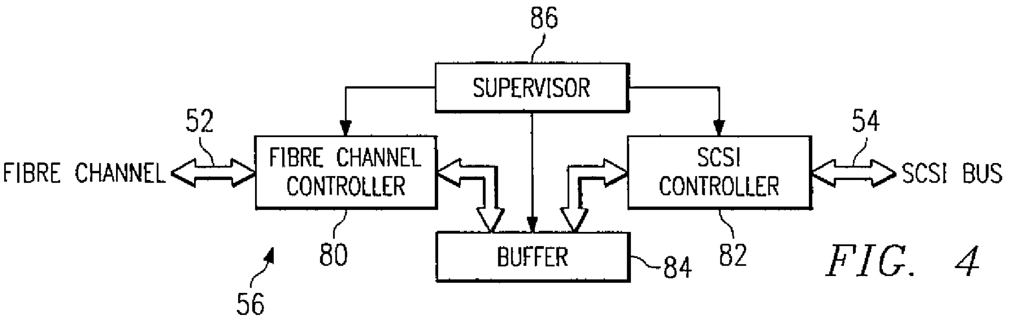

86
SUPERVISOR
52
FIBRE CHANNEL
FIBRE CHANNEL CONTROLLER
SCSI CONTROLLER
54
SCSI BUS
BUFFER
84
80
82
56

FIG. 4

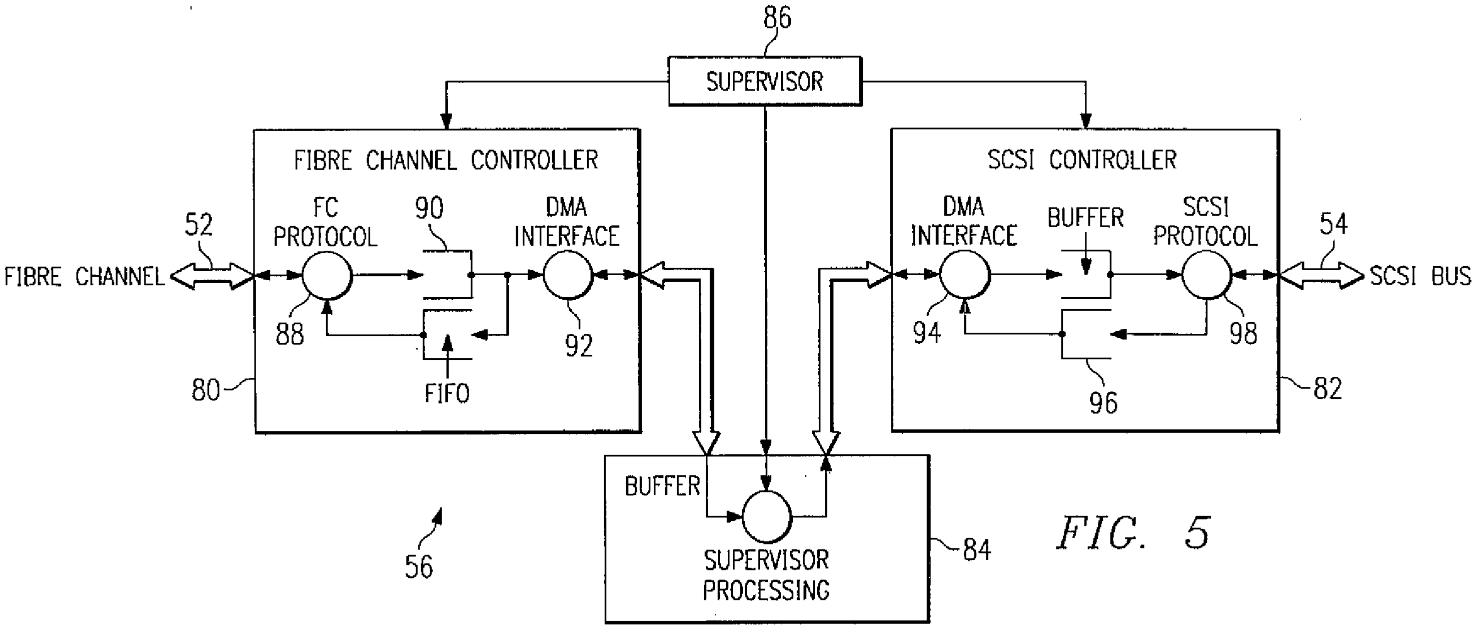

86
SUPERVISOR
FIBRE CHANNEL CONTROLLER
FC PROTOCOL
90
DMA INTERFACE
52
FIBRE CHANNEL
88
92
FIFO
80
SCSI CONTROLLER
DMA INTERFACE
BUFFER
SCSI PROTOCOL
54
SCSI BUS
94
98
96
82
BUFFER
SUPERVISOR PROCESSING
84
56

FIG. 5

PTO/SB/06 (12-04)

Date: **01/20/10**

Approved for use through 7/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875	Application or Docket Number 12/690,592

APPLICATION AS FILED – PART I

	(Column 1)	(Column 2)	SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
FOR	NUMBER FILED	NUMBER EXTRA	RATE ($)	FEE ($)		RATE ($)	FEE ($)
BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	**330**
SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	**540**
EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	**220**
TOTAL CLAIMS (37 CFR 1.16(i))	**53** minus 20 =	**33**	x$26		OR	x$52	**1716**
INDEPENDENT CLAIMS (37 CFR 1.16(h))	**3** minus 3 =	*	x$110			x$220	
APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $260 ($130 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR						
MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))			195			390	
			TOTAL			TOTAL	**2806**

* If the difference in column 1 is less than zero, enter "0" in column 2.

APPLICATION AS AMENDED – PART II

		(Column 1)		(Column 2)	(Column 3)	SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
AMENDMENT A		CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE ($)	ADDI-TIONAL FEE ($)		RATE ($)	ADDI-TIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=	X =		OR	X =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X =		OR	X =	
	Application Size Fee (37 CFR 1.16(s))									
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))					N/A		OR	N/A	
						TOTAL ADD'T FEE		OR	TOTAL ADD'T FEE	

		(Column 1)		(Column 2)	(Column 3)			OR		
AMENDMENT B		CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE ($)	ADDI-TIONAL FEE ($)		RATE ($)	ADDI-TIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=	X =		OR	X =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X =		OR	X =	
	Application Size Fee (37 CFR 1.16(s))									
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))					N/A		OR	N/A	
						TOTAL ADD'T FEE		OR	TOTAL ADD'T FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Paten and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING or 371(c) DATE	GRP ART UNIT	FIL FEE REC'D	ATTY.DOCKET.NO	TOT CLAIMS	IND CLAIMS
12/690,592	01/20/2010	2111	2806	CROSS1120-33	53	3

CONFIRMATION NO. 8115

44654
SPRINKLE IP LAW GROUP
1301 W. 25TH STREET
SUITE 408
AUSTIN, TX 78705

FILING RECEIPT

OC000000039986152

Date Mailed: 02/03/2010

Receipt is acknowledged of this non-provisional patent application. The application will be taken up for examination in due course. Applicant will be notified as to the results of the examination. Any correspondence concerning the application must include the following identification information: the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. **If an error is noted on this Filing Receipt, please submit a written request for a Filing Receipt Correction. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections**

Applicant(s)

Geoffrey B. Hoese, Austin, TX;
Jeffry T. Russell, Cibolo, TX;

Power of Attorney: The patent practitioners associated with Customer Number 44654

Domestic Priority data as claimed by applicant

This application is a CON of 12/552,885 09/02/2009
which is a CON of 11/851,724 09/07/2007
which is a CON of 11/442,878 05/30/2006 ABN *
which is a CON of 11/353,826 02/14/2006 PAT 7,340,549
which is a CON of 10/658,163 09/09/2003 PAT 7,051,147
which is a CON of 10/081,110 02/22/2002 PAT 6,789,152
which is a CON of 09/354,682 07/15/1999 PAT 6,421,753
which is a CON of 09/001,799 12/31/1997 PAT 5,941,972
(*)Data provided by applicant is not consistent with PTO records.

Foreign Applications

If Required, Foreign Filing License Granted: 02/02/2010

The country code and number of your priority application, to be used for filing abroad under the Paris Convention, is **US 12/690,592**

Projected Publication Date: 05/13/2010

Non-Publication Request: No

Early Publication Request: No

Title

STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE

Preliminary Class

710

PROTECTING YOUR INVENTION OUTSIDE THE UNITED STATES

Since the rights granted by a U.S. patent extend only throughout the territory of the United States and have no effect in a foreign country, an inventor who wishes patent protection in another country must apply for a patent in a specific country or in regional patent offices. Applicants may wish to consider the filing of an international application under the Patent Cooperation Treaty (PCT). An international (PCT) application generally has the same effect as a regular national patent application in each PCT-member country. The PCT process **simplifies** the filing of patent applications on the same invention in member countries, but **does not result** in a grant of "an international patent" and does not eliminate the need of applicants to file additional documents and fees in countries where patent protection is desired.

Almost every country has its own patent law, and a person desiring a patent in a particular country must make an application for patent in that country in accordance with its particular laws. Since the laws of many countries differ in various respects from the patent law of the United States, applicants are advised to seek guidance from specific foreign countries to ensure that patent rights are not lost prematurely.

Applicants also are advised that in the case of inventions made in the United States, the Director of the USPTO must issue a license before applicants can apply for a patent in a foreign country. The filing of a U.S. patent application serves as a request for a foreign filing license. The application's filing receipt contains further information and guidance as to the status of applicant's license for foreign filing.

Applicants may wish to consult the USPTO booklet, "General Information Concerning Patents" (specifically, the section entitled "Treaties and Foreign Patents") for more information on timeframes and deadlines for filing foreign patent applications. The guide is available either by contacting the USPTO Contact Center at 800-786-9199, or it can be viewed on the USPTO website at http://www.uspto.gov/web/offices/pac/doc/general/index.html.

For information on preventing theft of your intellectual property (patents, trademarks and copyrights), you may wish to consult the U.S. Government website, http://www.stopfakes.gov. Part of a Department of Commerce initiative, this website includes self-help "toolkits" giving innovators guidance on how to protect intellectual property in specific countries such as China, Korea and Mexico. For questions regarding patent enforcement issues, applicants may call the U.S. Government hotline at 1-866-999-HALT (1-866-999-4158).

LICENSE FOR FOREIGN FILING UNDER

Title 35, United States Code, Section 184

Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

The grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Bureau of Industry and Security, Department of Commerce (15 CFR parts 730-774); the Office of Foreign AssetsControl, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371(C) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
12/690,592	01/20/2010	Geoffrey B. Hoese	CROSS1120-33

CONFIRMATION NO. 8115

POA ACCEPTANCE LETTER

OC000000039985414

44654
SPRINKLE IP LAW GROUP
1301 W. 25TH STREET
SUITE 408
AUSTIN, TX 78705

Date Mailed: 02/03/2010

NOTICE OF ACCEPTANCE OF POWER OF ATTORNEY

This is in response to the Power of Attorney filed 01/20/2010.

The Power of Attorney in this application is accepted. Correspondence in this application will be mailed to the above address as provided by 37 CFR 1.33.

/abirhane/

Office of Data Management, Application Assistance Unit (571) 272-4000, or (571) 272-4200, or 1-888-786-0101

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING OR 371(C) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
12/690,592	01/20/2010	Geoffrey B. Hoese	CROSS1120-33

CONFIRMATION NO. 8115

PUBLICATION NOTICE

OC000000041607195

44654
SPRINKLE IP LAW GROUP
1301 W. 25TH STREET
SUITE 408
AUSTIN, TX 78705

Title:STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE

Publication No.US-2010-0121993-A1
Publication Date:05/13/2010

NOTICE OF PUBLICATION OF APPLICATION

The above-identified application will be electronically published as a patent application publication pursuant to 37 CFR 1.211, et seq. The patent application publication number and publication date are set forth above.

The publication may be accessed through the USPTO's publically available Searchable Databases via the Internet at www.uspto.gov. The direct link to access the publication is currently http://www.uspto.gov/patft/.

The publication process established by the Office does not provide for mailing a copy of the publication to applicant. A copy of the publication may be obtained from the Office upon payment of the appropriate fee set forth in 37 CFR 1.19(a)(1). Orders for copies of patent application publications are handled by the USPTO's Office of Public Records. The Office of Public Records can be reached by telephone at (703) 308-9726 or (800) 972-6382, by facsimile at (703) 305-8759, by mail addressed to the United States Patent and Trademark Office, Office of Public Records, Alexandria, VA 22313-1450 or via the Internet.

In addition, information on the status of the application, including the mailing date of Office actions and the dates of receipt of correspondence filed in the Office, may also be accessed via the Internet through the Patent Electronic Business Center at www.uspto.gov using the public side of the Patent Application Information and Retrieval (PAIR) system. The direct link to access this status information is currently http://pair.uspto.gov/. Prior to publication, such status information is confidential and may only be obtained by applicant using the private side of PAIR.

Further assistance in electronically accessing the publication, or about PAIR, is available by calling the Patent Electronic Business Center at 1-866-217-9197.

Office of Data Managment, Application Assistance Unit (571) 272-4000, or (571) 272-4200, or 1-888-786-0101

INFORMATION DISCLOSURE STATEMENT BY APPLICANT					
				Application Number	12/690,592
				Filing Date	01/20/2010
				First Named Inventor	Geoffrey B. Hoese
				Group Art Unit	2111
				Examiner Name	Unknown
Sheet	1	of	9	Attorney Docket Number	CROSS1120-33

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number Number-Kind Code *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	A1	3,082,406	3/19/1963	L.D. Stevens	
	A2	4,092,732	5/30/1978	Ouchi	
	A3	4,170,415	10/9/1979	Lemeshewsky, et al.	
	A4	4,415,970	11/15/1983	Swenson, et al.	
	A5	4,455,605	6/19/1984	Cormier, et al.	
	A6	4,504,927	3/12/1985	Callan	
	A7	4,533,996	8/6/1985	Gartung, et al.	
	A8	4,573,152	2/25/1986	Greene, et al.	
	A9	4,603,380	7/29/1986	Easton, et al.	
	A10	4,620,295	10/28/1986	Aiden, Jr.	
	A11	4,644,462	2/17/1987	Matsubara, et al.	
	A12	4,695,948	9/22/1987	Blevins, et al.	
	A13	4,697,232	9/29/1987	Brunelle, et al.	
	A14	4,715,030	12/22/1987	Koch, et al.	
	A15	4,751,635	6/14/1988	Kret	
	A16	4,787,028	11/22/1988	Finforck, et al.	
	A17	4,807,180	2/21/1989	Takeuchi, et al.	
	A18	4,811,278	3/7/1989	Bean, et al.	
	A19	4,821,179	4/11/1989	Jensen, et al.	
	A20	4,825,406	4/25/1989	Bean, et al.	
	A21	4,827,411	5/2/1989	Arrowood, et al.	
	A22	4,835,674	5/30/1989	Collins, et al.	
	A23	4,845,722	7/4/1989	Kent et al.	
	A24	4,864,532	9/5/1989	Reeve, et al.	
	A25	4,897,874	1/30/1990	Lidensky, et al.	
	A26	4,947,367	8/7/1990	Chang, et al.	
	A27	4,961,224	10/2/1990	Yung	
	A28	5,072,378	12/10/1991	Manka	
	A29	5,077,732	12/31/1991	Fischer, et al.	
	A30	5,077,736	12/31/1991	Dunphy, Jr., et al.	
	A31	5,124,987	6/23/1992	Milligan, et al.	
	A32	5,155,845	10/13/1992	Beal, et al.	
	A33	5,163,131	11/10/1992	Row, et al.	
	A34	5,185,876	2/9/1993	Nguyen, et al.	
	A35	5,193,168	3/9/1993	Corrigan, et al.	
	A36	5,193,184	3/9/1993	Belsan, et al.	
	A37	5,202,856	4/13/1993	Glider, et al.	
	A38	5,210,866	5/11/1993	Milligan, et al.	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

Application Number	12/690,592
Filing Date	01/20/2010
First Named Inventor	Geoffrey B. Hoese
Group Art Unit	2111
Examiner Name	Unknown
Attorney Docket Number	CROSS1120-33

Sheet 2 of 9

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number	Publication Date	Name of Patentee or	Pages, Columns, Lines Where Relevant Passages or Figures Appear
		Number-Kind Code *(if known)*	MM-DD-YYYY	Applicant of Cited Document	
	A39	5,212,785	5/18/1993	Powers, et al.	
	A40	5,214,778	5/25/1993	Glider, et al.	
	A41	5,226,143	7/6/1993	Baird, et al.	
	A42	5,239,632	8/24/1993	Larner	
	A43	5,239,643	8/24/1993	Blount, et al.	
	A44	5,239,654	8/24/1993	Ing-Simmons, et al.	
	A45	5,247,638	9/21/1993	O'Brien, et al.	
	A46	5,247,692	9/21/1993	Fujimura	
	A47	5,257,386	10/26/1993	Saito	
	A48	5,297,262	3/22/1994	Cox, et al.	
	A49	5,301,290	4/5/1994	Tetzlaff, et al.	
	A50	5,315,657	5/24/1994	Abadi, et al.	
	A51	5,317,693	5/31/1994	Elko, et al.	
	A52	5,331,673	7/19/1994	Elko, et al.	
	A53	5,347,384	9/13/1994	McReynolds, et al.	
	A54	5,355,453	10/11/1994	Glider, et al.	
	A55	5,361,347	11/1/1994	Glider, et al.	
	A56	5,367,646	11/22/1994	Pardillos, et al.	
	A57	5,379,385	1/3/1995	Shomler	
	A58	5,379,398	1/3/1995	Cohn, et al.	
	A59	5,388,243	2/7/1995	Glider, et al.	
	A60	5,388,246	2/7/1995	Kasi	
	A61	5,394,402	2/28/1995	Ross, et al.	
	A62	5,394,526	2/28/1995	Crouse et al.	
	A63	5,396,596	3/7/1995	Hashemi, et al.	
	A64	5,403,639	4/4/1995	Belsan, et al.	
	A65	5,410,667	4/25/1995	Belsan, et al.	
	A66	5,410,697	4/25/1995	Baird, et al.	
	A67	5,414,820	10/9/1995	McFarland, et al.	
	A68	5,416,915	5/16/1995	Mattson, et al.	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT		Application Number	**12/690,592**
		Filing Date	**01/20/2010**
		First Named Inventor	**Geoffrey B. Hoese**
		Group Art Unit	**2111**
		Examiner Name	**Unknown**
Sheet 3	of 9	Attorney Docket Number	**CROSS1120-33**

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number Number-Kind Code *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	A69	5,418,909	5/23/1995	Jachowski, et al.	
	A70	5,420,988	5/30/1995	Elliott	
	A71	5,423,026	6/6/1995	Cook, et al.	
	A72	5,423,044	6/6/1995	Sutton, et al.	
	A73	5,426,637	6/20/1995	Derby, et al	
	A74	5,430,855	7/4/1995	Wash, et al.	
	A75	5,450,570	9/12/1995	Richek, et al.	
	A76	5,452,421	9/19/1995	Beardsley, et al.	
	A77	5,459,857	10/17/1995	Ludlam, et al.	
	A78	5,463,754	10/31/1995	Beausoleil, et al.	
	A79	5,465,382	11/7/1995	Day, III, et al.	
	A80	5,469,576	11/21/1995	Dauerer, et al.	
	A81	5,471,609	11/28/1995	Yudenfriend, et al.	
	A82	5,487,077	1/23/1996	Hassner, et al.	
	A83	5,491,812	2/13/1996	Pisello, et al.	
	A84	5,495,474	2/27/1996	Olnowich, et al.	
	A85	5,496,576	3/5/1996	Jeong	
	A86	5,504,857	4/2/1996	Baird, et al.	
	A87	5,507,032	4/9/1996	Kimura	
	A88	5,511,169	4/23/1996	Suda	
	A89	5,519,695	5/21/1996	Purohit, et al.	
	A90	5,530,845	6/25/1996	Hiatt, et al.	
	A91	5,535,352	7/9/1996	Bridges, et al.	
	A92	5,537,585	7/16/1996	Blickerstaff, et al.	
	A93	5,544,313	8/6/1996	Shachnai, et al.	
	A94	5,548,791	8/20/1996	Casper, et al.	
	A95	5,564,019	10/8/1996	Beausoleil, et al	
	A96	5,568,648	10/22/1996	Coscarella, et al.	
	A97	5,581,709	12/3/1996	Ito, et al.	
	A98	5,581,714	12/3/1996	Amini, et al.	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

Application Number	12/690,592
Filing Date	01/20/2010
First Named Inventor	Geoffrey B. Hoese
Group Art Unit	2111
Examiner Name	Unknown
Attorney Docket Number	CROSS1120-33

Sheet	4	of	9

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number	Publication Date	Name of Patentee or	Pages, Columns, Lines Where Relevant Passages or Figures Appear
		Number-Kind Code *(if known)*	MM-DD-YYYY	Applicant of Cited Document	
	A99	5,581,724	12/3/1996	Belsan et al.	
	A100	5,596,562	6/21/1997	Chen	
	A101	5,596,736	1/21/1997	Kerns	
	A102	5,598,541	1/28/1997	Malladi	
	A103	5,613,082	3/18/1997	Brewer, et al.	
	A104	5,621,902	4/15/1997	Cases, et al.	
	A105	5,632,012	5/20/1997	Belsan, et al.	
	A106	5,634,111	5/27/1997	Oeda, et al.	
	A107	5,638,518	6/10/1997	Malladi	
	A108	5,642,515	6/24/1997	Jones, et al.	
	A109	5,659,756	8/19/1997	Hefferon, et al.	
	A110	5,664,107	9/2/1997	Chatwanni, et al.	
	A111	5,680,556	10/21/1997	Begun, et al.	
	A112	5,684,800	11/4/1997	Dobbins, et al.	
	A113	5,701,491	12/23/1997	Dunn, et al.	
	A114	5,712,976	1/27/1998	Falcon, et al.	
	A115	5,727,218	3/10/1998	Hotchkin	
	A116	5,729,705	3/17/1998	Weber	
	A117	5,743,847	4/28/1998	Nakamura, et al.	
	A118	5,748,924	5/5/1998	Llorens, et al.	
	A119	5,571,971	5/12/1998	Dobbins, et al.	
	A120	5,751,975	5/12/1998	Gillespie, et al.	
	A121	5,764,931	6/9/1998	Schmahl, et al.	
	A122	5,768,623	6/16/1998	Judd, et al.	
	A123	5,774,683	6/30/1998	Gulick	
	A124	5,778,411	7/7/1998	DeMoss	
	A125	5,781,715	7/14/1998	Sheu	
	A126	5,802,278	9/1/1998	Isfeld, et al.	
	A127	5,805,816	9/8/1998	Picazo, Jr., et al.	
	A128	5,805,920	9/8/1998	Sprenkle, et al.	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Application Number	**12/690,592**
				Filing Date	**01/20/2010**
				First Named Inventor	**Geoffrey B. Hoese**
				Group Art Unit	**2111**
				Examiner Name	**Unknown**
Sheet	5	of	9	Attorney Docket Number	**CROSS1120-33**

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number Number-Kind Code *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	A129	5,809,328	9/15/1998	Nogales, et al.	
	A130	5,812,754	9/22/1998	Lui, et al.	
	A131	5,819,054	10/6/1998	Ninomiya, et al.	
	A132	5,825,772	10/20/1998	Dobbins, et al.	
	A133	5,835,496	11/10/1998	Yeung, et al.	
	A134	5,845,107	12/1/1998	Fisch, et al.	
	A135	5,848,251	12/8/1998	Lomelino, et al.	
	A136	5,857,080	10/5/1999	Jander, et al.	
	A137	5,860,137	1/12/1999	Raz, et al.	
	A138	5,864,653	1/26/1999	Tavallaei, et al.	
	A139	5,867,648	2/2/1999	Foth, et al.	
	A140	5,884,027	3/16/1999	Garbus, et al.	
	A141	5,889,952	3/30/1999	Hunnicutt, et al.	
	A142	5,913,045	6/15/1999	Gillespie, et al.	
	A143	5,923,557	7/13/1999	Eidson	
	A144	5,933,824	8/3/1999	DeKoning, et al.	
	A145	5,935,205	8/10/1999	Murayama, et al.	
	A146	5,935,260	8/10/1999	Ofer	
	A147	5,941,969	8/24/1999	Ram, et al.	
	A148	5,941,972	8/24/1999	Hoese, et al.	
	A149	5,946,308	8/31/1999	Dobbins, et al.	
	A150	5,953,511	9/14/1999	Sescilia, et al.	
	A151	5,959,994	9/28/1999	Boggs, et al.	
	A152	5,963,556	10/5/1999	Varghese, et al.	
	A153	5,974,530	10/26/1999	Young	
	A154	5,978,379	11/2/1999	Chan, et al.	
	A155	5,978,875	11/2/1999	Asano, et al.	
	A156	5,991,797	11/23/1999	Futral, et al.	
	A157	6,000,020	12/7/1999	Chin, et al.	
	A158	6,041,058	3/21/2000	Flanders, et al.	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

Application Number	12/690,592
Filing Date	01/20/2010
First Named Inventor	Geoffrey B. Hoese
Group Art Unit	2111
Examiner Name	Unknown
Attorney Docket Number	CROSS1120-33

Sheet 6 of 9

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number Number-Kind Code *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	A159	6,021,451	2/1/2000	Bell, et al.	
	A160	6,029,168	2/22/2000	Frey	
	A161	6,032,269	2/29/2000	Renner, Jr.	
	A162	6,041,381	3/21/2000	Hoese	
	A163	6,055,603	4/25/2000	Ofer, et al.	
	A164	6,065,087	5/16/2000	Keaveny, et al.	
	A165	6,070,253	5/30/2000	Tavallaei, et al.	
	A166	6,073,209	6/6/2000	Bergsten	
	A167	6,073,218	6/6/2000	DeKoning, et al.	
	A168	6,075,863	6/13/2000	Krishnan, et al.	
	A169	6,081,849	6/27/2000	Born, et al.	
	A170	6,098,128	8/1/2000	Velez-McCaskey et al.	
	A171	6,098,149	8/1/2000	Ofer, et al.	
	A172	6,108,684	8/22/2000	DeKoning, et al	
	A173	6,118,766	9/12/2000	Akers	
	A174	6,131,119	10/10/2000	Fukui	
	A175	6,134,617	10/17/2000	Weber	
	A176	6,141,737	10/31/2000	Krantz, et al.	
	A177	6,145,006	11/7/2000	Vishlitsky, et al.	
	A178	6,147,976	11/14/2000	Shand, et al.	
	A179	6,147,995	11/14/2000	Dobbins, et al.	
	A180	6,148,004	11/14/2000	Nelson, et al.	
	A181	6,173,399	1/9/2001	Gilbrech	
	A182	6,185,203	2/6/2001	Berman	
	A183	6,202,153	3/13/2001	Diamant, et al.	
	A184	6,209,023	3/27/2001	Dimitroff, et al.	
	A185	6,219,771	4/17/2001	Kikuchi, et al.	
	A186	6,223,266	4/24/2001	Sartore	
	A187	6,230,218	5/8/2001	Casper, et al.	
	A188	6,243,827	6/5/2001	Renner, Jr.	

		Date Considered	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Application Number	**12/690,592**
				Filing Date	**01/20/2010**
				First Named Inventor	**Geoffrey B. Hoese**
				Group Art Unit	**2111**
				Examiner Name	**Unknown**
Sheet	7	of	9	Attorney Docket Number	**CROSS1120-33**

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number Number-Kind Code *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	A189	6,260,120	7/10/2001	Blumenau, et al.	
	A190	6,268,789	7/31/2001	Diamant, et al.	
	A191	6,308,247	10/23/2001	Ackerman	
	A192	6,330,629	12/11/2001	Kondo, et al.	
	A193	6,330,687	12/11/2001	Griffith	
	A194	6,341,315	1/22/2002	Arroyo, et al.	
	A195	6,343,324	1/29/2002	Hubis, et al.	
	A196	6,363,462	3/26/2002	Bergsten	
	A197	6,401,170	6/4/2002	Griffith, et al.	
	A198	6,421,753	7/16/2002	Hoese, et al.	
	A199	6,425,035	7/23/2002	Hoese, et al.	
	A200	6,425,036	7/23/2002	Hoese, et al.	
	A201	6,425,052	6/23/2002	Hashemi	
	A202	6,453,345	9/17/2002	Trcka, et al.	
	A203	6,484,245	11/19/2002	Sanada, et al.	
	A204	6,529,996	3/4/2003	Nguyen, et al.	
	A205	6,547,576	4/15/2003	Peng, et al.	
	A206	6,560,750	5/6/2003	Chien, et al.	
	A207	6,563,701	5/13/2003	Peng, et al.	
	A208	6,775,693	8/10/2004	Adams	
	A209	6,792,602	9/14/2004	Lin, et al.	
	A210	6,820,212	11/16/2004	Duchesne, et al.	
	A211	6,854,027	2/8/2005	Hsu, et al.	
	A212	6,862,637	3/1/2005	Stupar	
	A213	6,874,043	3/29/2005	Treggiden	
	A214	6,874,100	3/29/2005	Rauscher	
	A215	6,910,083	6/21/2005	Hsu, et al.	
	A216	7,065,076	6/20/2006	Nemazie	
	A217	7,127,668	10/24/2006	McBryde, et al.	
	A218	7,133,965	11/7/2006	Chien	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT					
				Application Number	**12/690,592**
				Filing Date	**01/20/2010**
				First Named Inventor	**Geoffrey B. Hoese**
				Group Art Unit	**2111**
				Examiner Name	**Unknown**
Sheet	**8**	of	**8**	Attorney Docket Number	**CROSS1120-33**

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number Number-Kind Code *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	A219	7,188,111	3/6/2007	Chen, et al.	
	A220	7,216,225	5/8/2007	Haviv, et al.	
	A221	7,251,248	7/31/2007	Trossell, et al.	
	A222	7,281,072	10/9/2007	Liu, et al.	
	A223	D470,486	2/18/2003	Cheng	
	A224	2002/0083221	6/27/2002	Tsai, et al.	
	A225	2006/0277326	12/7/2006	Tsai, et al.	
	A226	2006/0294416	12/28/2006	Tsai, et al.	
	A227	2006/0218322	09/2006	Hoese, et al.	

FOREIGN PATENT DOCUMENTS

Examiner Initials	Cite No.	**Foreign Patent Document** Country Code-Number-Kind Code *(if known)*	Publication Date MM-DD-YYYY (Number 43)	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	B1	GB 2296798 A	7/10/1996	Spring Consultants Limited	
	B2	GB 2297636 A	8/7/1996	Spring Consultants Limited	
	B3	JP 8-230895	9/10/1996	Kikuchi, et al.	
	B4	EP 0810530 A2	12/3/1997	Sun Microsystems, Inc.	
	B5	EP 0827059 A2	3/4/1998	NEC Corporation	
	B6	WO 99/34297 A1	7/8/1999	Crossroads Systems, Inc.	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

Application Number	12/690,592
Filing Date	01/20/2010
First Named Inventor	Geoffrey B. Hoese
Group Art Unit	2111
Examiner Name	Unknown
Attorney Docket Number	CROSS1120-33

Sheet	9	of	9

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number	Publication Date	Name of Patentee or	Pages, Columns, Lines Where Relevant Passages or Figures Appear
		Number-Kind Code *(if known)*	MM-DD-YYYY	Applicant of Cited Document	

FOREIGN PATENT DOCUMENTS

Examiner Initials	Cite No.	Foreign Patent Document	Publication Date MM-DD-YYYY (Number 43)	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines Where Relevant Passages or Figures Appear
		Country Code-Number-Kind Code *(if known)*			
	B7	GB 2341715	03/22/2000	SpringTek Limited	
	B8	JP 6301607	10/28/1994	Hitachi Ltd.	
	B9	WO 98/36357	08/20/1998	Transwitch Corporation	
	B10	WO 91/03788	3/21/1991	Auspex Systems, Inc.	
	B11	WO 1997033227	8/4/1998	Nippon Telegraph & AMP; Telephone Corp.	
	B12	AU 647414	3/24/1994	Auspex Systems, Inc.	
	B13	AU 670376	7/11/1996	Auspex Systems, Inc.	
	B14	CA 2066443	10/21/2003	Auspex Systems, Inc.	
	B15	EP 0490973	2/25/1998	Auspex Systems, Inc.	
	B16	IL 095447	5/30/1994	Auspex Systems, Inc	
	B17	IL 107645	9/12/1996	Auspex Systems, Inc.	
	B18	JP 10097493	4/14/1998	Sun Microsyst. Inc.	
	B19	JP 1997251437	9/22/1997	Toshiba Corporation	
	B20	JP 1993181609	7/23/1993	NEC Corp.	
	B21	JP 1997185594	7/15/1997	Tandem Computers, Inc.	
	B22	JP 1995020994	1/24/1995	Hitachi, Ltd.	
	B23	JP 5502525	4/28/1993	Auspex Systems, Inc.	
	B24	JP 5181609	7/23/1993	Nippon Electric Co.	
	B25	JP 720994	1/24/1995	Hitachi Seisakusho Co., Ltd.	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	**12/690,592**
Filing Date	**01/20/2010**
First Named Inventor	**Geoffrey B. Hoese**
Group Art Unit	**2111**
Examiner Name	**Unknown**
Atty Docket Number	**CROSS1120-33**

Sheet 1 of 9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C4	Black Box, SCSI Fiberoptic Extender, Single-Ended, Product Insert, 2 pages, 1996	6/18/1905
	C5	Block-Based Distributed File Systems, Anthony J. McGregor, July 1997	
	C6	Compaq StorageWorks HSG80 Array Controller ACS Version 8.3 (Maintenance and Service Guide) 11/98	
	C7	Compaq StorageWorks HSG80 Array Controller ACS Version 8.3 (Configuration and CLI Reference Guide) 11/98	
	C8	CRD-5500, RAID DISK ARRAY CONTROLLER Product Insert, pp. 1-5	
	C9	CRD-5500, SCSI RAID CONTROLLER OEM Manual, Rev. 1.3, February 26, 1996, pp. 1-54	2/26/1996
	C10	CRD-5500, SCSI RAID CONTROLLER Users Manual, Rev. 1.3, November 21, 1996, pp. 10-92	11/21/1996
	C11	DIGITAL StorageWorks HSZ70 Array Controller HSOF Version 7.0 EK-SHZ70-RM.A01 CLI Reference Manual.	7/1/1997
	C12	DIGITAL Storage Works, HSZ70 Array Controller, HSOF Version 7.0 EK-HSZ70-CG. A01, Digital Equipment Corporation, Maynard, Massachusetts	7/1/1997
	C13	DIGITAL StorageWorks, Using Your HSZ70 Array Controller in a SCSI Controller Shelf (DS-BA356-M Series), User's Guide, pp. 1-1 through A-5 with index, January 1998.	1/1/1998
	C14	DIGITAL StorageWorks HSZ70 Array Controller HSOF Version 7.0 EK-HSZ70-SV. A01	1997-
	C15	DIGITAL StorageWorks HSG80 Array Controller ACS Version 8.0 (User's Guide 1/98	
	C16	DP5380 Asynchronous SCSI Interface, National Semiconductor Corporation, Arlington, TX, May 1989, pp. 1-32	
	C17	Emerson, "Encor Communications: Performance evaluation of switched fibre channel I/O system using--FCP for SCSI" February 1995, IEEE, pp. 479-484	2/1/1995
	C18	Fiber channel (FCS)/ATM internetworking: a design solution	
	C19	Fiber Channel storage interface for video-on-demand servers by Anazaloni, et al.	6/15/1905
	C20	Fibre Channel and ATM: The Physical Layers, Jerry Quam WESCON/94, published 27-29 September 1994. Pages 648-652.	
	C21	Gen5 S-Series XL System Guide Revision 1.01 by Chen	6/18/1905
	C22	Graphical User Interface for MAXSTRAT Gen5/Gen-S Servers User's guide 1.1	6/11/1996
	C23	High Performance Data transfers Using Network-Attached Peripherals at the national Storage Laboratory by Hyer	2/26/1993

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	12/690,592
Filing Date	01/20/2010
First Named Inventor	Geoffrey B. Hoese
Group Art Unit	2111
Examiner Name	Unknown
Atty Docket Number	CROSS1120-33

Sheet 2 of 9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C24	IFT-3000 SCSI to SCSI Disk array Controller Instruction Manual Revision 2.0 by Infotrend Technologies, Inc.	1995-
	C25	Implementing a Fibre Channel SCSI transport by Snively	1994-
	C26	"InfoServer 150--Installation and Owner's Guide", EK-INFSV-OM-001, Digital Equipment Corporation, Maynard, Massachusetts 1991, Chapters 1 and 2	
	C27	InfoServer 150VXT Photograph	
	C28	IBM Technical Publication: Guide to Sharing and Partitioning IBM Tape Library Dataservers, November 1996, pp. 1-256	11/1/1996
	C29	IBM Technical Publication: Magstar and IBM 3590 High Performance Tape Subsystem Technical Guide, November 1996, pp. 1-269	11/1/1996
	C30	Misc. Reference Manual Pages, SunOS 5.09	
	C31	Infoserver 100 System Operations Guide, First Edition Digital Equipment Corporation, 1990	
	C32	Johnson, D.B., et al., The Peregrine High Performance RPC System", Software-Practice and Experience, 23(2):201-221, Feb. 1993	
	C33	Local-Area networks for the IBM PC by Haugdahl	
	C34	New serial I/Os speed storage subsystems by Bursky	2/6/1995
	C35	Petal: Distributed Virtual Disks, Edward K. Lee and Chandramohan A. Thekkath, ACM SIGPLAN Notices, Volume 31, Issue 9, September 1996, pages 84-92.	
	C36	Pictures of internal components of the InfoServer 150, taken from http://bindarydinosaurs.couk/Museum/Digital/infoserver/infoserver.php in Nov. 2004.	
	C37	Raidtec FibreArray and Raidtec FlexArray UltraRAID Systems", Windows IT PRO Article, October 1997	
	C38	S.P. Joshi, "Ethernet controller chip interfaces with variety of 16-bit processors," electronic Design, Hayden Publishing Co., Inc., Rochelle Partk, NJ, October 14, 1982. pp 193-200	
	C39	Simplest Migration to Fibre Channel Technology" Article, Digital Equipment Corporation, November 10, 1997, published on PR Newswire	11/10/1997
	C40	Systems Architectures Using Fibre Channel, Roger Cummings, Twelfth IEEE Symposium on Mass Storage Systems, Copyright 1993 IEEE. Pages 251-256	
	C41	Office Action dated 01/21/03 for 10/174,720 (CROSS1120-8)	1/21/2003
	C42	Office Action dated 02/27/01 for 09/354,682 (CROSS1120-1)	2/27/2001
	C43	Office Action dated 08/11/00 for 09/354,682 (CROSS1120-1)	8/11/2000
	C44	Office Action dated 12/16/99 for 09/354,682 (CROSS1120-1)	12/16/1999
	C45	Office Action dated 11/06/02 for 10/023,786 (CROSS1120-4)	11/6/2002
	C46	Office Action dated 01/21/03 for 10/081,110 (CROSS1120-5)	1/21/2003

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	12/690,592
Filing Date	01/20/2010
First Named Inventor	Geoffrey B. Hoese
Group Art Unit	2111
Examiner Name	Unknown
Atty Docket Number	CROSS1120-33

Sheet	3	of	9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C47	Office Action in Ex Parte Reexamination 90/007,127, mailed February 7, 2005.	2/7/2005
	C48	Office Action in Ex Parte Reexamination 90/007,125, mailed February 7, 2005.	2/7/2005
	C49	Office Action in Ex Parte Reexamination 90/007,126, mailed February 7, 2005.	2/7/2005
	C50	Office Action in Ex Parte Reexamination 90/007,124, mailed February 7, 2005.	2/7/2005
	C51	Office Action in Ex Parte Reexamination 90/007,123, mailed February 7, 2005.	2/7/2005
	C52	European Office Action issued April 1, 2004 in Application No. 98966104.6-2413	4/1/2004
	C53	Office Action dated 1/27/2005 in 10/658,163 (CROSS1120-13)	1/27/2005
	C54	DIGITAL "System Support Addendum", SSA 40.78.01-A, AE-PNZJB-TE, pgs 1-3	4/1/1993
	C55	DIGITAL "Software Product Description", SSA 40.78.01, AE-PNZJB-TE, pgs 1-3	4/1/1993
	C56	DIGITAL EQUIPMENT CORPORATION, "InfoServer 100 Installation and Owner's Guide", Order Number EK-DIS1K-IN-001, First Edition	10/1/1990
	C57	DIGITAL EQUIPMENT CORPORATION, "InfoServer 100 System Operation Guide", Order Number EK-DIS1K-UG-001, First Edition, pgs i-Index 5	10/1/1990
	C58	ELLIOTT, "Working Draft American National Standard, Project T10/1562-D, Revision 5, pgs. i-432	7/9/2003
	C59	SATRAN, "Standards-Track," May 2001, iSCSI, pgs. 9-87	11/1/2000
	C60	SATRAN, et al. "IPS Internet Draft, iSCSI, pgs 1-8	11/1/2000
	C61	APT TECHNOLOGIES, INC., "Serial ATA: High Speed Serialized AT Attachment", Rev. 1.0a, pgs. 1-310	1/7/2003
	C63	Defendant's First Supplemental Trial Exhibit List, Crossroads Systems, Inc., v. Chaparral Network Storage, Inc., C.A. No. A-00CA-217-SS (W.D. Tex. 2001). **(CD-Rom)**	
	C64	Defendant's Third Supplemental Trial Exhibit List, Crossroads Systems, Inc. v. Pathlight Technology, Inc., C.A. No. A-00CA-248-SS (W.D. Tex. 2001) **(CD-Rom)**	
	C65	Plaintiff's Fourth Amended Trail Exhibit List, Crossroads Systems, Inc. v. Chaparral Network Storage, Inc, C.A. No. A-00CA-217-SS (W.D. Tex. 2001) **(CD-Rom)**	
	C66	Plaintiff's Revised Trial Exhibit List, Crossroads Systems, Inc. v. Pathlight Technology, Inc., C.A. No. A-00CA-248-SS (W.D. Tex. 2001). **(CD-Rom)**	
	C67	Trail Transcripts, Crossroads Systems, Inc. v. Chaparral Network Storage, Inc., C.A. No. A-00CA-217-SS (W.D. Tex. 2001) Day 1 -5 **(CD-Rom)**	
	C68	Trail Transcripts, Crossroads Systems, Inc. v. Pathlight Technology, Inc., C.A. No. A-00CA-248-SS (W.D. Tex. 2001). Day 1-4 **(CD-Rom)**	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	**12/690,592**
Filing Date	**01/20/2010**
First Named Inventor	**Geoffrey B. Hoese**
Group Art Unit	**2111**
Examiner Name	**Unknown**
Atty Docket Number	**CROSS1120-33**

Sheet 4 of 9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C72	Datasheet for CrossPoint 4100 Fibre Channel to SCSI Router (Dedek Ex 41 (ANCT 117-120)) (**CD-ROM** Chaparral Exhibits D012)	
	C73	Symbios Logic- Software Interface Specification Series 3 SCSI RAID Controller Software Release 02.xx (Engelbrecht Ex 2 (LSI 1421-1658)) (**CD-ROM** Chaparral Exhibits D013)	12/3/1997
	C74	Press Release- Symbios Logic to Demonstrate Strong Support for Fibre Channel at Fall Comdex (Engelbrecht 12 (LSI 2785-86)) (**CD-ROM** Chaparral Exhibits D016)	11/13/1996
	C75	OEM Datasheet on the 3701 Controller (Engelbrecht 13 (LSI 01837-38)) (**CD-ROM** Chaparral Exhibits D017)	6/17/1905
	C76	Nondisclosure Agreement Between Adaptec and Crossroads Dated 10/17/96 (Quisenberry Ex 25 (CRDS 8196)) (**CD-ROM** Chaparral Exhibits D020)	10/17/1996
	C77	Organizational Presentation on the External Storage Group (Lavan Ex 1 (CNS 182242-255)) (**CD-ROM** Chaparral Exhibits D021)	4/11/1996
	C78	Bridge Phase II Architecture Presentation (Lavan Ex 2 (CNS 182287-295)) (**CD-ROM** Chaparral Exhibits D022)	4/12/1996
	C79	Bridge. C, Bridge Between SCSI-2 and SCSI-3 FCP (Fibre Channel Protocol) (**CD-ROM** Chaparral Exhibits P214)	
	C80	Attendees/Action Items from 4/12/96 Meeting at BTC (Lavan Ex 3 (CNS 182241)) (**CD-ROM** Chaparral Exhibits D023)	4/12/1996
	C81	Brooklyn Hardware Engineering Requirements Documents, Revision 1.4 (Lavan Ex 4 (CNS 178188-211)) (**CD-ROM** Chaparral Exhibits D024) by Pecone	5/26/1996
	C82	Brooklyn Single-Ended SCSI RAID Bridge Controller Hardware OEM Manual, Revision 2.1 (Lavan EX 5 (CNS 177169-191)) (**CD-ROM** Chaparral Exhibits D025)	3/2/1996
	C83	Coronado Hardware Engineering Requirements Document, Revision 0.0 (Lavan Ex 7 (CNS 176917-932)) (**CD-ROM** Chaparral Exhibits D027) by O'Dell	9/30/1996
	C84	ESS/FPG Organization (Lavan Ex 8 (CNS 178639-652)) (**CD-ROM** Chaparral Exhibits D028)	12/6/1996
	C85	Adaptec MCS ESS Presents: Intelligent External I/O Raid Controllers "Bridge" Strategy (Lavan Ex 9 (CNS 178606-638)). (**CD-ROM** Chaparral Exhibits D029)	2/6/1996
	C86	AEC-7313 Fibre Channel Daughter Board (for Brooklyn) Engineering Specification, Revision 1.0 (Lavan Ex 10 (CNS 176830-850)) (**CD-ROM** Chaparral Exhibits D030)	2/27/1997
	C87	Bill of Material (Lavan Ex 14 (CNS 177211-214)) (**CD-ROM** Chaparral Exhibits D034)	7/24/1997
	C88	AEC-. 4412B, AEC-7412/B2 External RAID Controller Hardware 0EM Manual, Revision 2.0 (Lavan Ex 15 (CNS 177082-123)) (**CD-ROM** Chaparral Exhibits D035)	6/27/1997

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	**12/690,592**
Filing Date	**01/20/2010**
First Named Inventor	**Geoffrey B. Hoese**
Group Art Unit	**2111**
Examiner Name	**Unknown**
Atty Docket Number	**CROSS1120-33**

Sheet 5 of 9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C89	Coronado II, AEC-7312A Fibre Channel Daughter (for Brooklyn) Hardware Specification, Revision 1.2 (Lavan Ex 16 (CNS 177192-210)) (**CD-ROM** Chaparral Exhibits D036) by Tom Yang	7/18/1997
	C90	AEC-4412B, AEC7412/3B External RAID Controller Hardware OEM Manual, Revision 3.0. (Lavan Ex 17 (CNS 177124-165)) (**CD-ROM** Chaparral Exhibits D037)	8/25/1997
	C91	Memo Dated 8/15/97 to AEC-7312A Evaluation Unit Customers re: B001 Release Notes (Lavan Ex 18 (CNS 182878-879)) (**CD-ROM** Chaparral Exhibits D038)	8/15/1997
	C92	Brooklyn Main Board (AES-0302) MES Schedule (Lavan Ex I9 (CNS 177759-763)) (**CD-ROM** Chaparral Exhibits D039)	2/11/1997
	C93	News Release-Adaptec Adds Fibre Channel Option to its External RAID Controller Family (Lavan Ex 20 (CNS 182932-934)) (**CD-ROM** Chaparral Exhibits D040)	5/6/1997
	C94	AEC-4412B/7412B User's Guide, Rev. A (Lavan Ex 21) (**CD-ROM** Chaparral Exhibits D041)	6/19/1905
	C95	Data Book- AIC-7895 PCI Bus Master Single Chip SCSI Host Adapter (Davies Ex 1 (CNS 182944-64)) (**CD-ROM** Chaparral Exhibits D046)	5/21/1996
	C96	Data Book- AIC-1160 Fibre Channel Host Adapter ASIC (Davies Ex 2 (CNS 181800-825)) (**CD-ROM** Chaparral Exhibits D047)	6/18/1905
	C97	Viking RAID Software (Davies Ex 3 (CNS 180969-181026)) (**CD-ROM** Chaparral Exhibits D048)	6/18/1905
	C98	Header File with Structure Definitions (Davies Ex 4 (CNS 180009-018)) (**CD-ROM** Chaparral Exhibits D049)	8/8/1996
	C99	C++ SourceCode for the SCSI Command Handler (Davies Ex 5 (CNS 179136-168)) (**CD-ROM** Chaparral Exhibits D050)	8/8/1996
	C100	Header File Data Structure (Davies Ex 6 (CNS 179997-180008)) (**CD-ROM** Chaparral Exhibits D051)	1/2/1997
	C101	SCSI Command Handler (Davies Ex 7 (CNS 179676-719)) (**CD-ROM** Chaparral Exhibits D052)	1/2/1997
	C102	Coronado: Fibre Channel to SCSI Intelligent RAID Controller Product Brief (Kalwitz Ex I (CNS 182804-805)) (**CD-ROM** Chaparral Exhibits D053)	
	C103	Bill of Material (Kalwitz Ex 2 (CNS 181632-633)) (**CD-ROM** Chaparral Exhibits D054)	3/17/1997
	C104	Emails Dated 1/13-3/31/97 from P. Collins to Mo re: Status Reports (Kalwitz Ex 3 (CNS 182501-511)) (**CD-ROM** Chaparral Exhibits D055)	
	C105	Hardware Schematics for the Fibre Channel Daughtercard Coronado (Kalwitz Ex 4 (CNS 181639-648)) (**CD-ROM** Chaparral Exhibits D056)	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	**12/690,592**
Filing Date	**01/20/2010**
First Named Inventor	**Geoffrey B. Hoese**
Group Art Unit	**2111**
Examiner Name	**Unknown**
Atty Docket Number	**CROSS1120-33**

Sheet 6 of 9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C106	Adaptec Schematics re AAC-340 (Kalwitz Ex 14 CNS 177215-251)) (**CD-ROM** Chaparral Exhibits D057)	
	C107	Bridge Product Line Review (Manzanares Ex 3 (CNS 177307-336)) (**CD-ROM** Chaparral Exhibits D058)	
	C108	AEC Bridge Series Products-Adaptec External Controller RAID Products Pre-Release Draft, v.6 (Manzanares Ex 4 (CNS 174632-653)). (**CD-ROM** Chaparral Exhibits D059)	10/28/1997
	C109	Hewlett-Packard Roseville Site Property Pass for Brian Smith (Dunning Ex 14 (HP 489) (**CD-ROM** Chaparral Exhibits D078)	11/7/1996
	C110	Distribution Agreement Between Hewlett-Packard and Crossroads (Dunning Ex 15 (HP 326-33) (**CD-ROM** Chaparral Exhibits D079)	
	C111	HPFC-5000 Tachyon User's Manuel, First Edition (PTI 172419-839) (**CD-ROM** Chaparral Exhibits D084)	5/1/1996
	C112	X3T10 994D - (Draft) Information Technology: SCSI-3 Architecture Model, Rev. 1.8 (PTI 165977) (**CD-ROM** Chaparral Exhibits D087)	
	C113	X3T10 Project 1047D: Information Technology- SCSI-3 Controller Commands (SCC), Rev, 6c (PTI 166400-546) (**CD-ROM** Chaparral Exhibits D088)	9/3/1996
	C114	X3T10 995D- (Draft) SCSI-3 Primary Commands, Rev. 11 (Wanamaker Ex 5 (PTI 166050-229)) (**CD-ROM** Chaparral Exhibits D089)	11/13/1996
	C115	VBAR Volume Backup and Restore (CRDS 12200-202) (**CD-ROM** Chaparral Exhibits D099)	
	C116	Preliminary Product Literature for Infinity Commstor's Fibre Channel to SCSI Protocol Bridge (Smith Ex 11; Quisenberry Ex 31 (SPLO 428-30) (**CD-ROM** Chaparral Exhibits D143)	8/19/1996
	C117	Letter dated 7/12/96 from J. Boykin to B. Smith re: Purchase Order for Evaluation Units from Crossroads (Smith Ex 24) CRDS 8556-57) (**CD-ROM** Chaparral Exhibits D144)	7/12/1996
	C118	CrossPoint 4100 Fibre Channel to SCSI Router Preliminary Datasheet (Hulsey Ex 9 (CRDS 16129-130)) (**CD-ROM** Chaparral Exhibits D145)	11/1/1996
	C119	CrossPoint 4400 Fibre Channel to SCSI Router Preliminary Datasheet (Bardach Ex. 9, Quisenberry Ex 33 (CRDS 25606-607)) (**CD-ROM** Chaparral Exhibits D153)	11/1/1996
	C120	Fax Dated 07/22/96 from L. Petti to B. Smith re: Purchase Order from Data General for FC2S Fibre to Channel SCSI Protocol Bridge Model 11 (Smith Ex 25; Quisenberry Ex 23; Bardach Ex 11 (CRDS 8552-55; 8558) (**CD-ROM** Chaparral Exhibits D155)	7/22/1996
	C121	Email Dated 12/20/96 from J. Boykin to B. Smith re: Purchase Order for Betas in February and March (Hoese Ex 16, Quisenberry Ex 25; Bardach Ex 12 (CRDS 13644-650) (**CD-ROM** Chaparral Exhibits D156)	12/20/1996
	C122	Infinity Commstor Fibre Channel Demo for Fall Comdex, 1996 (Hoese Ex 15, Bardach Ex 13 (CRDS 27415) (**CD-ROM** Chaparral Exhibits D157)	
Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	**12/690,592**
Filing Date	**01/20/2010**
First Named Inventor	**Geoffrey B. Hoese**
Group Art Unit	**2111**
Examiner Name	**Unknown**
Atty Docket Number	**CROSS1120-33**

Sheet 7 of 9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C123	Fax Dated 12/19/96 from B. Bardach to T. Rarich re: Purchase Order Information (Bardach Ex. 14; Smith Ex 16 (CRDS 4460)) (**CD-ROM** Chaparral Exhibits D158)	12/19/1996
	C124	Miscellaneous Documents Regarding Comdex (Quisenberry Ex 2 (CRDS 27415-465)) (**CD-ROM** Chaparral Exhibits D165)	
	C125	CrossPoint 4100 Fibre Channel to SCSI Router Preliminary Datasheet (Quisenberry) Ex 3 (CRDS 4933-34) (**CD-ROM** Chaparral Exhibits D166) (CD-ROM Chaparral Exhibits D166)	
	C126	CrossPoint 4400 Fibre to Channel to SCSI Router Preliminary Datasheet; Crossroads Company and Product Overview (Quisenberry Ex 4 (CRDS 25606; 16136)) (**CD-ROM** Chaparral Exhibits D167)	
	C127	Crossroads Purchase Order Log (Quisenberry Ex 9 (CRDS 14061-062)) (**CD-ROM** Chaparral Exhibits D172)	
	C128	RAID Manager 5 with RDAC 5 for UNIX V.4 User's Guide (LSI-01854) (**CD-ROM** Chaparral Exhibits P062)	9/1/1996
	C129	Letter dated May 12, 1997 from Alan G. Leal to Barbara Bardach enclosing the original OEM License and Purchase Agreement between Hewlett-Package Company and Crossroads Systems, Inc. (CRDS 02057) (**CD-ROM** Chaparral Exhibits P130)	
	C130	CR4x00 Product Specification (CRDS 43929) (**CD-ROM** Chaparral Exhibits P267)	6/1/1998
	C131	Symbios Logic – Hardware Functional Specification for the Symbios Logic Series 3 Fibre Channel Disk Array Controller Model 3701 (Engelbrecht Ex 3 (LSI-1659-1733) (**CD-ROM** Pathlight Exhibits D074)	
	C132	Report of the Working Group on Storage I/O for Large Scale Computing; Department of Computer Science Duke University: CS-1996-21 (PTI 173330-347). (**CD-ROM** Pathlight Exhibits D098)	
	C133	Brian Allison's 1999 Third Quarter Sales Plan (PDX 38)CNS 022120-132)) (**CD-ROM** Pathlight Exhibits D201)	6/5/2001
	C134	Brooklyn SCSI-SCSI Intelligent External RAID Bridge Definition Phase External Documentation ((**CD-ROM** Pathlight Exhibits D129)	
	C135	StorageWorks HSx70 System Specification by Steve Sicola dated 6/11/96 4:57pm, Revision 4.	6/11/1996
	C136	ANSI TR X3.xxx-199x, Revision 9 of X3-991D. Draft Proposed X3 Technical Report - Small Computer System Interface - 3 Generic Packetized Protocol (SCSI-GPP). Computer and Business Equipment Manufacturers Assoc.	
	C137	Enterprise Systems Connection (ESON) Implementation Guide, July 1996, IBM International Technical Support Organization, Poughkeepsie Center	7/1/1996

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	12/690,592
Filing Date	01/20/2010
First Named Inventor	Geoffrey B. Hoese
Group Art Unit	2111
Examiner Name	Unknown
Atty Docket Number	CROSS1120-33

Sheet 8 of 9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C138	Digital Delivers Industry-Leading Enterprise-Class Storage Solutions. StorageWorks Family Provides Easiest Path to Fibre Channel. Three pages by Company News Oncall dated 09/09/04	9/9/2004
	C139	American National Standard for Information Technology – Fibre Channel Protocol for SCSI. ANSI X3.269-1996	6/18/1905
	C140	F1710A File Control Unit and F6493 Array Disk Subsystem by Hitoshi Matsushima, Shojiro Okada and Tetsuro Kudo.	2/3/1995
	C141	The Legend of AMDAHL by Jeffrey L. Rodengen (5 pages)	
	C142	Office Action dated February 6, 2007 from the Japanese Patent Office regarding related application No. 526873/2000.	2/6/2007
	C143	InfoServer 100 System Operation Guide, Order Number EK-DIS1K-UG-001	
	C144	iNFOsERVER 100 Installation and Owner's Guide, Order Number EK-DIS1K-IN-001	
	C145	Software Product Description: Product Name: InfoServer 100 Software, Version 1.1 SPD 38.59.00	11/1/1991
	C146	Software Product Description: Product Name: InfoServer Client for ULTRIX, Version 1.1, SPD 40.78.01	4/1/1993
	C147	Draft Proposed American National Standard. X3.269-199X, Revision 012. Information System - dpANS Fibre Channel Protocol fo SCSI.	12/4/1995
	C148	Impactdata Launches Breakthrough Architecture for Network Storage.	11/13/1996
	C149	Impactdata..News Release: Impactdata Introduces New Storage Architecture for High Performance Computing. 2 Pages.	11/12/1996
	C150	Impactdata..News Release: Impactdata's Network Peripheral Adapter (NPA) Pushes Technology Envelope of Data Storage Management in High-Speed Computing Environments. 2 Pages.	11/12/1996
	C151	Impactdata..News Release: Impactdata and Storage Concepts Announce Integration of FibreRAID II Storage Solution with Impactdata's Distributed Storage Node Architecture (DSNA). 2 pages.	11/18/1996
	C152	Impactdata..News Release: Breece Hill Libraries Now Able to Attach Directly to High Speed Networks Peripheral Adapter from Impactdata. 2 Pages.	11/20/1996
	C153	Impactdata - DSNA Questions and Answers. 22 Pages.	
	C154	Impactdata - Network Storage Solutions. 4 pages.	
	C155	Network Storage Building Blocks. 2 Pages.	
	C156	Impactdata - NPA (Network Peripheral Interface). 4 Pages	
	C157	Impactdata - CPI (Common Peripheral Interfae). 2 Pages	
	C158	Impactdata - SNC (Storage Node Controller). 2 Pages	
	C159	Impactdata - DSNA (Distributed Storage Node Architecture) Protocol. 2 Pages	
	C160	Impactdata - DS-50. 2 Pages	
	C161	Impactdata - Corporate Fact Sheet. 1 Page	

Examiner Signature		Date Considered	

INFORMATION DISCLOSURE STATEMENT

Application Number	12/690,592
Filing Date	01/20/2010
First Named Inventor	Geoffrey B. Hoese
Group Art Unit	2111
Examiner Name	Unknown
Atty Docket Number	CROSS1120-33

Sheet 9 of 9

Examiner Initials	Cite No.	OTHER PRIOR ART -- NON PATENT LITERATURE DOCUMENTS	Date
	C162	Raider-5 "Disk Array Manual for the UltraSCSI Controller". Part No. 261-0013-002. 191 Pages	
	C163	Impactdata - White Paper: Distributed Storage Node Architecture (DSNA). January, 1997	01/97
	C164	Impactdata - DSNA Distributed Storage Node Architecture "Reference Guide". 44 Pages	
	C165	F1710 Logic Specification	
	C166	Translation of Final Office Action issued in JP 526873/2000 mailed 05/14/08. 4 Pages.	5/14/2008
	C167	Office Action issued in USPA 11/851,837 dated 12/22/08, Hoese, 7 pages	12/22/2008
	C168	English Translation of Japanese Laid-Open Publication No. 5-181609. 9 pgs.	7/23/1993
	C169	English Translation of Japanese Laid-Open Publication No. 7-20994. 57 pgs	1/24/1995
	C170	F1710 File Control Unit (FCU) Logical Specifications. 11 Pages	12/9/1997
Examiner Signature		Date Considered	

(B)

(12) UK Patent Application (19) GB (11) 2 296 798 (13) A

(43) Date of A Publication 10.07.1996

(21) Application No 9500173.1

(22) Date of Filing 05.01.1995

(71) Applicant(s)
Spring Consultants Limited

(Incorporated in the United Kingdom)

Unit 5, Ashbrook Mews, Westbrook Street,
BLEWBURY, Oxon, OX11 9QA, United Kingdom

(72) Inventor(s)
Andrew Paul George Randall
Norman Hamilton Burkies

(74) Agent and/or Address for Service
Atkinson & Co
Sixth Floor,High Holborn House, 52-54 High Holborn,
LONDON, WC1V 6SE, United Kingdom

(51) INT CL^6
G06F 12/02

(52) UK CL (Edition O)
G4A AMX

(56) Documents Cited
None

(58) Field of Search
UK CL (Edition N) G4A AMX
INT CL^6 G06F 12/02
ONLINE DATABASES : WPI, INSPEC

(54) Storing data efficiently on a RAID

(57) Data is stored in such a way that a plurality of user terminals 16 are given access to a large storage volume in the form of a redundant array of inexpensive drives (RAID 5) 21 to 25. The large storage volume is divided into a plurality of storage blocks and each of said blocks has a capacity which is smaller than the size of an emulated logical disc drive. In operation, physical blocks of data are mapped onto an emulated drive as storage is required up to a predetermined capacity.

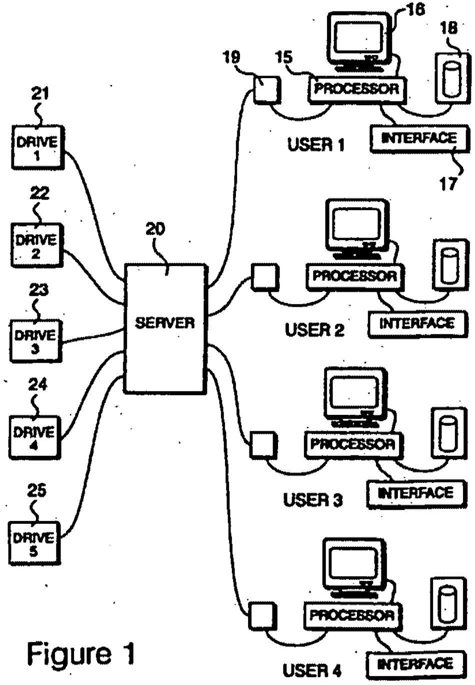


Figure 1

GB 2 296 798 A

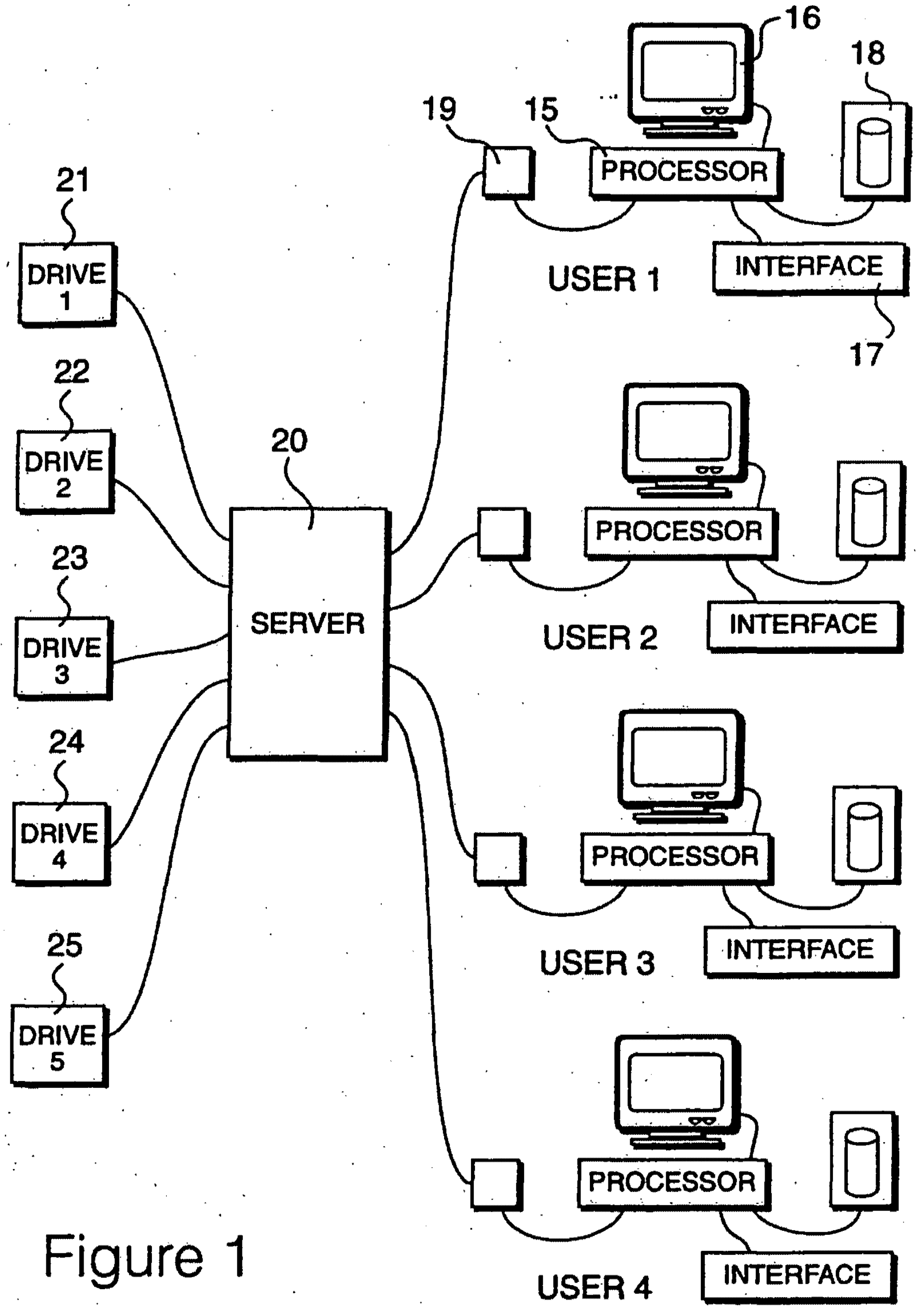

16
18
19
15
PROCESSOR
21
DRIVE 1
USER 1
INTERFACE
17
22
DRIVE 2
20
PROCESSOR
23
DRIVE 3
SERVER
USER 2
INTERFACE
24
DRIVE 4
PROCESSOR
USER 3
INTERFACE
25
DRIVE 5
PROCESSOR
USER 4
INTERFACE

Figure 1

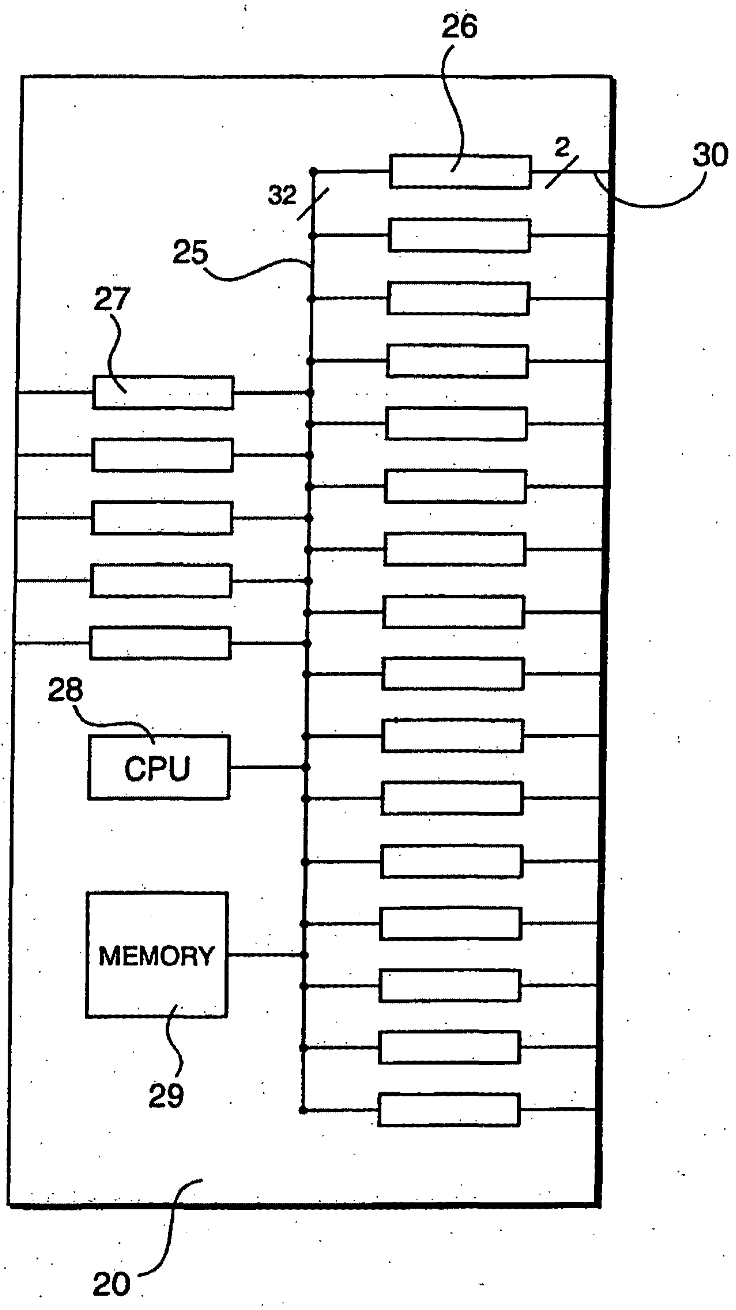


Figure 2

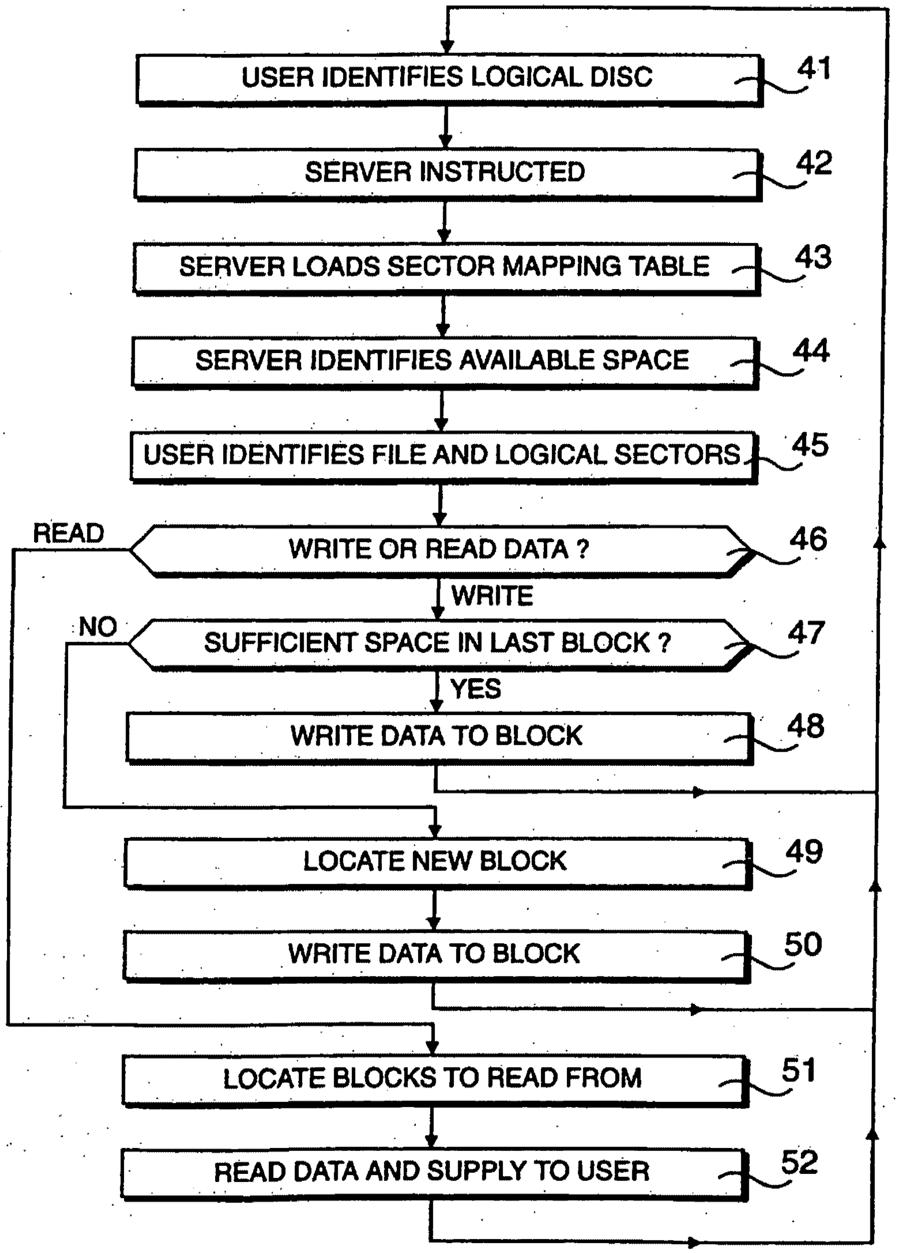

USER IDENTIFIES LOGICAL DISC
41
SERVER INSTRUCTED
42
SERVER LOADS SECTOR MAPPING TABLE
43
SERVER IDENTIFIES AVAILABLE SPACE
44
USER IDENTIFIES FILE AND LOGICAL SECTORS
45
READ
WRITE OR READ DATA ?
46
WRITE
NO
SUFFICIENT SPACE IN LAST BLOCK ?
47
YES
WRITE DATA TO BLOCK
48
LOCATE NEW BLOCK
49
WRITE DATA TO BLOCK
50
LOCATE BLOCKS TO READ FROM
51
READ DATA AND SUPPLY TO USER
52

Figure 3

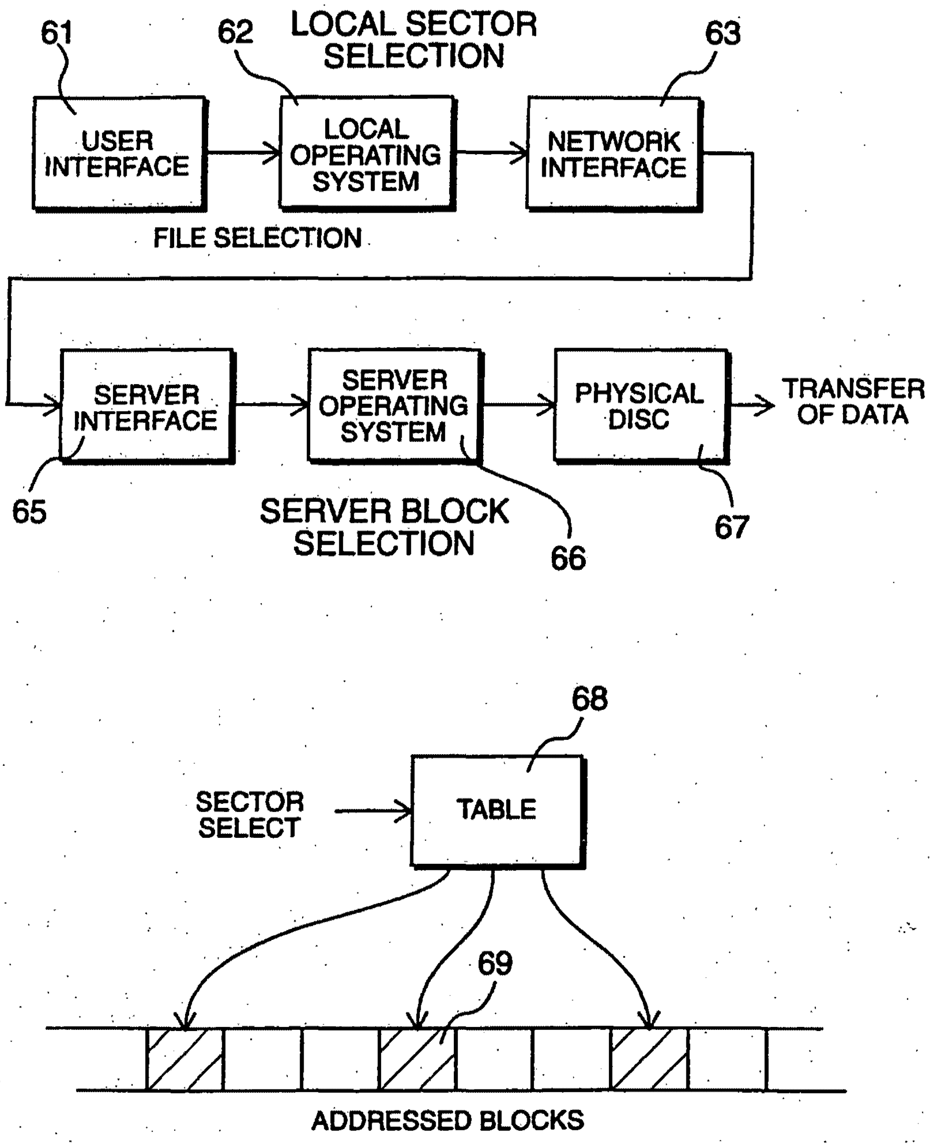

61
62
LOCAL SECTOR SELECTION
63
USER INTERFACE
LOCAL OPERATING SYSTEM
NETWORK INTERFACE
FILE SELECTION
SERVER INTERFACE
SERVER OPERATING SYSTEM
PHYSICAL DISC
TRANSFER OF DATA
65
SERVER BLOCK SELECTION
66
67
68
SECTOR SELECT
TABLE
69
ADDRESSED BLOCKS

Figure 4

2296798

STORING DATA

The present invention relates to storing data. In particular, the present invention relates to an environment in which a plurality of user terminals have shared access to a large storage volume.

Systems are known in which data storing devices, often referred to as volumes, are shared amongst a plurality of user terminals or workstations. Typically, the volume is associated with a local workstation, referred to as a server, and the totality of the workstations are interconnected by a network, such as an ethernet. Such an arrangement provides efficient shared access to files provided that the amount of data contained within each file is small compared to the transmission bandwidth provided by the network. In operation, given that many users may be sharing the network bandwidth, the bandwidth allocated to any one particular user will be significantly less than the theoretical maximum provided by the network. Thus, as files get larger, it is preferable for the workstations to be given direct access to a storage volume such that operational time is not lost while waiting for data to be transferred. For example, an A4 full colour image may consist of a total of 30 Mbytes of data. When transmitted over typical networks, a transfer duration of several minutes may take place before the totality of the data has been received.

A problem with providing direct access to discs is that only one workstation may be given access to the data and in order for the data to be loaded into another machine, it may be necessary to physically move transferrable discs, such as SCSI optical discs. Systems also exist under which a plurality of users may share direct access to a data storage device and, consequently, measures must be implemented to remove the risk of contention problems. Thus, a particular workstation must release access to a particular file or disc partition before any of the other workstations may be allowed to write to that file.

In known systems, system specific software must be loaded into each workstation, so that each workstation is provided with instructions relating to the contention protocols. In addition, a plurality of workstations are given access to the shared volume by effectively dividing the volume into a plurality of partitions. Thus, in this way, a first workstation may write and read data to a first partition of the disc, with a second workstation writing and reading to a second partition of the disc. At a later date, the first workstation may release the first partition, thereby allowing another workstation to be given access to this partition. In this way, a plurality of workstations may each access partitions within the volume without the data needing to be transferred, thereby significantly improving operational performance.

A problem with the above arrangement is that the partitioning of the disc may result in substantial storage regions being taken up that are only available for one workstation at any one time but do not actually contain valid data. Thus, for example, ten partitions of a very large disc volume may each contain a relatively small amount of data. However, although a substantial amount of empty space remains on the disc, as far as the system is concerned, it would not be possible for this space to be allocated to another workstation, given that, as far as the system is concerned, the storage volume is fully allocated.

According to a first aspect of the present invention, there is provided a method of storing data wherein a plurality of user terminals access a large storage volume, comprising steps of emulating the presence of a logical disc drive having a predetermined capacity; dividing said storage volume into a plurality of storage regions, wherein each of said regions is smaller than the size of an emulated logical disc drive; and mapping physical regions of data to an emulated drive dynamically as additional storage is required, up to said predetermined capacity.

Thus, in accordance with said first aspect, a workstation may be given access to a logical disc drive which it perceives as having a predetermined capacity. For example, the predetermined capacity may be similar to that provided by an optical disc providing 600 Mbytes of storage. However, physical storage locations on the large storage volume are only allocated, region by region, as the workstation demands additional storage through the writing of larger files to the disc.

In a preferred embodiment, a look-up table is associated with each accessible logical drive and a particular look-up table is loaded when its associated logical drive is selected.

According to a second aspect of the present invention, there is provided apparatus for storing data, having a plurality of user terminals and means for each of said terminals to be given access to said stored data, comprising means for emulating the presence of a logical disc drive having a predetermined capacity; means for dividing a storage volume into a plurality of storage regions, wherein each of said regions is smaller than the size of an emulated logical disc drive; and mapping means for mapping said physical regions of data to an emulated drive dynamically as additional storage is required, up to said predetermined capacity.

The system will now be described by way of example only, with reference to the accompanying Figures, in which:

Figure 1 shows an environment in which a plurality of workstations have access to a shared storage volume including a shared file server;

Figure 2 details the shared file server identified in Figure 1;

Figure 3 illustrates an application of the system shown in Figure 1; and

Figure 4 shows a schematic representation of the system, including the dynamic allocation of storage regions.

An environment in which a plurality of users have access to a shared storage volume is illustrated in Figure 1. In the environment shown in Figure 1, each workstation is provided with a processor 15, a visual display unit 16, an interface device in the form of a keyboard and/or a mouse or trackerball etc. 17 and a local disc drive storage device 18.

Each processor 15 is connected to a server interface 19 which allows said processors 15 to communicate with a shared file server 20. The file server 20 is connected to typically five physical hard disc drives 21, 22, 23, 24 and 25. This disc drive combination provides typically thirty-six Gbytes of storage with an access speed of typically 10 Mbytes per second.

Disc drives 21 to 25 may be configured as a redundant array, commonly referred to as a redundant array of inexpensive discs (RAID). In the preferred implementation, five discs are provided and the coding used to write data to the disc is commonly referred to as RAID 5. Thus, under this protocol, redundant data is written to the discs such that if one of the drives becomes inoperable or suffers irretrievable damage, all of the data can be reconstituted from the remaining four drives.

Data is written to the drives in the form of identifiable blocks or regions of a predetermined length. The size of these blocks is determined from a trade-off between disc space optimisation and disc fragmentation. However, the system is primarily designed for storing large graphics files, therefore blocks may be quite large and it is proposed that said blocks should have a size between two Mbytes and thirty-two Mbytes. Similarly, it is possible that the block size could be configurable for a particular application.

In operation, a user issues commands under software control which effectively result in a logical drive being made available by the server 20. Communication between the user and the server 20 is effected via the interface 19 and as far as the user is concerned, interface 19 presents a standard small computer serial interface (SCSI) to the processor 15. Once a logical disc has been established, the user may access this drive.

The user's workstation receives data to the effect that it has been given access to a disc of a predetermined size, say 600 Mbytes for example, but in actuality, physical space is only allocated dynamically in regions as storage space for the storage of actual data is required.

Thus, in the system shown in Figure 1 the server does not immediately allocate 600 Mbytes of storage to a user when access to a 600 Mbyte logical drive is requested. Space on drives 21 through 25 is not divided into 600 Mbytes (or similar) partitions. Drives 21 through 25 are divided into blocks of between two and thirty-two Mbytes and blocks are only written to as data becomes available.

For the benefit of this illustration, it will be assumed that storage space on drives 21 through 25 has been divided into blocks of two Mbytes, thereby making two Mbyte blocks available for data storage purposes. As data is written to the drives, via an interface 19, said data will occupy one of said two Mbyte blocks. As the volume of data increases beyond two Mbytes, the server 20 will identify a new block of two Mbytes and data originating from a user will then continue to be written to this new two Mbyte block. Thus, for example, if a user has written a total of five Mbytes, the server is required to maintain a list of where these five Mbytes actually reside on the drives, in terms of three two-Mbyte blocks. However, as far as the user is concerned, five Mbytes of data have been written to on a logical drive having 600 Mbytes of available capacity.

Data is conventionally written to disc drives in terms of identifiable blocks. As far as the user is concerned, data is written to as blocks on a 600 Mbyte logical drive, which are in turn mapped onto real blocks on the RAID. However, the logical blocks may be written to in a substantially similar way to that in which real drives would be re-written to. Thus, it is not necessary for data to be written to the logical drives in what appears to be a contiguous region of disc space. Although the actual storage allocated for a logical drive is distributed over the RAID, the logical drives may appear, from the user's point of view, to be fragmented themselves. Thus, logical blocks of data may appear displaced over a logical drive, effectively emulating the presence of fragmentation on the logical disc. The system emulates such a situation by providing mapping firstly of blocks to logical drive locations and then mapping from logical drive locations to block locations on the RAID.

Many users may be given access to many virtual drives, allowing data to be accessed via many workstations without actually being transferred over a network. However, when capacity is allocated it is not wasted, in that blocks of two Mbytes are only allocated as actual storage is required.

In a preferred embodiment, it is envisaged that a server 20 would allow up to sixteen users to be connected thereto, although provision is made for server boxes to be connected in tandem, thereby providing access to a further 16 users for each box so connected.

The server 20 is detailed in Figure 2. Internally, a 32 bit parallel bus 25 provides communication between user interface circuits 26, disc drive interfaces 27, an internal processing unit 28 and internal program and data memory 29.

The server 20 is connected to each user interface 19 via a respective interface circuit 26 via two coaxial cables 30, providing a bi-directional link capable of conveying 100 Mbytes per second. Similarly, disc interface circuits 27 provide a parallel access to disc drives 21 through 25 and using connections of this type, it is necessary for disc drives 21 through 25 to be in close proximity to server box 20. In practice, the combination of server 20 along with disc drives 21 through 25 could be housed in a common housing with a shared power supply. However, coaxial cables 30 allow the users to be positioned at a significant distance from the server 20 and the interfaces are such that they will allow runs in excess of 100 metres. Thus, these serial connections are similar or may take advantage of high speed ethernet links.

In an alternative embodiment, user processors 15 are connected to the server 20 via conventional SCSI interfaces which, although reducing the overall complexity of the system, also reduce the maximum distance between the server 20 and the processors 15.

An application of the system is illustrated in Figure 3. At step 41 a user identifies a logical disc, either by running server related software or, alternatively, in response to manual operations of a device connected to interface 19. Thus, if it is not possible to embed server software within a user's terminal, it is possible to provide interfaces 19 with additional control devices such that, in response to manual operation of switches etc., commands are sent to server 20 so as to establish a logical disc connection.

Communication of this type, allowing a user to send commands to the server 20, is achieved using vendor unique command blocks, which are data areas provided for specific proprietary applications within the SCSI standard. Thus, in response to user originating commands, the server is instructed at step 42 to the effect that a user requires access to a logical drive.

For each logical drive which may be made available to the users, it being noted that once a logical drive has been established by any particular user, other users may be given access to it, it is necessary for the server 20 to create a sector mapping table for that particular logical drive. Thus, in response to commands generated by a user's processor, establishing logical sectors of a SCSI disc, it is necessary for the server 20 to map these logical sectors onto physical blocks or groups of physical blocks stored within the physical drives 21 through 25. At the CPU 28, reference is made to a look-up table stored within memory 29 which, as previously stated, identifies physical data blocks held by the redundant disc array. Thus, the CPU is required to generate the sector instructions relevant for the physical drives 21 through 25, which are issued to respective ones of said drives via respective interface circuits 27.

Once a user has requested use of a logical drive, the server identifies the space available to the user at step 44, in response to which the user may identify particular files to be written to or read from the logical drive.

At step 46 it is determined whether the user wishes to write data to or read data from a logical drive. If data is being written to the drive, an enquiry is made at step 47 as to whether space is available on the last block to be written to. If space is available, data is written to the next identified block at step 48. Alternatively, if sufficient space is not available on the last block, a new block is selected at step 49 and data is written to this block at step 50.

If a read operation is identified at step 46, the physical blocks to be read are identified at step 51, the data is read at step 52 and supplied to the requesting user in a suitable form. Thereafter, the process may be repeated and further identifications may be made at step 41.

A schematic representation of the system is illustrated in Figure 4. At a workstation, a user is presented with a user interface, capable of providing an environment for allowing existing logical drives to be selected and providing the capacity for new drives to be defined.

The user interface 61 is in turn supported by a local operating system 62. Thus, an operator makes a file selection via user interface 61 and it is then necessary for the local operating system 62 to generate commands which may be interpreted by the physical storage system.

As far as the local operating system 62 is concerned, the system is making access to conventional SCSI disc drives. Thus, the local operating system 62 communicates with a network interface, illustrated as 63 and physically consisting of interface 19 shown in Figure 1. The network interface 63 receives standard SCSI commands from the local operating system 62 and in turn generates modulated data for transmission over the serial link, shown as 64, connecting the network interface 63 to a server interface 64. A physical representation of server interface 64 is identified in Figure 2 as 26.

The transmission of data between the local operating system 62 and the network interface 63 conforms to establish SCSI protocols. However, the communication between network interface 63 and server interface 64 is internally defined by the system and is designed, in a preferred embodiment, to provide maximum data transfer rates over substantial lengths of cable, such as coaxial cable. Furthermore, the connection between the network interface 63 and the server interface 65 is bi-directional.

The network interface 63 is primarily concerned with driving signals generated by the local operating system 62 so that they may be transmitted over the serial communication link 64. However, the sector indications generated by the local operating system 62 are conveyed to the server interface 65 and it is the server operating system 66 which is required to convert SCSI sector selections into addresses for physical blocks located on the array of physical drives.

Thus, the server operating system 66 supplies addressing signals to the physical discs, identified as 67 whereafter data transfer is effected.

The server operating system 66 converts SCSI sector definitions into addressable physical data blocks by means of a look-up table, identified as 68. A look-up table is defined for each logical drive and when a logical drive is selected by an operator its associated look-up table is loaded to an operating area of memory 29 within the server 20. Thus, within the operating system 66, a logical drive is identified, resulting in a table 68 being loaded. Thereafter, SCSI sector selections are supplied as inputs to said table, which then results in addresses for physical data blocks being generated as outputs. Thus, as illustrated in Figure 4, the table 68 effectively points to addressable data blocks 69 in the array of physical data storing discs 21 through 25.

CLAIMS

1. A method of storing data wherein a plurality of user terminals access a large storage volume, comprising steps of

emulating the presence of a logical disc drive having a predetermined capacity;

dividing said storage volume into a plurality of storage regions, wherein each of said regions is smaller than the size of an emulated logical disc drive; and

mapping said physical regions of data to an emulated drive dynamically as additional storage is required, up to said predetermined capacity.

2. A method according to claim 1, wherein a plurality of logical drives are accessible to a user.

3. A method according to claim 2, wherein a look-up table is associated with each accessible logical drive and a particular look-up table is loaded when its associated logical drive is selected.

4. A method according to any of claims 1 to 3, wherein the logical drives appear to a user system in a form compatible with a local physical disc drive.

5. A method according to claim 4, wherein said logical drive is connected via a small computer serial interface (SCSI).

6. A method according to any of claims 1 to 5, wherein the size of said regions is variable and pre-set for a particular application.

7. Apparatus for storing data, having a plurality of user terminals and means for each of said terminals to be given access to said stored data, comprising

means for emulating the presence of a logical disc drive having a predetermined capacity;

means for dividing a storage volume into a plurality of storage regions, wherein each of said regions is smaller than the size of an emulated logical disc drive; and

mapping means for mapping said physical regions of data to an emulated drive dynamically as additional storage is required, up to said predetermined capacity.

8. Apparatus according to claim 7, including means for defining a plurality of logical drives, each accessible to a user.

9. Apparatus according to claim 8, including means for defining a look-up table associated with each of said logical drives and means for loading a particular look-up table when its associated logical drive is selected.

10. Apparatus according to any of claims 7 to 9, including means for presenting a logical drive to a system user in a form compatible with a local physical disc drive.

11. Apparatus according to claim 10, wherein said logical disc drive is connectable via a small computer serial interface (SCSI).

12. Apparatus according to any of claims 7 to 11, including means for pre-setting the size of said regions for a particular application.

13. Apparatus according to any of claims 7 to 11, wherein the size of said regions is variable in response to operator requests and said means for emulating the presence of the logical drive is arranged to supply data to a user terminal identifying the size of a logical drive being emulated.

14. A method of storing data substantially as herein described with reference to the accompanying Figures.

15. Apparatus for storing data substantially as herein described with reference to the accompanying Figures.

The Patent Office

15

Application No:	GB 9500173.1	**Examiner:**	Mr S J Probert
Claims searched:	1-15	**Date of search:**	6 April 1995

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.N): G4A AMX

Int Cl (Ed.6): G06F 12/02

Other: Online Databases : WPI, INSPEC

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
	None	

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

52

(12) UK Patent Application (19) GB (11) 2 297 636 (13) A

(43) Date of A Publication 07.08.1996

(21) Application No 9502377.6

(22) Date of Filing 02.02.1995

(71) Applicant(s)
Spring Consultants Limited

(Incorporated in the United Kingdom)

Unit 5, Ashbrook Mews, Westbrook Street,
BLEWBURY, Oxon, OX11 9QA, United Kingdom

(72) Inventor(s)
Norman Hamilton Burkies
Andrew Paul George Randall

(74) Agent and/or Address for Service
Atkinson & Co
Sixth Floor,High Holborn House, 52-54 High Holborn,
LONDON, WC1V 6SE, United Kingdom

(51) INT CL^6
G06F 3/06

(52) UK CL (Edition O)
G4A AFS AMX

(56) Documents Cited
EP 0078683 A2
Dialog record 01425541 of UNIX Review, vol. 9, No.4,
April 1991, page 98

(58) Field of Search
UK CL (Edition N) G4A AFS AMX
INT CL^6 G06F 3/06
ONLINE : WPI, INSPEC, COMPUTER DATABASE

(54) Storing data on emulated, logical, removable, disc drives

(57) Data is stored on a large storage volume implemented as a redundant array of five inexpensive discs (21-25). This volume is controlled so as to emulate the presence of a plurality of logical drives. Workstations (15,16) accessing the drives perceive them as removable SCSI drives. Consequently, when a remote workstation closes access to a previously accessed logical drive, a disc dismount command is generated, as required by a removable disc drive, thereby enabling other workstations to obtain access to that drive.

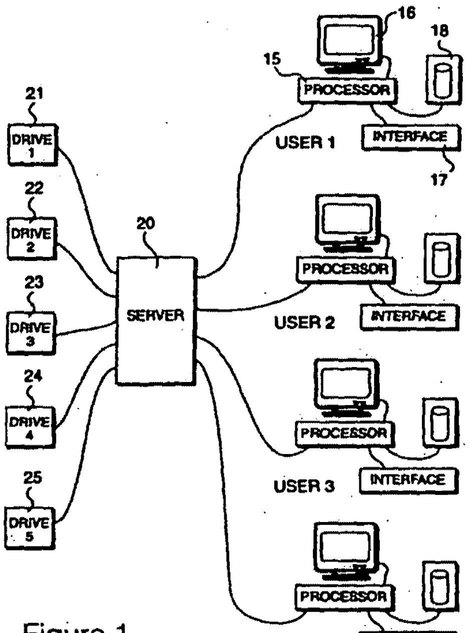


Figure 1

GB 2 297 636 A

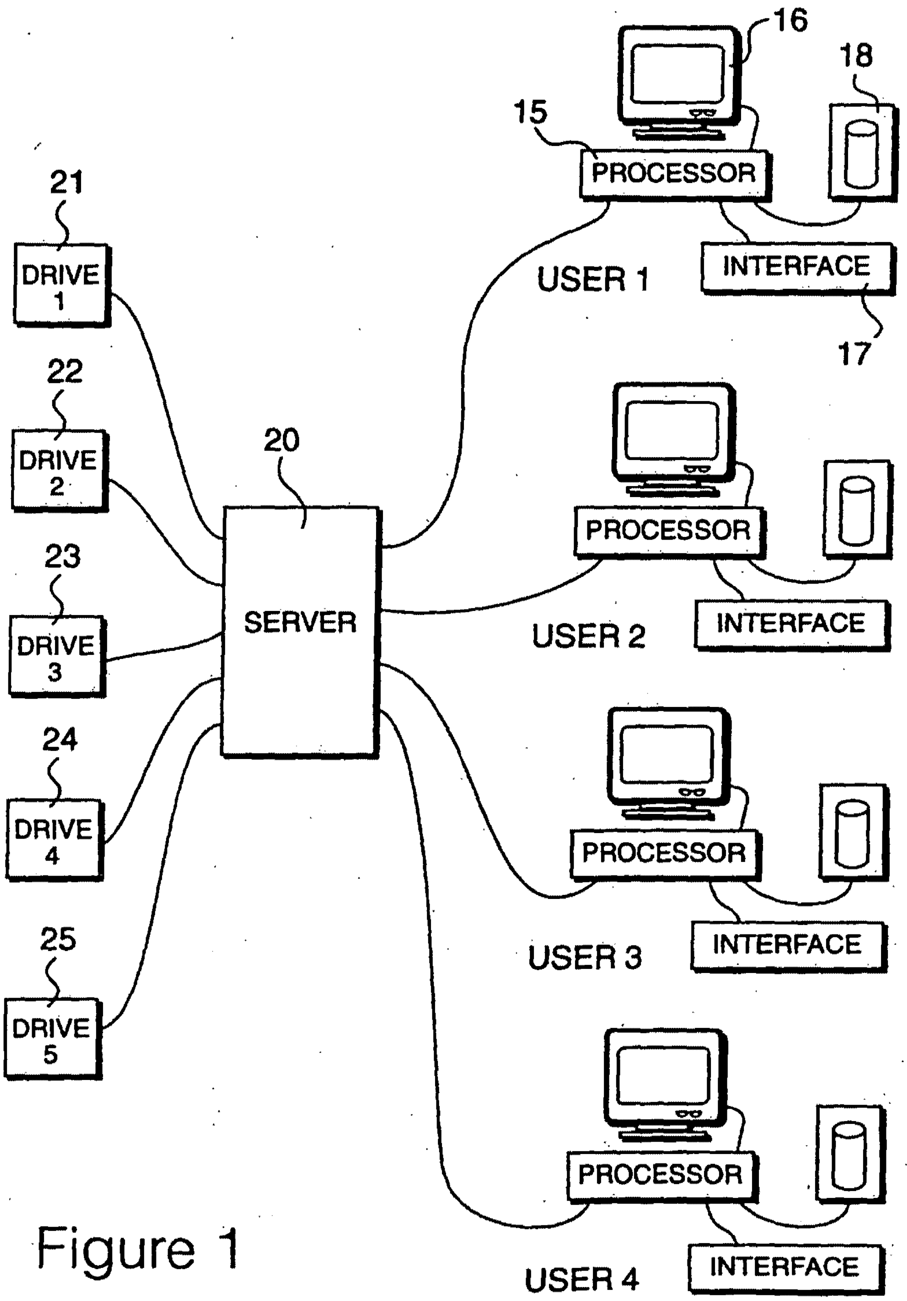

16
18
15
PROCESSOR
INTERFACE
USER 1
17
21
DRIVE 1
22
DRIVE 2
20
SERVER
23
DRIVE 3
24
DRIVE 4
25
DRIVE 5
PROCESSOR
INTERFACE
USER 2
PROCESSOR
INTERFACE
USER 3
PROCESSOR
INTERFACE
USER 4

Figure 1

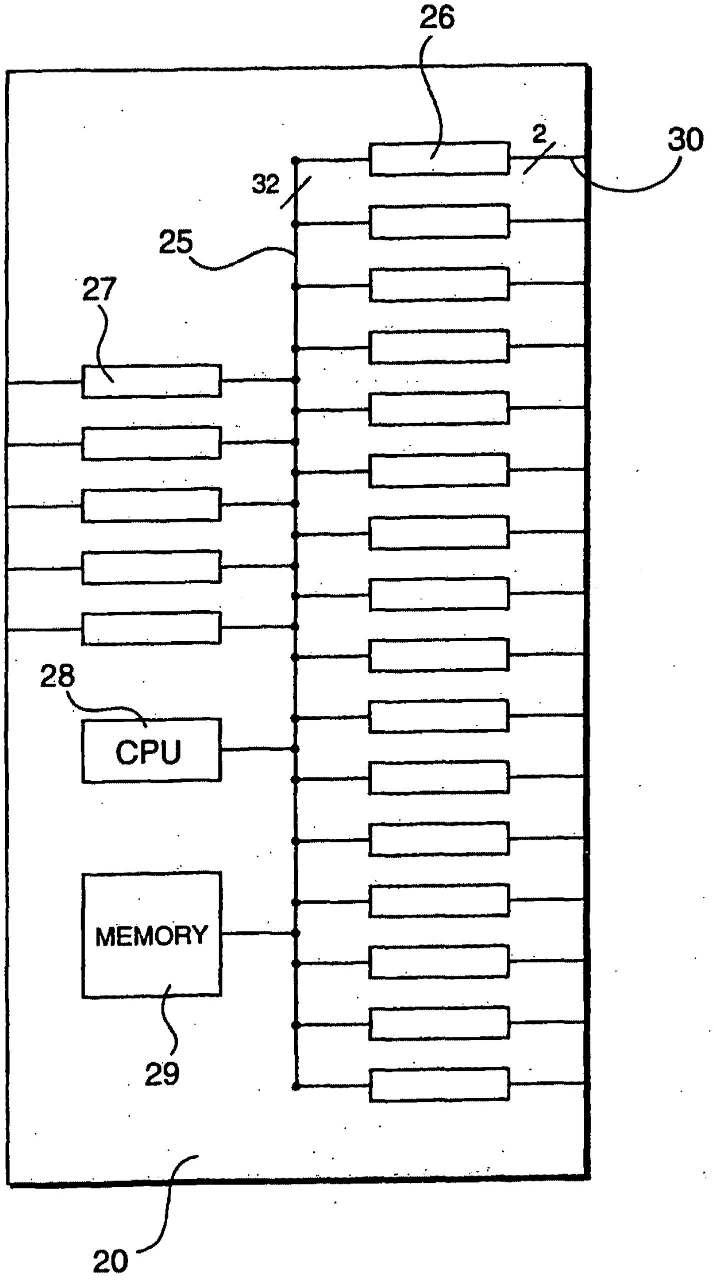


Figure 2

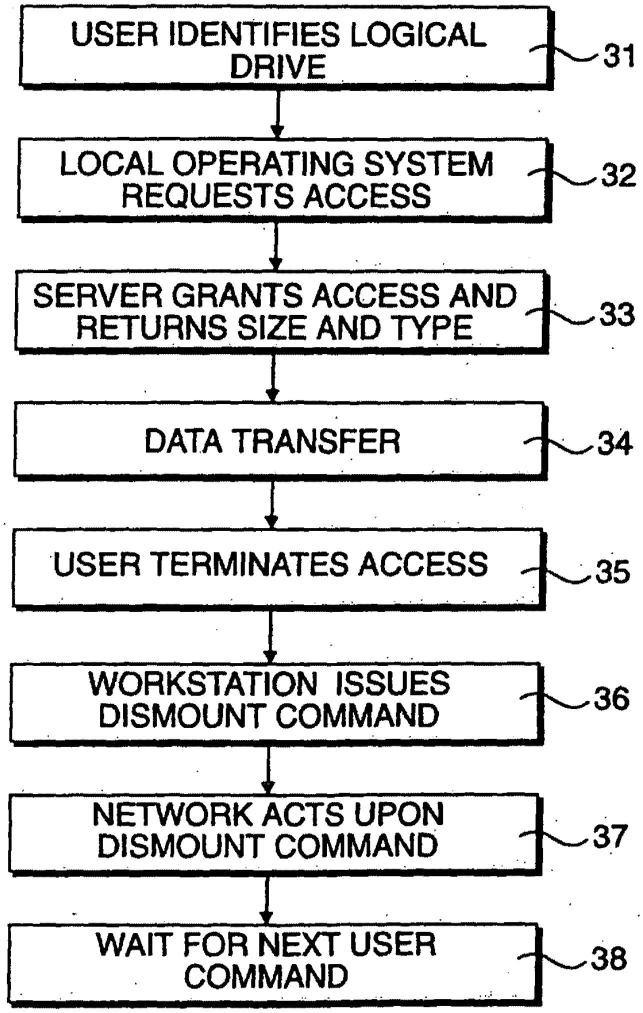

USER IDENTIFIES LOGICAL DRIVE
31
LOCAL OPERATING SYSTEM REQUESTS ACCESS
32
SERVER GRANTS ACCESS AND RETURNS SIZE AND TYPE
33
DATA TRANSFER
34
USER TERMINATES ACCESS
35
WORKSTATION ISSUES DISMOUNT COMMAND
36
NETWORK ACTS UPON DISMOUNT COMMAND
37
WAIT FOR NEXT USER COMMAND
38

Figure 3

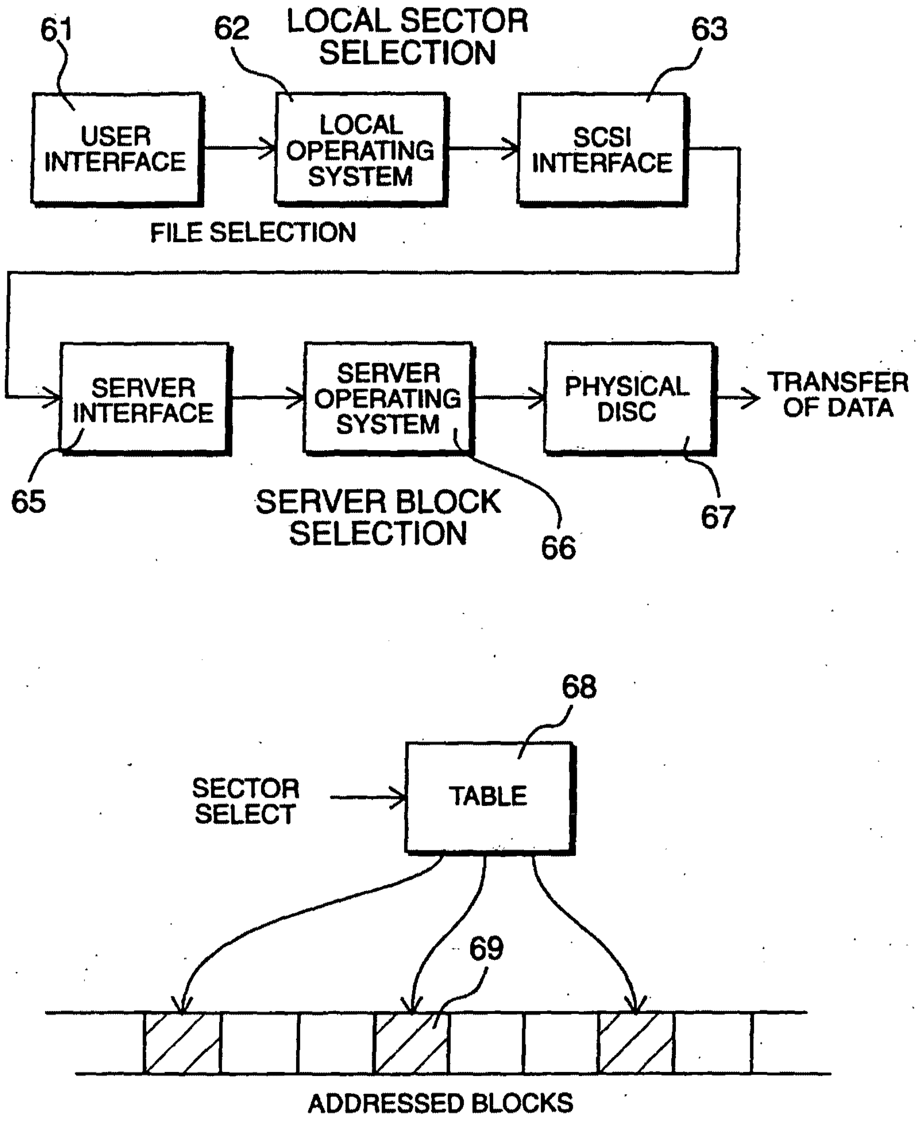

61
62
LOCAL SECTOR SELECTION
63
USER INTERFACE
LOCAL OPERATING SYSTEM
SCSI INTERFACE
FILE SELECTION
SERVER INTERFACE
SERVER OPERATING SYSTEM
PHYSICAL DISC
TRANSFER OF DATA
65
SERVER BLOCK SELECTION
66
67
68
SECTOR SELECT
TABLE
69
ADDRESSED BLOCKS

Figure 4

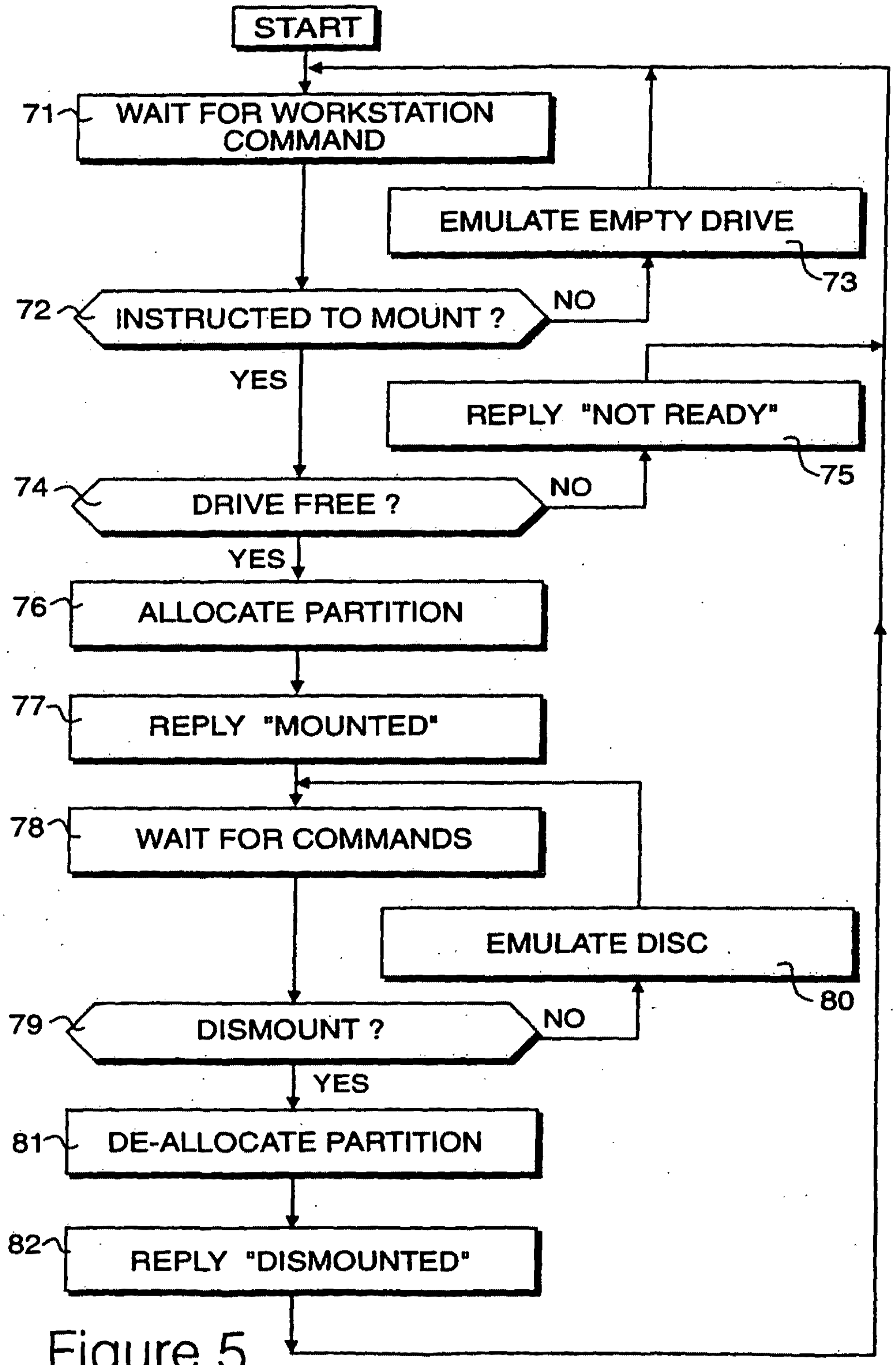

START
71
WAIT FOR WORKSTATION COMMAND
EMULATE EMPTY DRIVE
73
72
INSTRUCTED TO MOUNT ?
NO
YES
REPLY "NOT READY"
75
74
DRIVE FREE ?
NO
YES
76
ALLOCATE PARTITION
77
REPLY "MOUNTED"
78
WAIT FOR COMMANDS
EMULATE DISC
80
79
DISMOUNT ?
NO
YES
81
DE-ALLOCATE PARTITION
82
REPLY "DISMOUNTED"

Figure 5

2297636

STORING DATA

The present invention relates to storing data. In particular, the present invention relates to large storage volumes controlled so as to emulate the presence of a plurality of logical drives.

Systems are known in which a large storage volume emulates the presence of a plurality of smaller volumes, which in turn may assist a user by facilitating logical arrangement of data, such that data of a first type may be kept separate from data of a second type. As far as an operating system is concerned, it has access to a plurality of drives as an alternative to having access to only one drive. Most operating systems are capable of controlling a plurality of logical drives in this way; within limits.

In more sophisticated environments, it is possible for a plurality of users to be given access to a shared volume divided into a plurality of logical drives. The division of the volume into a plurality of logical drives facilitates the interchange of information between users. Thus, a first user may log onto a logical drive, manipulate data contained within that drive and then log off, so as to allow another user to be given access to the logical drive. Such a procedure is particularly attractive when large data files are being handled, such as data files representing full colour graphic images, where the transfer of data, even over relatively fast networks, may take a considerable amount of time.

In addition, a large shared volume may be constructed first to provide relatively fast access times, along with levels of redundancy, such that a single destructive event would not result in the whole data being lost, with recovery procedures being included as part of the overall structure.

Increasingly, computer workstations are being provided with localised processing capabilities having recognised and well supported operating systems. Examples are Apple Macintosh computers, IBM personal computers and Unix workstations etc. All of these systems have recognised protocols for the transfer of data. Thus, given the abundance of well supported operating systems, it is preferable to take full advantage of these operating systems so as to minimise the degree of bespoke software which needs to be generated and subsequently supported. System designs are restricted if full adherence to existing standards must be maintained, however, in some environments, an established system of operation may already be functional and the extent to which this system may be modified by the addition of new software etc., may be severely restricted. In some situations, the installation of a new suite of networking software may invalidate software agreements relating to primary localised processing.

In an environment in which a large storage volume emulates a plurality of discs, contention problems occur and the control processor must ensure that strict housekeeping routines are maintained, such that, for example, a previously accessed logical drive is properly deactivated when a particular user has finished with it, so that said drive may be accessed by other users and the overall integrity of this system is maintained. However, the degree to which network software requires to be embedded within workstation software should be minimised and it is undesirable for the network to place additional constraints on the workstations so as to assist the network's processing devices with their housekeeping tasks.

According to a first aspect of the present invention, there is provided a method of storing data, wherein a large storage volume emulates a plurality of logical drives; said logical drives emulate removable disc drives; and the closing of access to a previously accessed logical drive generates a disc dismount command.

Thus, an advantage of the present invention is that the logical drives emulated by the large storage volume are presented to users in the form of removable disc drives, although in preferred practical realisations, they would actually be embodied within an environment of large fixed drives, so as to optimise data capacity and disc access speed. However, operating systems for the individual workstations are fully conversant with the requirements of removable disc drives and, as required by the present invention, they will issue commands to said drives, informing the drive that access is no longer required.

In this way, it is possible to ensure that all necessary housekeeping procedures are effected when control over a logical disc drive is relinquished, either as part of normal operations or due to a software or hardware fault. Thus, for example, it is possible to ensure that directory information, cached in memory, is written back to disc, thereby updating the disc's directory, before releasing access to the logical drive. Thus, by emulating removable drives of this type, workstation software will automatically provide the necessary levels of housekeeping in order to ensure that access to a logical drive is released when no longer required by a particular operator.

The local workstation will interface with a logical drive over standard interfaces, provided for accessing removable disc drives. The workstation software will generate a disc dismount command and as far as the said software is concerned, a dismount of the removable disc will be effected, thereby releasing the tie between the local workstation and that particular logical disc drive. However, within the network, this command will be interpreted to the effect that the processor no longer requires access to the logical drive, thereby allowing housekeeping procedures to be performed by the network processor.

Preferably, the logical drives emulate removable SCSI drives which may be capable of storing between 200 MBytes and 900 MBytes of data. According to a second aspect of the present invention there is provided apparatus, including a large storage volume; a control device arranged to control data transfer with said storage volume and to provide user terminal access to said storage volume by emulating the presence of a plurality of removable disc drives wherein user terminals generate a disc dismount command when closing access to a previously accessed logical drive; and the control device responds to said disc dismount command by terminating connection to said previously connected logical drive.

In a preferred embodiment, the control device is arranged to read directory information from an access logical drive and said directory information stored on the disc is updated in response to a disc dismount command.

The invention will now be described by way of example only, with reference to the accompanying figures, in which:

Figure 1 shows a system in which a plurality of workstations have access to a shared storage volume, including a file server;

Figure 2 details the file server shown in Figure 1;

Figure 3 details operations performed by the system shown in Figure 1; and

Figure 4 represents the logical operations effected by the system shown in Figure 1, including removable disc emulation;

Figure 5 details the removable disc emulation procedures performed by the file server shown in Figure 1.

A system is shown in Figure 1 in which a plurality of users have access to a shared storage volume. At each user workstation, the user is provided with a processor 15, a visual display unit 16, a keyboard, mouse or similar interface device 17 and a local disc drive 18.

Each processor 15 includes conventional software so as to implement an operating system, allowing data transfer between the processor 15 and the disc drive 18. In addition, the operating system also facilitates data transfer between the processors 15 and a shared file server 20. In this preferred embodiment, the file server 20 is connected to five physical hard disc drives 21, 22, 23, 24 and 25, which in combination provide a total of thirty-six GBytes of storage with an access speed of typically 10 MBytes per second.

Disc drives 21 to 25 are configured as a redundant array, in which actual data is stored on four of the drives, with parity data stored on the fifth. In this way, any one of the physical drives 21 to 25 may be removed from the system, possibly due to operational failure (head crash etc.) whereafter said data may be re-constituted from the data available from the other four.

Thus, data integrity and reliability are assured without the need for implementing regular back-up procedures. The use of a plurality of disc drives in this way is known in the art as a redundant array of inexpensive discs. In the preferred embodiment this is implemented in accordance with the RAID 5 recommendation.

Data is written to the drives in the form of identifiable blocks or regions of a predetermined length. The size of these blocks is determined from a trade-off between disc space optimisation and disc fragmentation. The system is primarily designed for storing large full colour graphics files and blocks have a size of, typically, between two MBytes and thirty-two MBytes, although block size may be configurable so as to suit particular applications. In operation, users issue commands under software control which result in logical drives being made available by the server 20. Communication between users and the server 20 is implemented using established protocols. In the preferred embodiment, the standard small computer systems interface (SCSI) is implemented and suitable interface cards are mounted in association with processor 15 and server 20. Thus, once a logical drive has been established by the server 20, this drive may be accessed by the user who perceives the drive as a conventional SCSI drive, accessed via conventional protocols within the local operating system.

The server 20 is arranged to provide access to a total of sixteen user workstations and a further sixteen workstations may be given access by connecting a similar server in tandem with the first. The server is detailed in Figure 2 and, internally, a thirty-two bit parallel bus 25 provides communication between the user interface circuits 26 and disc drive interfaces 27. The server is controlled in response to commands issued by the central processing unit 28 which in turn receives programmed instructions from an internal memory device 29.

As previously stated, the server 20 is connected to each processor of a user workstation via a SCSI interface. The range of such interfaces is limited and in alternative embodiments it may be necessary to provide alternative connections, possibly via coaxial cables, so as to increase the distance between the server and the workstations. It is therefore envisaged that systems will be designed specifically for particular applications, so as to optimise connections between workstations and the server. Thus, in some environments, a large number of workstations may be provided relatively close to the server 20, in which case conventional SCSI interfaces may be employed whereas, in alternative arrangements, workstations may be distributed quite widely throughout a building, requiring more robust connections between the processors and the server 20. It is envisaged that connections of this type should allow the workstations to be displaced from the server by distances in excess of 100 metres, having characteristics similar to high speed ethernet links.

Typical operation of the system shown in Figure 1 is detailed in Figure 3. As far as the operating system executable by each user workstation is concerned, the workstation effectively has access to a large number of removable disc drives, although these are actually emulated by the server 20. In some situations, standard operating system software interfaces may be implemented within the user workstations so as to allow users to gain access to these logical drives. However, as the number of logical drives increases, it may be necessary to improve the environment provided for users, so that they are aware of the presence of the disc drives and are provided with an interface which facilitates access to them. However, these user interfaces would be overlaid over the operating system so that computer generated commands would result in instructions being generated at the operating system level.

Referring to Figure 3, a user identifies a logical disc drive to which access is required and identifies this logical disc drive at step 31. In response to the local request made at step 31, the local operating system implements measures to effect a request to access the logical disc drive, using conventional protocols. In particular, the processor 15 issues commands over the SCSI interface connected to the server 20.

In response to the request made at step 32, the server 20 will determine whether the logical disc drive is available and if the drive is available, it will grant access to the requesting workstation. As part of the SCSI protocol, the server will return data back to the requesting workstation, identifying the size of the logical drive and the drive type. Data relating to the drive type is very relevant to the present invention. In particular, data is returned back to the requesting workstation identifying the drive type as a removable drive having, in the preferred embodiment, a total of 600 MBytes of available capacity.

Thus, it should be appreciated, that the emulated drives differ significantly from the actual physical drives in two respects. Firstly, the emulated drives are significantly smaller than the actual physical drives on which they are being emulated, primarily to ensure that a large number of such drives may be supported by the system. Secondly, the physical drives are actually fixed drives and remain permanently in place. Thus, when the server writes data to a particular physical location, the server is assured that this physical location will remain in place and will not be exchanged for some other data storage medium. However, in the emulated environment, the requesting processors are informed that the drives to which they are writing should be treated removable drives, effectively warning the processor that these drives may be replaced and that a subsequent data transfer operation to that particular drive would not necessarily result in the same information being available on the storage medium.

In the system itself, the emulated drives are not physically replaced by other recording media and it is not actually necessary for a physical dismounting operation to be performed when data access has been completed. However, by informing the remote processors that they are dealing with removable disc drives, the resulting dismount or unload command issued by the operating systems of the remote processors will ensure that the server 20 has been instructed to the effect that the remote processors have completed their data transfer operations, thereby ensuring that the processor 20 receives sufficient information for it to complete its housekeeping tasks, thereby allowing other workstations to be given access to emulated drives once they have been released from a data transfer operation.

Thus, to summarise, when the server 20 grants access to an emulated logical disc drive, it informs the requesting processor that it has been given access to a removable disc drive having a total capacity of 600 MBytes.

Conventionally, data is written to disc drives as identifiable blocks. In order to optimise available storage space, these blocks would normally reside on physical drives as contiguous regions of storage, effectively reducing fragmentation. However, it is not essential for the data to be perceived as residing in contiguous regions. In the present embodiment, the workstation processors may write data to the logical disc drives as they feel fit. Thus a logical disc drive may be perceived as being fragmented.

Thus, at step 34 data transfer takes place and the workstation's local operating software may read and write to the logical drives as if they were local removable disc drives. However, given the nature of the RAID 5 drives 21 to 25, the rate of data transfer is substantially higher and only restricted by the capabilities of the interface circuits employed. Thus, as far as the workstation processor is concerned, along with its operating software, it is interfacing with a standard removable disc drive. However, as far as the actual operator is concerned, the rate of data transfer is significantly higher and, due to the parallel nature of the array, said transfer rate significantly exceeds that available from fast local hard drives. Thus, the operator is provided with the advantage of fast data access while at the same time allowing data to be shared between a plurality of users as if the data were contained on removable exchangeable drives. Furthermore, the physical removing and exchange of drives is not necessary and only occurs at a logical level.

After data transfer has been completed, a user will normally take measures to terminate access to the logical drive. Thus, at step 35, a user may request access to another drive or implement alternative local processing operations. In either event, the workstation operating system issues a dismount command to the server 20 at step 36. This dismount command is required when the operating system has been given access to real dismountable drives which, as previously stated, is acted upon by the server 20 so as to complete the housekeeping procedures.

At step 37 the server 20 acts upon the dismount command by releasing the logical drive such that it may be accessed by other workstations. Thereafter, at step 38, the server waits for the next user command.

The releasing of a logical drive will include updating the directory for that drive. In order to improve disc access speed, disc directories are cached in memory and directory updates are made locally while the processor has access to the disc. Upon receiving the dismount command, the updated directory information from the cache memory will be rewritten back to the directory on the disc, thereby maintaining the integrity of the directory data stored on the disk.

The system operating the software will be aware of the way in which removable disc directories are handled and the system will include measures for accommodating power failures and program errors etc. Thus, measures can be taken to effect a disc reset, upon detecting that a particular partition has become unavailable or disconnected, whereafter, when access has been regained in that particular drive, information will be read to the effect that no assumptions may be made about the data contained on the disc and it would be necessary to re-assess that data.

Although the system emulates logical drives having, for example, 600 MBytes of available storage, physical space on the RAID 5 drives 21 to 25 is actually allocated dynamically in regions as storage space for the storage of actual data is required. Thus, although users appear to be given access to logical drives having a total of 600 MBytes, space on the actual RAID 5 drives is not divided into 600 MByte partitions. Drives 21 to 25 are divided into blocks of between two and thirty-two MBytes and blocks are allocated dynamically as and when they are required.

The actual size of blocks on the RAID 5 drives may be variable, although it will be assumed herein that, for a particular application, two MByte blocks will be identified. As data is written to a logical drive, via the server 20, the data will physically occupy an identifiable two MByte block. As the volume of data increases beyond two MBytes, the server 20 will identify a new two MByte data block and data originating from the user will then be directed to this new block. Thus, if a user has created a total of five MBytes, the server is required to maintain a list of where these five MBytes actually reside on the drives, in terms of three two MByte blocks. However, as far as the user is concerned, five MBytes of data have been written to on a removable drive having a total of 600 MBytes of available capacity.

At a workstation, a user is presented with the user interface capable of providing an environment for allowing existing logical drives to be selected and for new logical drives to be defined. The user interface 61 is in turn supported by a local operating system 62, which is responsible for generating commands which are in turn interpreted by the interface.

As far as the local operating system 62 is concerned, access is being made to a conventional SCSI disc drive and communication is effected over a conventional SCSI interface 63, resident at the workstation, to a server SCSI interface 65. This communication conforms to establish SCSI protocols, thereby substantially reducing the need for embedding bespoke software within the local workstation environments.

A server operating system 66 converts SCSI sector definitions into addressable physical data blocks by means of a look-up table, identified by reference 68. A look-up table is defined for each logical drive and when a logical drive is selected by an operator, its associated look-up table is loaded to an operating area of memory 28 within the server 20. Thus, within the server operating system 66, a logical drive is identified, resulting in a table 68 being loaded. Thereafter, SCSI sector selections are supplied as inputs to the table, which then results in addresses for physical data blocks being generated as outputs. Thus, as illustrated in Figure 4, the table 68 effectively points to addressable data blocks 69 in the array of physical data storing discs 21 to 25.

The server operating system 66 allows the SCSI environment of the user terminal to interface with the emulated environment of the server. Thus, it is necessary for the server operating system to emulate an SCSI disc drive and procedures for performing this emulation are detailed in Figure 5.

The procedures shown in Figure 5 are executed within a multi-tasking environment, such that similar procedures may be performed for each of the user terminals. The procedures shown in Figure 5 therefore represent instructions executed on behalf of a particular workstation.

At step 71 the system waits for a workstation command and upon receiving such a command a question is asked at step 72 as to whether this is a "mount" command. A "mount" command instructs the server to mount a selected removable drive and data transfers via the server 20 can only be performed if the server has received such an instruction. Thus, if the question asked at step 72 is answered in the negative, control is directed to step 73, whereupon procedures are performed to emulate an empty drive. Thus, this would include the generation of error messages to the effect that the drive is not ready etc.

If an instruction to mount a drive is generated by the workstation, the question asked at step 72 is answered in the affirmative, resulting in control being directed to step 74. At step 74 a question is asked as to whether the drive is free and if another user workstation has been given access to that particular drive, the question asked at step 74 will be answered in the negative, resulting in a reply being generated at step 75 to the effect that the drive is not ready. Thereafter, control is returned to step 71. However, if the drive is free the question asked at step 74 is answered in the affirmative, resulting in control being directed to step 76.

At step 76 a partition is identified representing the regions within which data for the emulated drive may be read from or written to. Thereafter, control is directed to step 77, whereupon a reply is returned back to the requesting workstation to the effect that the disk has been mounted and control is directed to step 78.

At step 78 the server waits for further commands from the user workstation and in response to receiving such a command, a question is asked at step 79 as to whether this is a dismount command. If the command is not a dismount command further emulation of a removable disc is performed at step 81 and control is returned to step 78.

Upon detecting a dismount command at step 79, control is directed to step 81, whereupon the partition is de-allocated and a reply is issued to the user workstation at step 82 to the effect that the disc has been dismounted. Thereafter control is returned to step 71, whereupon the server waits for the next workstation command.

CLAIMS

1. A method of storing data, wherein a large storage volume emulates a plurality of logical drives; said logical drives emulate removable disc drives; and the closing of access to a previously accessed logical drive generates a disc dismount command.

2. A method according to claim 1, wherein the logical drives emulate removable SCSI drives.

3. A method according to claim 2, wherein each of said logical drives provides between 200 MBytes and 900 MBytes of data storage.

4. A method according to any of claims 1 to 3, wherein data is written to the physical storage volume in identifiable blocks.

5. A method according to claim 4, wherein each of said blocks provides between one MByte and sixty-four MBytes of storage.

6. A method according to claim 4 or claim 5, wherein a mapping table maps sectors of an emulated disc onto blocks of the physical volume.

7. A method according to claim 4 or claim 5, wherein blocks are allocated dynamically as storage is required.

8. A method according to any of claims 1 to 7, wherein the storage volume is implemented as an array of disc storage devices.

9. A method according to claim 8, wherein the array has redundant discs.

10. A method according to claim 8 or claim 9, wherein the array has between four and twelve discs.

11. A method according to any of claims 1 to 10, wherein directory information stored on an accessed disc is updated in response to a disc dismount command.

12. A method according to any of claims 1 to 10, wherein directory information stored on an accessed disc is updated on detecting that a user terminal has been disconnected and can no longer access a previously accessed logical drive.

13. Data storage apparatus, including a large storage volume;

a control device arranged to control data transfer with said storage volume and to provide user terminal access to said storage volume by emulating the presence of a plurality of removable disc drives, wherein

user terminals generate a disc dismount command when closing access to a previously accessed logical drive; and

the control device responds to said disc dismount command by terminating connection to said previously connected logical drive.

14. Apparatus according to claim 13, wherein the logical drives emulate removable SCSI drives.

15. Apparatus according to claim 14, wherein each of said logical drives provides between 200 MBytes and 900 MBytes of data storage.

16. Apparatus according to any of claims 13 to 15, wherein the control device is arranged to write data to the physical storage volume in the form of identifiable blocks.

17. Apparatus according to claim 16, wherein each of blocks provides between 1 MByte and 64 Bytes of storage.

18. Apparatus according to claim 16 or claim 17, wherein the control device is arranged to access mapping tables, mapping sectors of an emulated disc onto blocks of the physical volume.

19. Apparatus according to any of claims 16 to 18, wherein the control device is arranged to dynamically allocate blocks as storage is required.

20. Apparatus according to any of claims 13 to 19, where the storage volume is implemented as an array of disc storage devices.

21. Apparatus according to claim 20, wherein the array includes redundant discs.

22. Apparatus according to claim 20 or claim 21, wherein the array has between four and 12 discs.

23. Apparatus according to any of claims 13 to 22, wherein the control device is arranged to read directory information from an accessed logical drive, and the directory information stored on the disc is updated in response to a disc dismount command.

24. Apparatus according to any of claims 13 to 22, wherein the control device is arranged to read directory information from an accessed logical drive and directory information stored on a logical disc drive is updated by the control device in response to detecting that a user terminal has been disconnected and can no longer access a previously accessed logical drive.

25. A method of storing data substantially as herein described with reference to the accompanying drawings.

26. A data storage apparatus substantially as herein described with reference to the accompanying drawings.

The
Patent
Office

19

Application No:	GB 9502377.6	**Examiner:**	Geoff Western
Claims searched:	1-26	**Date of search:**	3 May 1995

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.N): G4A (AFS, AMX)

Int Cl (Ed.6): G06F (3/06)

Other: On-line : WPI, INSPEC, COMPUTER DATABASE

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
A	EP-0078683-A2 (FUJITSU) See whole document	-
A	Dialog record 01425541 of UNIX Review, vol 9, No 4, April 1991, page 98	-

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		

An Executive Agency of the Department of Trade and Industry

63

Bibliographic Fields

Document Identity

(19)【発行国】	(19) [Publication Office]
日本国特許庁(JP)	Japan Patent Office (JP)
(12)【公報種別】	(12) [Kind of Document]
公開特許公報(A)	Unexamined Patent Publication (A)
(11)【公開番号】	(11) [Publication Number of Unexamined Application]
特開平8−230895	Japan Unexamined Patent Publication Hei 8- 230895
(43)【公開日】	(43) [Publication Date of Unexamined Application]
平成8年(1996)9月10日	1996 (1996) September 10*

Public Availability

(43)【公開日】	(43) [Publication Date of Unexamined Application]
平成8年(1996)9月10日	1996 (1996) September 10*

Technical

(54)【発明の名称】	(54) [Title of Invention]
穀類貯蔵用の袋体	**BAG FOR CEREAL STORAGE**
(51)【国際特許分類第 6 版】	(51) [International Patent Classification, 6th Edition]
B65D 30/10	B65D 30/10
81/20	81/20
【FI】	[FI]
B65D 30/10 B	B65D 30/10 B
81/20 B	81/20 B
【請求項の数】	[Number of Claims]
3	3
【出願形態】	[Form of Application]
OL	OL
【全頁数】	[Number of Pages in Document]
3	3

Filing

【審査請求】	[Request for Examination]
有	*
(21)【出願番号】	(21) [Application Number]
特願平7−39844	Japan Patent Application Hei 7- 39844
(22)【出願日】	(22) [Application Date]
平成7年(1995)2月28日	1995 (1995) February 28*

Parties

Applicants

(71)【出願人】	(71) [Applicant]
【識別番号】	[Identification Number]
595029554	595029554
【氏名又は名称】	[Name]
樋口 秀一	**HIGUCHI HIDEKAZU**
【住所又は居所】	[Address]
新潟県三島郡与板町大字本与板3504番地	Niigata Prefecture Mishima-gun *sheet *Oaza **sheet 3504address

Inventors

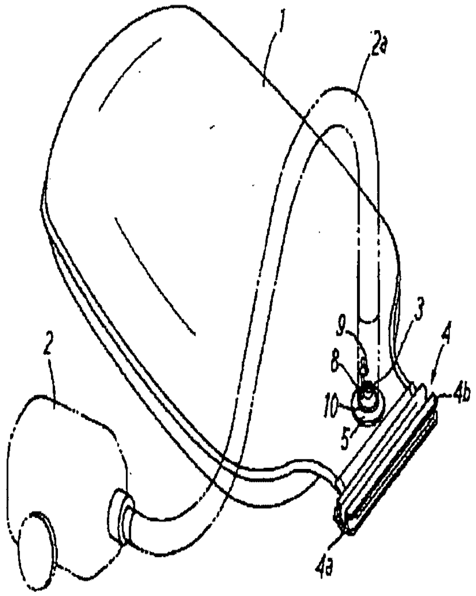
(72)【発明者】	(72) [Inventor]
【氏名】	[Name]
樋口 秀一	Higuchi Hidekazu
【住所又は居所】	[Address]
新潟県三島郡与板町大字本与板3504番地	Niigata Prefecture Mishima-gun *sheet *Oaza **sheet 3504address

Agents

(74)【代理人】	(74) [Attorney(s) Representing All Applicants]
【弁理士】	[Patent Attorney]
【氏名又は名称】	[Name]
吉井 昭栄 (外2名)	Yoshii *Sakae (2 others)

Abstract

(57)【要約】	(57) [Abstract]
【目的】	[Objective]
本発明は米の長期貯蔵を簡単に行い得る穀類貯蔵用の袋体を提供することを目的とする。	this invention designates that bag for cereal storage which can do long-term storage of rice simply is offered as objective .
【構成】	[Constitution]
米,麦,大豆等の穀類を貯蔵する穀類貯蔵用の袋体であって、一側に開口部 1aを形成した袋体 1 を設け、該開口部 1a を適宜な手段により密封可能に構成し、公知の掃除機 2 により内部の空気を吸引する吸引部 3 を該袋体 1 に設けたものである。	With bag for cereal storage which stores rice , cereal grain , soybean or other cereal , bag 1 which formed opening 1a in one side is provided, said opening part 1a sealing up configuration is possibly done with appropriate means , it is somethingwhich provides aspiration part 3 which air of interior is absorbed with cleaner 2 of public knowledge in said bag 1.



Claims

【特許請求の範囲】

【請求項1】

米,麦,大豆等の穀類を貯蔵する穀類貯蔵用の袋体であって、一側に開口部を形成した袋体を設け、該開口部を適宜な手段により密封可能に構成し、公知の掃除機により内部の空気を吸引する吸引部を該袋体に設けたことを特徴とする穀類貯蔵用の袋体。

【請求項2】

請求項1記載の穀類貯蔵用の袋体において、袋体の開口部を基部に対して巾細に形成したことを特徴とする穀類貯蔵用の袋体。

【請求項3】

請求項1,2いずれか1項に記載の穀類貯蔵用の袋体において、袋体の開口部を挟持する挟持体を設けたことを特徴とする穀類貯蔵用の袋体。

Specification

【発明の詳細な説明】

【0001】

[Claim(s)]

[Claim 1]

With bag for cereal storage which stores rice , cereal grain , soybean or other cereal , bag which formed opening in one side is provided, said opening part sealing up configuration is possibly done with appropriate means , bag . for cereal storage which designates that aspiration part which air of interior is absorbed with cleaner of public knowledge is provided in the said bag as feature

[Claim 2]

In bag for cereal storage which is stated in Claim 1, the opening of bag was formed in width detail vis-a-vis base the bag . for cereal storage which designates that as feature

[Claim 3]

In bag for cereal storage which is stated in Claim 1 , 2any one claim , the bag . for cereal storage which designates that clamps which the opening of bag clamping is done is provided as feature

[Description of the Invention]

[0001]

【産業上の利用分野】

本発明は、米,麦,大豆等の穀類を長期間貯蔵するに便利な穀類貯蔵用の袋体に関するものである。

【0002】

【従来の技術及び発明が解決しようとする課題】

従来、米,麦,大豆等の穀類を長期間貯蔵する袋体として、袋体内に脱酸素剤を配設する穀類貯蔵専用の袋体(以下、従来例)が提案されている。

【0003】

この従来例は、袋体の下方に透明なフィルムを貼着して内部が視認できる窓部を形成し、この袋体の中に米等の穀類を収納したら脱酸素剤を当該窓部位置に配設し、脱酸素剤の変色(酸素を吸着すると変色する。)を視認しながら米等を長期間貯蔵するものである。

【0004】

しかしながら、脱酸素剤は酸素の吸着作用が所定期間しか発揮されず、よって、当該従来例の場合、適宜脱酸素剤を交換しなければならない。

【0005】

ところで、この脱酸素剤の交換には当然袋体の開け閉めが伴うことになるが、この袋体の開け閉めにより酸素が少なくなっている袋体内に再び酸素が流入し、従って、また、一から酸素の吸着除去をしなければならず、結局、この穀物貯蔵専用の袋体は無駄が多く、非効率的である。

【0006】

本発明は問題を解決した穀類貯蔵用の袋体を提供するものである。

【0007】

【課題を解決するための手段】

添付図面を参照して本発明の要旨を説明する。

【0008】

米,麦,大豆等の穀類を貯蔵する穀類貯蔵用の袋体であって、一側に開口部 1aを形成した袋体1を設け、該開口部 1aを適宜な手段により密封可能に構成し、公知の掃除機 2 により内部の空

[Field of Industrial Application]

rice , cereal grain , soybean or other cereal long period it stores this invention, it is something regarding the bag for convenient cereal storage.

[0002]

[Prior Art And Problems To Be Solved By The Invention]

Until recently, bag (Below, Prior Art Example) of cereal storage dedicated which arranges oxygen scavenger inside bag long period is stored rice , cereal grain , soybean or other cereal as bag which, is proposed.

[0003]

As for this Prior Art Example , adhering doing transparent film in lower of bag ,interior when window portion which visible it is possible is formed and rice or other cereal is stored up in this bag , while arranging oxygen scavenger in the this said window portion position , visible doing color change (When oxygen it adsorbs, it changes color.) of oxygen scavenger it issomething which rice etc long period is stored.

[0004]

But, oxygen scavenger is shown, depends and adsorption action of oxygen only the specified time when it is a this said Prior Art Example , must exchange as needed oxygen scavenger .

[0005]

It means that by way, opening closing of bag accompaniesexchange of this oxygen scavenger naturally, but oxygen flows into the bag where oxygen has decreased depending upon opening closingof this bag again, therefore, in addition, if adsorptive elimination of oxygen is not done from one, it does not become, after all, as for bag of this grain storage dedicated waste is many, it is a inefficient .

[0006]

this invention is something which offers bag for cereal storagewhich solves problem .

[0007]

[Means to Solve the Problems]

Referring to attached figure , you explain gist of this invention .

[0008]

It is something which relates to bag for cereal storage whichdesignates that aspiration part 3 where with bag for cereal storagestoring rice , cereal grain , soybean or other cereal , it provides bag 1 which formed opening 1a in the one

気を吸引する吸引部3を該袋体1に設けたことを特徴とする穀類貯蔵用の袋体に係るものである。

【0009】

請求項1記載の穀類貯蔵用の袋体において、袋体1の開口部1aを基部に対して巾細に形成したことを特徴とする穀類貯蔵用の袋体に係るものである。

【0010】

請求項1,2いずれか1項に記載の穀類貯蔵用の袋体において、袋体1の開口部1aを挟持する挟持体4を設けたことを特徴とする穀類貯蔵用の袋体に係るものである。

【0011】

【作用】

袋体1に米等の穀類を収納し、開口部1aを適宜な手段で密封し、吸引部3に公知の掃除機2を連設して袋体1内の空気を吸引する。

【0012】

【実施例】

図面は本発明の一実施例を図示したもので、以下に説明する。

【0013】

本実施例の袋体1は適度に強度を有する透明な合成樹脂部材で成形する。

該袋体1の上部は先細り状に形成され、この先細り部の端部が開口部1aに設定される。

【0014】

この開口部1aは適宜な合成樹脂で成形した挟持凹体4aと挟持凸体4bからなる公知の挟持体4により挟持する。

具体的には挟持凹体4aの凹条に挟持凸体4bの凸条を嵌入して両者により開口部1aを閉塞する。

【0015】

袋体1の上端側には吸引部3が形成されている。

この吸引部3は袋体1に付設される合成樹脂製の止着体8に突設されている。

side , sealing up configuration does said opening part 1a possibly with theappropriate means , absorbs air of interior with cleaner 2 of public knowledge is provided in said bag 1 as feature.

[0009]

In bag for cereal storage which is stated in Claim 1, the opening 1a of bag 1 was formed in width detail vis-a-vis base itis something which relates to bag for cereal storage whichdesignates that as feature.

[0010]

It is something which relates to bag for cereal storage whichdesignates that clamps 4 which opening 1a of bag 1 clamping isdone is provided as feature in bag for cereal storage whichis stated in Claim 1 , 2any one claim .

[0011]

[Working Principle]

rice or other cereal is stored up in bag 1, opening 1a is sealed up with theappropriate means , cleaner 2 of public knowledge is connected to aspiration part 3 and air inside bag 1 is absorbed.

[0012]

[Working Example(s)]

Being something which illustrates one Working Example of this invention , you explain drawing below.

[0013]

As for bag 1 of this working example it forms with transparent synthetic resin member which possesses strength moderately.

upper part of said bag 1 is formed by taper , end of this taper section is set to opening 1a.

[0014]

clamping it does this opening 1a with clamping concave body 4 a and clamps 4 formed with appropriate synthetic resin of public knowledge which whichconsists of clamping convex body 4 b.

Inserting convex stripe of clamping convex body 4 b in recessed rib of clamping concave body 4 a concretely, opening 1a it is plugged by both .

[0015]

aspiration part 3 is formed to top end of bag 1.

this aspiration part 3 is installed in afixing body 8 of synthetic resin which is installedin bag 1.

この止着体8は袋体1に穿設された窓孔位置に止着されるものであって、止着体 8 の外周に繞設した止着板5で窓孔周縁を挟持して袋体1に止着される。

符号 6 は弁、7 は米等が吸引されることを防止するフィルター、9 は栓、10 は掃除機 2 の吸引ホース 2a を隙間なく可及的に密着状態にする為の柔軟板である。

【0016】

本実施例は上述のように構成したから、袋体 1 内に例えば米を収納し、該袋体 1 の開口部 1a を挟持体4で挟持して袋体1を密封する(開口部 1a は折り返して挟持する。)。

この密封された状態で袋体1の吸引部3に掃除機 2 の吸引ホース 2a を被嵌してその下端を柔軟板10に当接せしめ、掃除機2を作動させて袋体 1 内の空気を吸引すると、袋体 1 内は排気され可及的に真空状態となる。

【0017】

よって、米を長期間貯蔵する場合には適宜掃除機 2 で袋体 1 内を排気するという簡単な作業で済むことになる。

【0018】

また、本実施例の吸引部3は弁 6 が設けられている為、袋体 1 内の真空状態が不良となって再吸引する際、栓 9 を開放しても袋体 1 内に空気が流入することは確実に防止され、前記した従来例に比し効率的に米の貯蔵を行い得ることになる。

【0019】

更に、本実施例は酸素のみを消失させる従来例とは異なり、空気を消失させ、真空状態を作出するものであるから、袋体1の容積が減少し、それだけ袋体 1 の保管スペースが少なくて済むことになるとともに袋体 1 内の水分も除去され、この点においても米の良好な長期貯蔵が可能となる。

【0020】

【発明の効果】

本発明は上述のように構成したから、米等の穀類の長期貯蔵を且つ良好簡単に行い得る秀れた穀類貯蔵用の袋体となる。

this afixing body 8 being something which afixing is done in window hole position which is installed in bag 1, in outer perimeter of afixing body 8 the clamping doing window hole surrounding edge with afixing sheet 5 which * facilities itdoes, afixing is done in bag 1.

As for sign 6 as for valve , 7 as for filter , 9 which prevents thefact that rice etc is absorbed as for plug , 10 it is a softening sheet inorder to designate suction hose 2a of cleaner 2 if possible as closely adhered state without gap .

[0016]

Because above-mentioned way configuration it did this working example , for example rice is stored up inside bag 1, opening 1a of said bag 1 clamping isdone with clamps 4 and bag 1 is sealed up (Turning back, clamping it does opening 1a.).

this with state which is sealed up fitted covering doing suction hose 2a of cleaner 2 in aspiration part 3 of bag 1, bottom end contacting the softening sheet 10, cleaner 2 operating, when it absorbs air inside the bag 1, inside of bag 1 is done and exhaust if possiblebecomes vacuum state .

[0017]

Depending, when long period it stores rice , it means that issufficient simple operation that exhaust it does inside bag 1 with theas needed cleaner 2.

[0018]

In addition, as for aspiration part 3 of this working example because valve 6 isprovided, vacuum state inside bag 1 becoming defect , whenre-absorbing, opening plug 9, as for air flowing into the bag 1 it is prevented securely, before it compares to Prior Art Example whichwas inscribed and it means to be possible to store rice in the efficient .

[0019]

Furthermore, because as for this working example only oxygen it disappears, itis something which produces vacuum state , volume of bag 1 decreases, as it means that that much storage space of bag 1 may belittle and also moisture inside bag 1 is removed air unlike Prior Art Example which disappears, Satisfactory long-term storage of rice becomes possible at this point.

[0020]

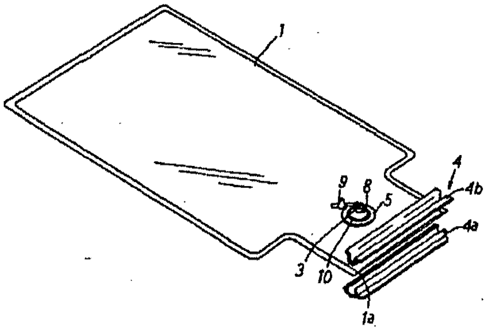
[Effects of the Invention]

Because above-mentioned way configuration it did this invention, it becomes bag for cereal storage which and can do long-term storage of the rice or other cereal satisfactorily simply, is superior.

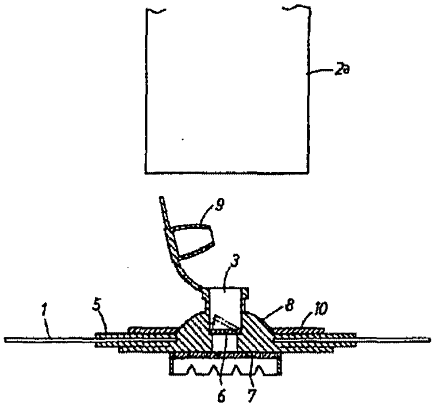
【図面の簡単な説明】	[Brief Explanation of the Drawing(s)]
【図1】	[Figure 1]
本実施例の斜視図である。	It is a oblique view of this working example .
【図2】	[Figure 2]
本実施例の要部の断面図である。	It is a sectional view of principal part of this working example .
【図3】	[Figure 3]
本実施例の使用状態を示す斜視図である。	It is a oblique view which shows use state of this working example .
【符号の説明】	[Explanation of Symbols in Drawings]
1	1
袋体	bag
1a	1a
開口部	opening
2	2
掃除機	cleaner
3	3
吸引部	aspiration part
4	4
挟持体	clamps

Drawings

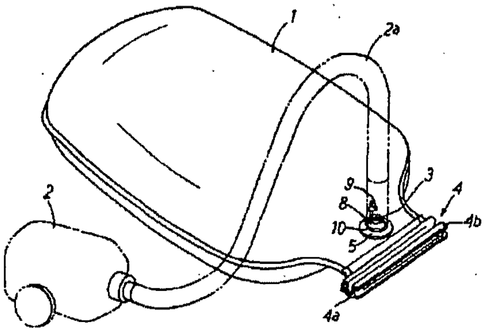
【図1】 [Figure 1]



【図2】 [Figure 2]



【図3】 [Figure 3]



K4

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 810 530 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
03.12.1997 Bulletin 1997/49

(51) Int. Cl.^{6}: G06F 13/368

(21) Application number: 97107935.5

(22) Date of filing: 15.05.1997

(84) Designated Contracting States:
DE FR GB NL SE

(30) Priority: 31.05.1996 US 656641

(71) Applicant:
SUN MICROSYSTEMS, INC.
Mountain View, CA 94043 (US)

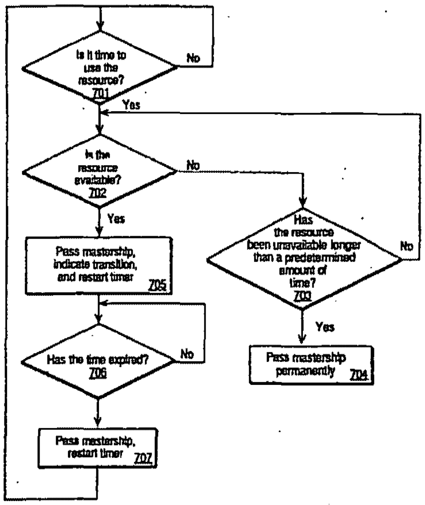
(72) Inventors:
- Schmahl, Kenneth A.
San Jose, CA 95136 (US)
- Tedone, Matthew J.
Sunnyvale, CA 94087 (US)
- Schell, John C.
Sunnyvale, CA 94087 (US)
- Karminsky, Igor
San Jose, CA 95129 (US)
- Chan, Ray P.
Cupertino, CA 95014 (US)

(74) Representative:
Zangs, Rainer E., Dipl.-Ing. et al
Hoffmann Eitle,
Patent- und Rechtsanwälte,
Arabellastrasse 4
81925 München (DE)

(54) A method and apparatus for passing bus mastership

(57) A method for passing mastership of a bus is described. According to the method, it is determined whether to use the bus. If the bus is to be used, it is determined whether the bus is available. If the bus is available, the bus is accessed and a signal is generated to indicate that the bus is being accessed. A timer is also started and access to the bus is yielded when the timer expires. A processor that passes mastership to a shared resource is also described. The processor comprises a resource accessing unit. The resource accessing unit allows the processor to access a resource upon receiving a first signal from a component coupled to the resource. The resource accessing unit yields access of the resource to the component upon receiving a second signal from the component.

FIG. 7


Is it time to use the resource? 701
No
Yes
Is the resource available? 702
No
Yes
Pass mastership, indicate transition, and restart timer 705
Has the resource been unavailable longer than a predetermined amount of time? 703
No
Yes
Has the time expired? 706
No
Pass mastership permanently 704
Pass mastership, restart timer 707

Printed by Rank Xerox (UK) Business Services
2.14.23/3.4

Description

FIELD OF THE INVENTION

The present invention pertains to the field of bus regulation. More specifically, the present invention relates to an apparatus and method for passing bus mastership between multiple devices.

BACKGROUND OF THE INVENTION

When multiple devices reside on a bus, coordination of access to the bus is necessary. Coordination of access to the bus insures that multiple devices desiring to communicate will not assert control and data lines for different transfers at the same time and cause bus contention.

One approach to coordinating bus access is the use of one or more bus masters in the system. A bus master controls access to the bus. It initiates and controls all bus requests. A processor must be able to initiate a bus request for access to a memory device and thus is always a bus master. A memory device is usually a slave since it will respond to read and write requests but never generate its own requests.

A bus has multiple masters when there are multiple central processing units (CPUs) or when input/output (I/O) devices can initiate a bus transaction. If there are multiple masters, an arbitration scheme is required among the masters to decide who gets the bus next. A bus arbiter is typically used to implement the arbitration scheme. In a bus arbitration scheme, a device wanting to use the bus signals a bus request and is later granted the bus. After a grant, the device can use the bus, later signaling to the bus arbiter that the bus is no longer required. The bus arbiter can then grant the bus to another device. Most multiple-master buses have a set of bus signals for performing requests and grants. A bus release line is also needed if each device does not use its own request line to release the bus. Sometimes the signals used for bus arbitration have physically separate lines, while in other systems the data lines of the bus are used for this function. Arbitration is often a fixed priority, as is the case with daisy-chained devices or an approximately fair scheme that randomly chooses which master gets the bus.

The use of a bus arbiter has several drawbacks. The addition of a bus arbiter requires additional power to operate. This is a problem for computer systems operating under tight power constraints. Implementing a bus arbiter also requires additional space in the computer system. Thus, depending upon the environment of the computer system, the availability of physical space may not permit the implementation of a bus arbiter. Perhaps most importantly, the use of an additional component for the purpose of arbitration adds an undesirable cost to the overall computer system.

Thus, what is needed is an apparatus that passes ownership of a resource between a plurality of devices without using an external arbiter.

SUMMARY OF THE INVENTION

A method for passing mastership of a resource is described. According to the method, it is determined whether to use the bus. If the bus is to be used, it is determined whether the bus is available. If the bus is available, the bus is accessed and a signal is generated to indicate that the bus is being accessed. A timer is also started and access to the bus is yielded when the timer expires.

A processor that passes mastership of a shared resource is described. The processor comprises a resource accessing unit. The resource accessing unit allows the processor to access a resource upon receiving a first signal from a component coupled to the resource. The resource accessing unit yields access of the resource to the component upon receiving a second signal from the component. The processor further comprises a signal generation unit. The signal generation unit is coupled to the resource accessing unit. The signal generation unit generates a third signal when the processor has gained access to the resource and generates a fourth signal when the processor has yielded access to the resource.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not by way of limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

Figure 1 illustrates a multi-processor computer system implementing an embodiment of the invention;
Figures 2 illustrates processors from two different computer systems implementing an embodiment of the invention;
Figures 3 illustrates the present invention as implemented in a mass storage system;
Figure 4 is a table illustrating the mastership states in one embodiment of the present invention;
Figure 5 is a state diagram illustrating the transition order of the states illustrated in Figure 4;
Figure 6 illustrates a block diagram of one embodiment of a processor implementing the present invention; and
Figure 7 is a flow chart illustrating a method of passing mastership of a shared resource.

DETAILED DESCRIPTION

A method and apparatus for accessing data in a memory is described. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

Referring to Figure 1, the computer system upon which the preferred embodiment of the present invention can be implemented is shown as 100. Computer system 100 comprises a bus or other communication means 101 for communicating information, and processors 102 and 103 coupled with bus 101 for processing information. System 100 further comprises a random access memory (RAM) or other dynamic storage device 104 (referred to as main memory), coupled to bus 101 for storing information and instructions to be executed by processors 102 and 103. Main memory 104 also may be used for storing temporary variables or other intermediate information during execution of instructions by processors 102 and 103. Computer system 100 also comprises a read only memory (ROM) and/or other static storage device 106 coupled to bus 101 for storing static information and instructions for processors 102 and 103. Data storage device 107 is coupled to bus 101 for storing information and instructions. Instructions from a computer readable media which are executable by processors 102 or 103 may be stored onto data storage device 107. A data storage device 107 such as a magnetic disk or optical disk and its corresponding disk drive can be coupled to computer system 100.

Computer system 100 can also be coupled via bus 101 to a display device 121, such as a cathode ray tube (CRT), for displaying information to a computer user. An alphanumeric input device 122, including alphanumeric and other keys, is typically coupled to bus 101 for communicating information and command selections to processors 102 and 103. Another type of user input device is cursor control 123, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 102 and for controlling cursor movement on display 121. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), which allows the device to specify positions in a plane.

Alternatively, other input devices such as a stylus or pen can be used to interact with the display. A displayed object on a computer screen can be selected by using a stylus or pen to touch the displayed object. The computer detects the selection by implementing a touch sensitive screen. Similarly, a light pen and a light sensitive screen can be used for selecting a displayed object. Such devices may thus detect selection position and the selection as a single operation instead of the "point and click," as in a system incorporating a mouse or trackball. Stylus and pen based input devices as well as touch and light sensitive screens are well known in the art. Such a system may also lack a keyboard such as 122 wherein all interface is provided via the stylus as a writing instrument (like a pen) and the written text is interpreted using optical character recognition (OCR) techniques.

Figure 1 illustrates one embodiment of the present invention where bus 101 is shared between two processors 102 and 103 in the same computer system 100. In order to prevent bus contention, only one of processors 102 or 103 may access bus 101 at one time. Processor 102 is only allowed to access bus 101 during its designated bus mastership state. Similarly, processor 103 is only allowed to access bus 101 during its designated bus mastership state. The bus mastership state of the system is determined by tokens or signals that processors 102 and 103 generate. In one embodiment of the present invention, processors 102 and 103 generate a signal on line 130 each time they gain access to bus 101, relinquish access to bus 101 or wish to gain access to bus 101. In another embodiment of the present invention, the signal generated by one of the processors on line 130 may be a single signal or a plurality of signals. The signals generated by processor 102 are sent to processor 103 via line 130 and the signals generated by processor 103 are sent to processor 102 via line 130. Each processor has a copy of the signals generated by itself and the signals generated by the other processor. Each processor is aware of the current bus mastership state of the system 100.

Figure 2 illustrates an embodiment of the present invention where a processor 102 from a first computer system 250 and a second processor 202 from a second computer system 251 share access to a shared resource 210. Shared resource 210 is a resource which may be accessed by only one of either processor 102 or processor 202 at one time. Shared resource 210 may be, for example, a bus or a memory. Shared resource 210 may be directly coupled to processor 102 and 202 or coupled to processors 102 and 202 via other buses or components. Processor 102 is only allowed to access shared resource 210 during its designated resource mastership state. Processor 202 is only allowed to access shared resource 210 during its designated resource mastership state. The resource mastership state of the systems is determined by tokens or signals that the processors 102 and 202 generate. In one embodiment of the present invention, processors 102 and 202 generate a signal each time they gain access to shared resource 210, relinquish access to shared resource 210 or wish to gain access to shared resource 210. In one embodiment of the present invention, the signal generated by the processor 102 or 210 may be a single signal or a plurality of signals. The signals generated by processor 102 are sent to processor 202 on line 230 and the signals generated by processor 202 are sent to processor 102 on line 230. Each processor has a copy of the signals generated by itself and the other processor. Each processor is aware of the current bus mastership state of the computer systems.

Figure 3 illustrates an embodiment of the present invention as implemented in a mass storage system 300. Mass storage system 300 comprises a first array of storage elements 335 coupled to a hard disk assembly

331 and a second array of storage elements 345 coupled to a hard disk assembly 341. The first and second array of storage elements 335 and 345 are accessed by a host (not shown) via one of the host interface units 304 or 314 and one of buses 301 or 311. Buses 301 and 311 maybe implemented, for example, by a conventional fiber channel interface, a serial storage architecture interface, a small computer system interface (SCSI), a P1394 interface, or other well known interfaces. Hard disk assembly 331 comprises to interface the first array of storage elements 335 with bus 301. Hard disk assembly 331 includes a register 332 which is used for storing data to be read by processors 302 and 312. Hard disk assembly 341 operates to interface the second array of storage elements 345 with bus 311. Hard disk assembly 341 includes a register 342 which is used for storing data to be read by processors 302 and 312.

An environmental service center 325 provides environmental services such as temperature control and power to mass storage system 300. Environmental service center 325 also provides data regarding the environmental services of mass storage system 300. Environmental service center 325 may be implemented by any known circuitry. Processor 302 is coupled to bus 301 and shared bus 320. Processor 302 polls the environmental service center 325 by reading environmental service data from environmental service center 325 via shared bus 320. Processor 302 stores the environmental service data in memory unit 303. Processor 302 operates to monitor the environment of mass storage system 300 and maintains the system's integrity when the environment is out of tolerance range. Similarly, processor 312 is coupled to bus 311 and shared bus 320. Processor 312 polls the environmental service center 325 by reading environmental service data from environmental service center 325 via shared bus 320. Processor 312 stores the environmental service data in memory unit 313. Processor 312 operates to monitor the environment of mass storage system 300 and maintains the system's integrity when the environment is out of tolerance range.

Environmental service data from environmental service center 325 may only be accessed by one of processors 302 and 312 via shared bus 320 at a time. Processor 302 is only allowed to access shared bus 320 during its designated bus mastership state. Processor 312 is only allowed to access shared bus 320 during its designated bus mastership state. The bus mastership state of the system 300 is determined by tokens or signals that processors 302 and 312 generate. In one embodiment of the present invention, the bus mastership state is changed by signals generated by processors 302 or 312 when one of the processors gains access to bus 320, relinquishes access to bus 320, or wishes to gain access to bus 320. In another embodiment of the present invention, the signal generated by each processor 302 or 312 may be a single signal or a plurality of signals. In still another embodiment of the present invention, a timer 355 in processor 302 and a timer 356 in processor 312 is set each time mastership of shared bus 320 is taken by a new master. The mastership of shared bus 320 is passed each time the timers 355 and 356 time out. The signals generated by processor 302 are sent to processor 312 via line 350 and the signals generated by processor 312 are sent to processor 302 via line 350. Each processor has a copy of the signals generated by itself and the other processor. Each processor 302 or 312 is aware of the current bus mastership state of the system 300.

In one embodiment of the present invention, there are four bus mastership states recognized by processors 302 and 312 of system 300. Figure 4 is a table illustrating the four states. At state 1, processor 302 (Device 1) has mastership of shared bus 320. State 1 occurs when processor 302 generates a 0 signal and processor 312 (Device 2) generates a 0 signal on line 350. At state 2, bus mastership is to be transferred from processor 302 to processor 312. State 2 occurs when processor 302 generates a 1 signal and processor 312 generates a 0 signal on line 350. At state 3, processor 312 has mastership of shared bus 320. State 3 occurs when processor 302 generates a 1 signal and processor 312 generates a 1 signal on line 350. At state 4, bus mastership is to be transferred from processor 312 to processor 302. State 4 occurs when processor 302 generates a 0 signal and processor 312 generates a 1 signal on line 350. Figure 5 is a state diagram illustrating the order in which states 1-4 shown in Figure 4 are executed. It should be appreciated that the number of states, the order in which the states are executed, and the number of signals used to represent the states may change depending on the implementation of the present invention.

Figure 6 illustrates one embodiment of processor 302. Processor 302 includes computation and control unit 610. In one embodiment of the present invention, computation and control unit 610 includes two fiber channel arbitrated loop ports, a block of embedded RAM, a host bus interface, and a processing unit. Computation and control unit 610 operate to poll environmental service data from the environmental service center and to control the environment of computer system 300.

Processor 302 further includes resource accessing unit 620, timer 355, and signal generation unit 631. Resource accessing unit 620 keeps track of the bus mastership states of memory storage system 300 and signals computation and control units 610 to poll the environmental service center 325 when processor 302 receives mastership of shared bus 320. Resource accessing unit 620 receives signals from processor 312 via line 350 which indicate when processor 320 is ready to transition into a next state. Resource accessing unit 620 is coupled to timer 355. Resource accessing unit 620 resets timer 355 when mastership of bus 320 is taken by a new master. After a predetermined amount of time, timer 355 times out. This informs resource accessing unit 620 that shared bus 320 is to be passed to another master. Resource accessing unit 620 instructs signal generation unit 630 to generate a signal on line 631 to indicate that processor 302 is ready to transition into the next state. The bus mastership state of system 300 is determined by the signals generated by processors 302 and 312. Resource accessing unit 620, timer 355 and signal generation unit 630 may be implemented in hardware, software or a combination of hardware and software. In the embodiment of the invention shown in Figure 6, resource accessing unit 620, timer 355, and signal generation unit 630 are implemented in hardware external to computation and control unit 610. In an alternate embodiment of the present invention, resource accessing unit 620 and signal generation unit 630 are software modules implemented by a set of instructions executed by processor 302. Processor 312 operates similarly to processor 302 and may be implemented by the same components which may be used to implement processor 302.

The present invention allows arbitration of mastership to a shared resource between two devices where neither is master of the other without the use of an external arbiter. In a preferred embodiment of the present invention where the resource accessing unit and signal generation unit is implemented in software, arbitration is achieved without requiring additional power or space from the system.

Although Figure 6 illustrates an embodiment of the present invention where resource accessing unit 620, signal generation unit 630 and timer 355 reside inside processor 302, it should be appreciated that these components may reside in any agent sharing access to a shared resource to arbitrate access to the shared resource.

In one embodiment of the present invention, processor 302 updates the environmental service data in main memory 313 after processor 302 has polled environmental service data from environmental service center 325 and while system 300 is in a state where processor 302 has bus mastership of shared bus 320. In this embodiment of the present invention, processor 312 also updates the environmental service data in main memory 303 after processor 312 has polled environmental service data from environmental service center 325 and while system 300 is in a state where processor 312 has bus mastership of shared bus 320.

Processor 302 updates the environmental service data in main memory 313 through a data exchange. A second line (not shown) is used to communicate mastership of shared bus 320 between processors 302 and 312 during the data exchange in a manner similar to which line 350 communicates mastership of shared bus 320 during data polling. Processor 302 writes environmental service data into registers 332 and 342 of hard disk assembly 332 and 342 when it has mastership of shared bus 320 during data exchange. Processor 312 reads the environmental system data from registers 332 and 342 when it has mastership of shared bus 320 during data exchange and stores the data into memory unit 313. Processor 302 continues to write new data into registers 332 and 342 until all the environmental service data in memory unit 303 has been written into registers 332 and 342 and transferred into main memory 313. Processor 312 operates similarly to processor 302 in updating the environmental service data in memory unit 303 when system 300 is in a state where processor 312 has mastership of shared bus 320. In an alternate embodiment of the present invention, a single line and a single set of signals are used by processors 302 and 312 to pass mastership of shared bus 320 during polling and exchange of environmental service data.

In a situation where processor 302 becomes inoperable and fails to generate a signal to processor 312 indicating that it is ready to transition into the next bus mastership state within a predetermined period of time, a timer in processor 312 will time out. This will indicate to processor 312 that processor 302 is inoperable. In response, processor 312 will take exclusive bus mastership of shared bus 320. Similarly, in a situation where processor 312 inoperable and fails to generate a signal to processor 312 indicating that it is ready to transition into the next bus generation state within a predetermined period of time, a timer in processor 302 will time out. This will indicate to processor 302 that processor 312 is inoperable. In response, processor 302 will take exclusive bus mastership of shared bus 320.

Figure 7 is a flow chart illustrating a method for passing mastership of a shared resource between two devices. At step 701, it is determined whether to use the shared resource. This determination may be made by checking a timer which records the time a first device has had access to the resource. After a first predetermined amount of time, the timer times out indicating that it is time for the second device to access the shared resource. If it is not time to use the shared resource, control returns to step 701. If it is time to use the shared resource, control proceeds to step 702.

At step 702, it is determined whether the shared resource is available. This determination may be made by checking a resource accessing unit for the current resource mastership state. If the resource mastership state is one where the first device has mastership, the shared resource is unavailable and control proceeds to step 703. If the shared resource is available, control proceeds to step 705.

At step 703, it is determined whether the first device has had mastership of the shared resource for over a second predetermined amount of time. This determination may be made by checking the timer which records the time when the first device had access to the shared resource. If the first device did not have mastership of the shared resource for over the second predetermined period of time, control returns to step 702. If the first device did have mastership of the shared resource for over the second predetermined amount of time, control proceeds to step 704.

At step 704, exclusive mastership of the shared resource is given to the second device and the first device is excluded from being considered a possible master of the shared resource in the future.

At step 705, mastership of the shared resource is given to the second device. A signal is generated indicating that the shared resource has been accessed by the second device and the timer is reset.

At step 706, determine whether mastership of the shared resource should be passed to a different device. This determination can be made by checking to see if the timer has timed out past the first predetermined period of time. If the timer has timed out past the first predetermined period of time, it is time to pass mastership of the shared resource to a different resource and control proceeds to step 707. If the timer has not timed out past the first predetermined period of time, control returns to step 706.

At step 707, a signal is generated by the second device indicating that the second device is ready to transition to the next state of resource mastership where it is not the master of the shared resource. Control proceeds to step 701.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

Claims

1. A method for passing bus mastership, comprising:

   determining whether a bus is available;
   accessing the bus and generating a signal indicating that the bus is being accessed if the bus is available;
   starting a timer in response to accessing the bus; and
   yielding access to the bus when the timer expires.

2. The method of claim 1 further comprising the step of re-starting the timer after yielding access to the bus.

3. The method of claim 1 further comprising the step of generating a signal indicating that access to the bus has been yielded.

4. The method of claim 1 further comprising the step of determining whether the bus has been accessed longer than a predetermined amount of time if the bus is unavailable and gaining access to the bus if the bus has been accessed longer than the predetermined amount of time.

5. The method of claim 1, wherein determining whether the bus is available comprises the step of checking to see whether a bus agent has generated a signal indicating that it is accessing the bus.

6. A computer-readable medium having stored thereon sequences of instructions, the sequences of instructions including instructions which, when executed by a processor, cause the processor to perform the steps of:

   determining whether a bus is available;
   accessing the bus and generating a signal indicating that the bus is being accessed if the bus is available;
   starting a timer in response to accessing the bus; and
   yielding access to the bus when the timer expires.

7. The computer-readable medium of claim 6 further comprising instructions which, when executed by the processor, would cause the processor to perform the step of restarting the timer after yielding access to the bus.

8. The computer-readable medium of claim 6 further comprising instructions which, when executed by the processor, would cause the processor to perform the step of generating a signal indicating that access to the bus has been yielded.

9. The computer-readable medium of claim 6 further comprising instructions which, when executed by the processor, would cause the processor to perform the step of determining whether the bus has been accessed longer than a predetermined amount of time if the bus is unavailable and gaining access to the bus if the bus has been accessed longer than the predetermined amount of time.

10. The computer-readable medium of claim 6, wherein the step of determining whether the bus is available comprises the step of checking to see whether a bus agent has generated a signal indicating that it is accessing the bus.

11. A processor, comprising:

    a resource accessing unit allowing the processor to access a resource upon receiving a first signal from a component coupled to the resource and yielding access of the resource to the component upon receiving a second signal from the component.

12. The processor of claim 11 further comprising:

    a signal generation unit, coupled to the resource accessing unit, generating a third signal when the processor has gained access to the resource and generating a fourth signal when the processor has yielded access to the resource.

13. The apparatus of claim 11 further comprising a timer, coupled to the signal generation unit, allocating a time period when the third and fourth signals are generated.

14. The apparatus of claim 11, wherein the component is a second processor.

15. The apparatus of claim 11, wherein the component is a plurality of processors.

16. The apparatus of claim 11, wherein the resource is a bus.

17. The apparatus of claim 11, wherein the resource is a memory.

18. A computer system, comprising

(A) a bus;
(B) a first processor, coupled to the bus, having

(1) a first signal generation unit generating a first signal when the first processor has gained access to the bus and generating a second signal when the first processor has yielded access to the bus; and
(2) a first bus accessing unit allowing the first processor to access the bus upon receiving a third signal and yielding access to the bus upon receiving a fourth signal;

(C) a second processor, coupled to the bus and the first processor, having

(1) a second signal generation unit generating the fourth signal when the second processor has gained access to the bus and generating the third signal when the second processor has yielded access to the bus; and
(2) a second bus accessing unit allowing the second processor to access the bus upon receiving the second signal and yielding access to the bus upon receiving the first signal.

19. The computer system of claim 18 further comprising an array of storage devices coupled to the first and second processors.

20. The computer system of claim 18 further comprising an environmental service center coupled to the bus.

21. A bus arbitrating apparatus residing in a bus agent configured to communicate with a processor based system including a memory, bus, and display, comprising:

a resource accessing unit allowing the bus agent to access the bus upon receiving a first signal from a component coupled to the bus and yielding access of the bus to the component upon receiving a second signal from the component.

22. The bus arbitrating apparatus of claim 21, further comprising:

a signal generation unit, coupled to the resource accessing unit, generating a third signal when the bus agent has gained access to the resource and generating a fourth signal when the bus agent has yielded access to the resource.

23. A system for arbitrating a bus between a first bus agent and a second bus agent comprising:

a first signal generation unit generating a first signal when the first bus agent has gained access to the bus and generating a second signal when the first bus agent has yielded access to the bus;
a first bus accessing unit allowing the first bus agent to access the bus upon receiving a third signal and yielding access to the bus upon receiving a fourth signal, wherein the first signal generation unit and the first bus accessing unit reside inside the first bus agent;
a second signal generation unit generating the fourth signal when the second bus agent has gained access to the bus and generating the third signal when the second bus agent has yielded access to the bus; and
a second bus accessing unit allowing the second bus agent to access the bus upon receiving the second signal and yielding access to the bus upon receiving the first signal, wherein the second signal generation unit and second bus accessing unit reside inside the second bus agent.

24. The system of claim 23 further comprising an array of storage devices coupled to the first and second bus agents.

25. The system of claim 23 further comprising an environmental service center coupled to the bus.

FIG. 1

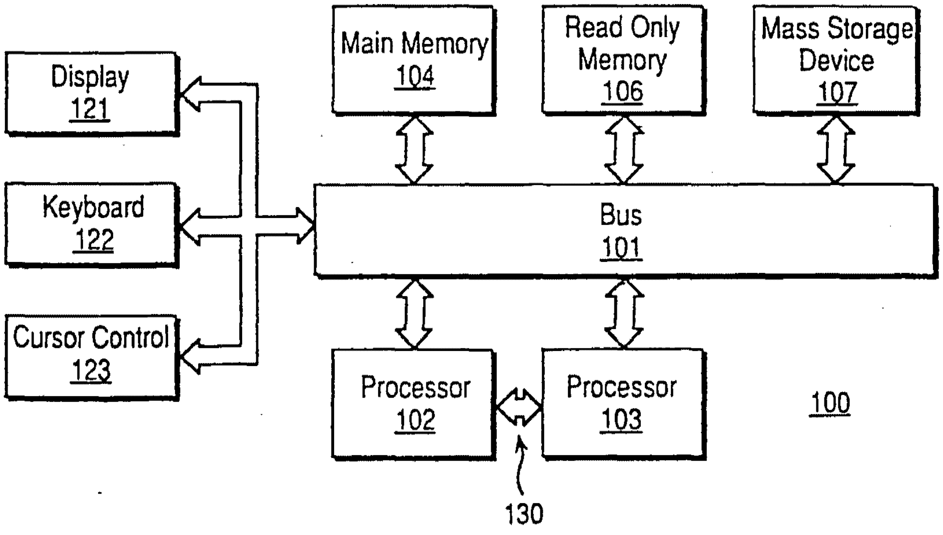

Display
121
Keyboard
122
Cursor Control
123
Main Memory
104
Read Only
Memory
106
Mass Storage
Device
107
Bus
101
Processor
102
Processor
103
130
100

FIG. 2

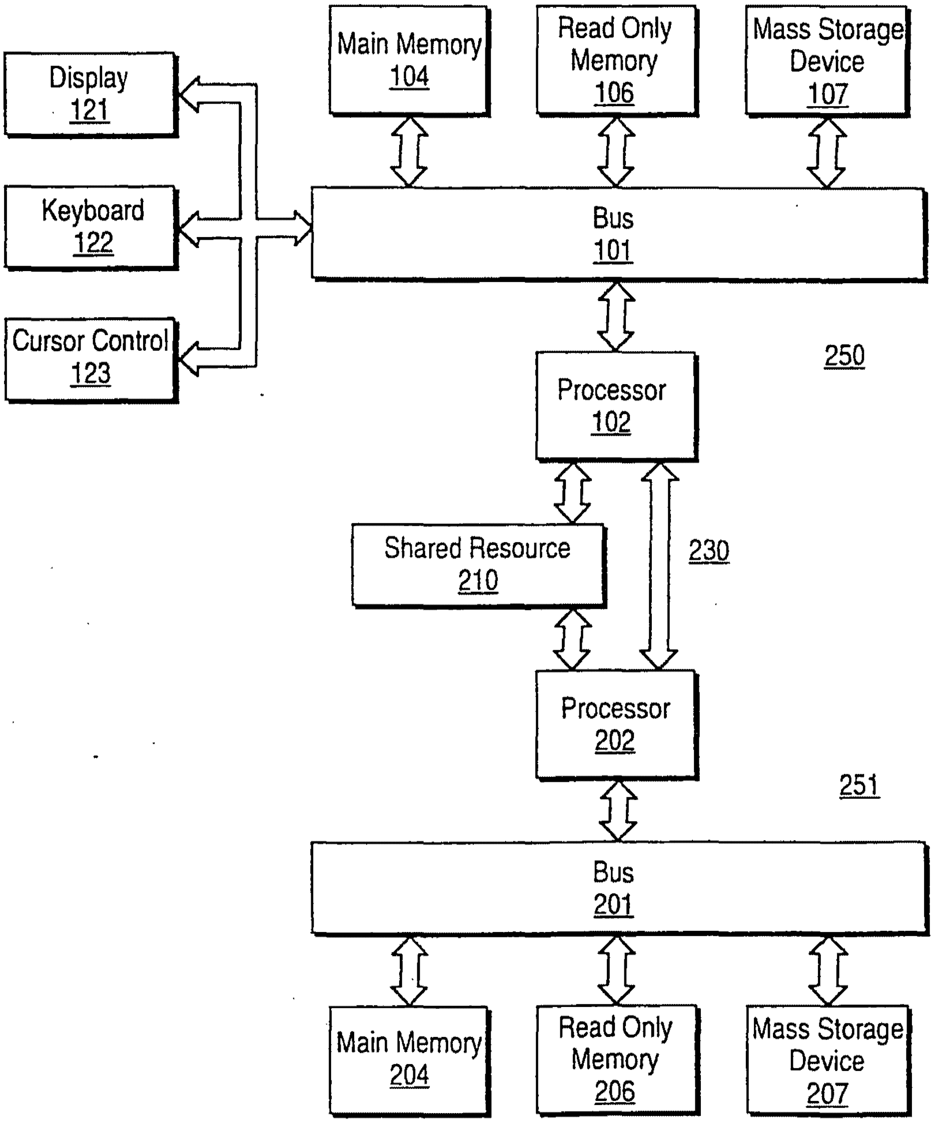

Display
121
Keyboard
122
Cursor Control
123
Main Memory
104
Read Only Memory
106
Mass Storage Device
107
Bus
101
250
Processor
102
Shared Resource
210
230
Processor
202
251
Bus
201
Main Memory
204
Read Only Memory
206
Mass Storage Device
207

FIG. 3

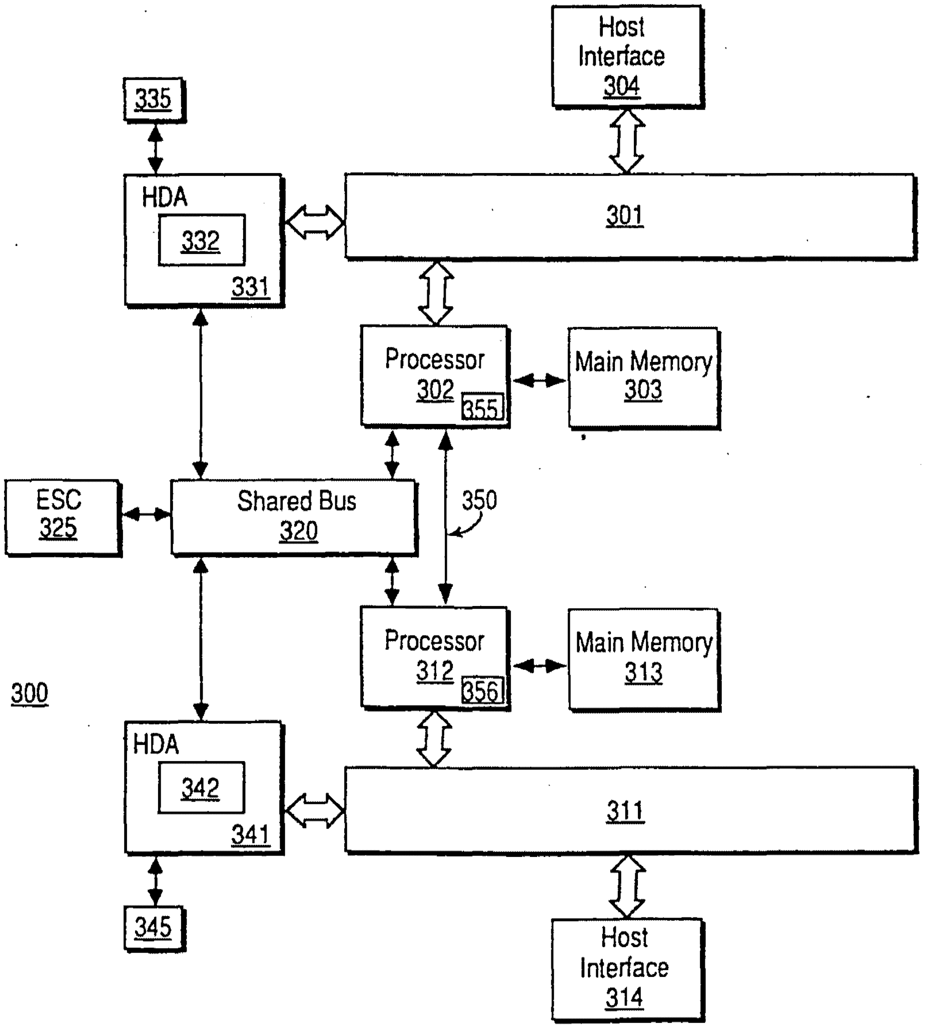

Host Interface 304
335
HDA
332
331
301
Processor 302
355
Main Memory 303
ESC 325
Shared Bus 320
350
Processor 312
356
Main Memory 313
300
HDA
342
341
311
345
Host Interface 314

FIG. 4

State	Device 1	Device 2	Mastership
1	0	0	Device 1 is master
2	1	0	Mastership is to be passed from Device 1 to Device 2
3	1	1	Device 2 is master
4	0	1	Mastership is to be passed from Device 2 to Device 1

FIG. 5

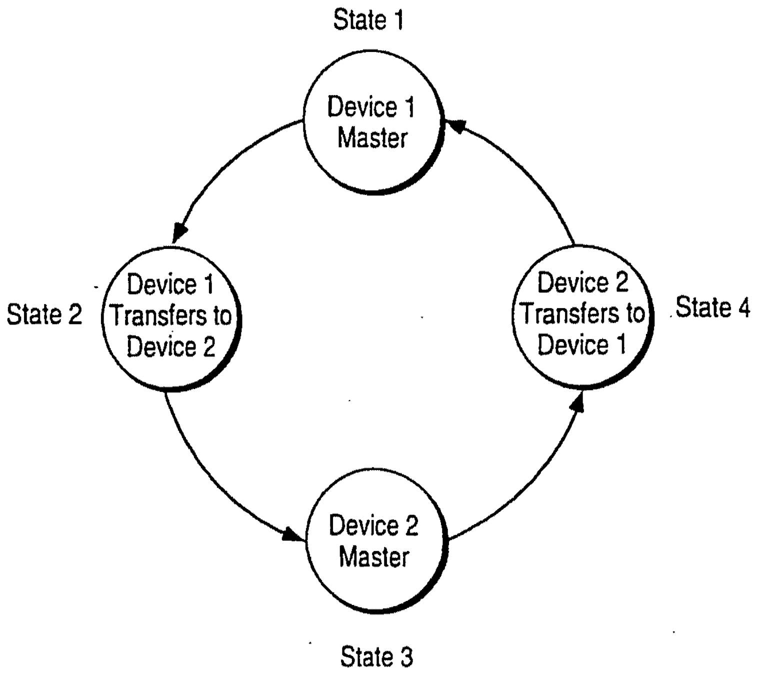

State 1
Device 1 Master
State 2
Device 1 Transfers to Device 2
Device 2 Transfers to Device 1
State 4
Device 2 Master
State 3

FIG. 6

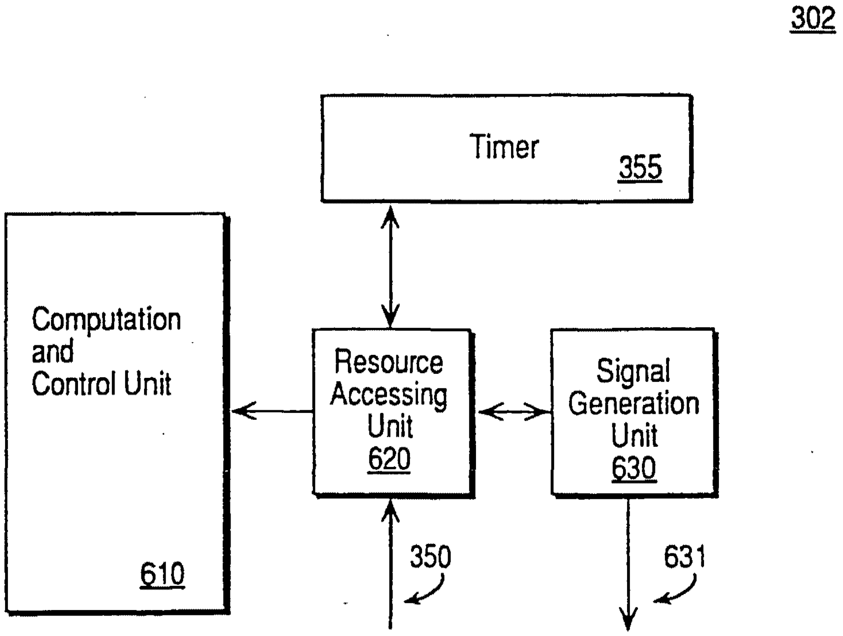
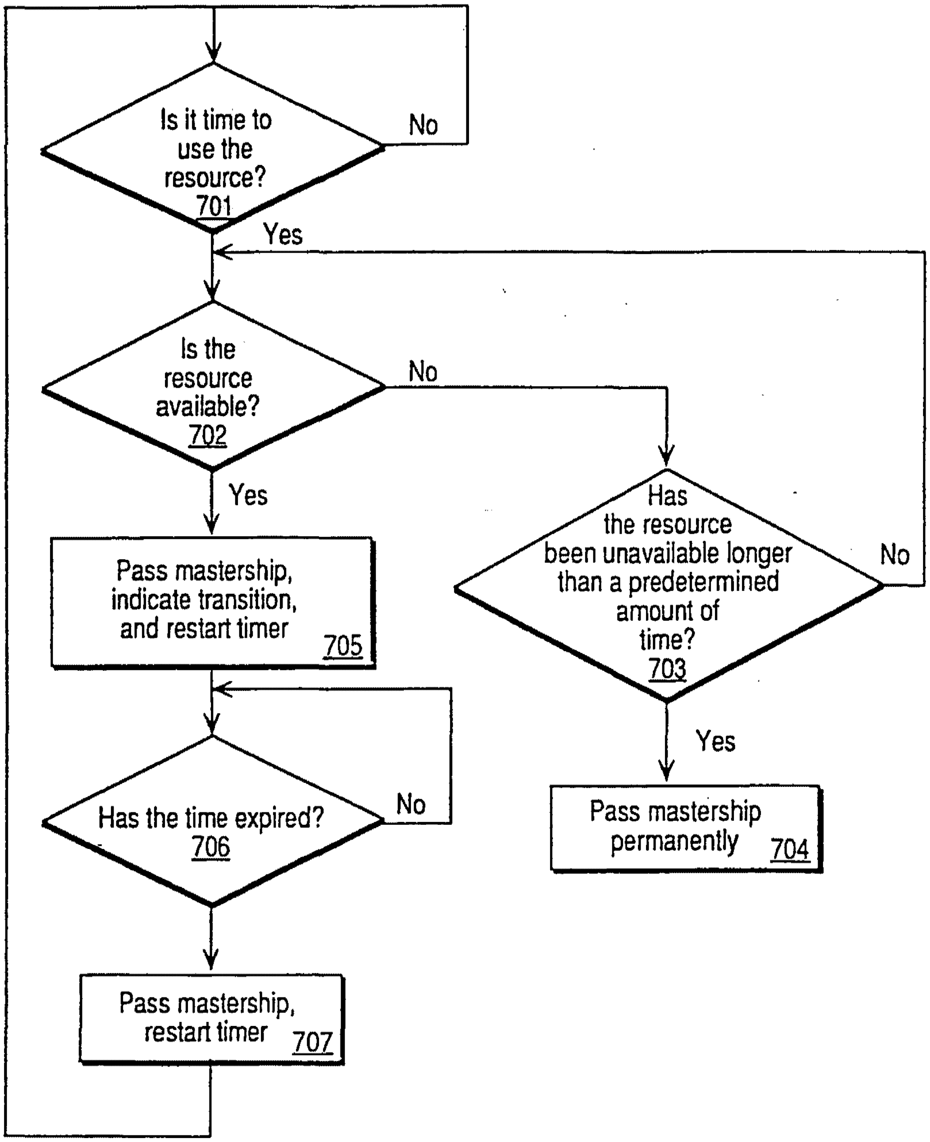

302
Timer
355
Computation
and
Control Unit
610
Resource
Accessing
Unit
620
Signal
Generation
Unit
630
350
631

FIG. 7


Is it time to use the resource?
701
No
Yes
Is the resource available?
702
No
Yes
Has the resource been unavailable longer than a predetermined amount of time?
703
No
Yes
Pass mastership, indicate transition, and restart timer
705
Has the time expired?
706
No
Pass mastership, restart timer
707
Pass mastership permanently
704

ß5

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 827 059 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
04.03.1998 Bulletin 1998/10

(51) Int. Cl.^{6}: G06F 1/00, G06F 3/06

(21) Application number: 97114612.1

(22) Date of filing: 22.08.1997

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV RO SI

(30) Priority: 30.08.1996 JP 230895/96

(71) Applicant: NEC Corporation
Minato-ku, Tokyo 108-01 (JP)

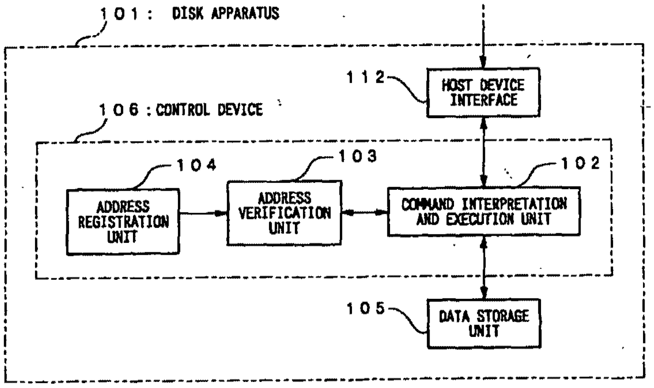
(72) Inventors:
- Kikuchi, Yoshihide
Minato-ku, Tokyo 108-01 (JP)
- Akagi, Masanobu
Minato-ku, Tokyo 108-01 (JP)

(74) Representative:
von Samson-Himmelstjerna, Friedrich R., Dipl.-Phys.
SAMSON & PARTNER
Widenmayerstrasse 5
80538 München (DE)

(54) **Disk apparatus**

(57) The apparatus enables access authorization to be assigned solely to specific host devices. A control device (106) comprises: an address registration unit (104), in which the host address of each host device has been registered for authorizing access, a command interpretation and execution unit (102) which on receipt of a command from a host device via a host device interface outputs the host address of the host device based on the command, and an address verification unit (103) for verifying the host address output from a command interpretation and execution unit (102) against the host address registered in the address registration unit (104), as well as determining whether or not the particular host device has access authorization. The command interpretation and execution unit (102) incorporates an authorization pending function, so that on receipt of a command from a host device, the command is interpreted and executed only after access is authorized by the address verification unit (103).

FIG.1



Printed by Xerox (UK) Business Services
2.15.11/3.4

Description

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to a disk apparatus, and in particular to a disk apparatus which can be accessed by a plurality of host devices.

Description of the Related Art

With conventional disk apparatus, each host controls the disk or disk array directly, and disk security is controlled by the host device to which the disk is connected. File sharing with this type of file server client system is disclosed for example in Japanese Patent Application, First Publication No. Hei-4-58349.

A block diagram showing the configuration of a conventional disk apparatus is shown in Figure 6. A conventional disk apparatus 201 comprises a command interpretation and execution unit 202 which interprets commands from a host device as well as executing those commands, and a data storage unit 203 in which data is stored. The command interpretation and execution unit 202, in the case of a read command for example, interprets the command, and recognizing the command as a read command directs the data storage unit 203 to read. The data storage unit 203 reads the stored data based on the read directions from the command interpretation and execution unit 202, and then transfers the data to the host device.

Common ways of connecting the host device and the disk apparatus include a SCSI (Small Computer System Interface) and Fibre Channel. Consequently, the command interpretation and execution unit 202 interprets commands from the SCSI or Fibre Channel and then outputs commands such as read and/or write, to the disk data storage unit 203.

With this type of conventional disk apparatus, usually a single host device is connected to the disk apparatus. Furthermore, even in those cases where a plurality of host devices are connected to a common disk interface, with current technology it is possible for any of the host devices to access the disk.

With advances in technology relating to the interface between the host device and the disk apparatus however, it has become feasible to connect a plurality of host devices. Using Fibre Channel, it is possible for example to use loops (FC-AL) to connect together more than 100 devices including both host devices and disk apparatus. Moreover, if switching fabric is employed the number of devices which can be connected together increases even further. Utilizing the high speed of interfaces, it is also possible to connect a plurality of host devices and disk apparatus to a single interface. With conventional disk apparatus, a problem arises that in the case where a single disk is able to be accessed by a plurality of hosts devices, access authorization can not be restricted to specific host devices.

Furthermore, with the move to large volume disk apparatus, it is possible to consider partitioning a single disk and then having each host use a different partition, but with conventional disk apparatus it has not been possible, while using a single interface, to identify a host device and then have each host device use a different partition.

SUMMARY OF THE INVENTION

It is an object of the present invention to improve the deficiencies inherent in the conventional devices discussed above, and in particular to provide a disk apparatus in which each host device can be treated differently, so that for example access authorization can be assigned solely to specific host devices, or furthermore, each host device can gain access to a different partition while using the same interface.

A first apparatus according to the present invention comprises: a host device interface for sending and receiving data to and from a plurality of host devices, a data storage device for storing data to be sent to a host device, and a control device for controlling the writing of data to, and the reading of data from, the data storage device.

The control device comprises an address registration unit, in which the host address of each host device has been registered in advance, for the purpose of authorizing access, a command interpretation and execution unit which on receipt of a command from a host device via the host device interface outputs the host address of the host device based on the command, and an address verification unit for verifying the host address output from the command interpretation and execution unit against the host address registered in the address registration unit, and for determining whether or not the particular host device has access authorization. The command interpretation and execution unit is configured to include an authorization pending function, so that on receipt of a command from a host device, the command is interpreted and executed only after access is authorized by the address verification unit.

With this first apparatus, the host address is extracted from the command sent from a host device and verified against those host addresses registered in the address registration unit for the purpose of determining access authorization. As a result, if access is authorized, the disk apparatus accepts the command which has been sent and disk read/write functions are performed. In this way, only authorized host devices gain access to the data storage unit.

As a second apparatus according to the present invention a construction is adopted where, in addition to the items which characterize the first apparatus, a host information storage unit in which information about the hosts such as host names and passwords is stored, is incorporated into the address registration unit, and a host check unit which, on receipt of host information from a host, determines whether or not that particular host has access authorization based on the host information received from the host and the host information stored in the host information storage unit, is incorporated into the command interpretation and execution unit, and this host check unit incorporates an address registration function which registers the access authorization based on the host information, and the host address determined for the host device, in the address registration unit.

With this second apparatus, when a host device logs in to the disk apparatus seeking authorization to use the disk, the address is registered in the address registration unit, and subsequently, the host address is extracted from any commands sent from the host device and verified against the host address registered in the address registration unit, and in those cases where access is authorized the command interpretation and execution unit transmits the command from the host device to the data storage unit and executes the command. In this way, any alterations in host address can be easily accommodated.

With a third apparatus, a construction is adopted where in addition to the items which characterize the second apparatus, the host check unit incorporates a startup setting function which requests host information from a plurality of host devices when the control device is activated.

With this third apparatus, host information relating to access authorization is not stored internally beforehand, but rather is sent from the host devices which control the disk at the point of disk startup. Consequently, the amount of non volatile memory set aside for data storage can be reduced.

As a fourth apparatus according to the present invention a construction is adopted where, in addition to the items which characterize the first apparatus, the control device comprises: an offset information generation unit, which on the basis of a host address output from the command interpretation and execution unit generates offset information for the disk partition for that particular host device, and an actual partition address generation unit which on the basis of the address for reading and writing to the disk apparatus, and the offset information, generates an actual disk partition address and then outputs that actual partition address to the command interpretation and execution unit.

With this fourth apparatus, the disk capacity is partitioned amongst the various host devices, and the various host addresses and the offset information for each partition are coordinated beforehand. When a command is received from a host device, the command interpretation and execution unit extracts the host address from the command and sends it to the offset information generation unit. The offset information generation unit then uses a correlation chart of host devices and offset information which has been stored in advance, and generates offset information which corresponds to the particular host device and sends this information to the actual partition address generation unit. The actual partition address generation unit combines the theoretical disk address included in the command from the host device and the offset information, and generates an actual disk partition address. In this way, the disk partition corresponding to the host device from which the command was sent is accessed.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram showing the configuration of a first embodiment of the present invention;
Figure 2 is an explanatory diagram displaying a phase transition state of a SCSI bus;
Figure 3 is a block diagram showing an example configuration of hardware resources of a disk apparatus according to the first embodiment shown in Figure 1;
Figure 4 is a block diagram showing the configuration of a second embodiment of the present invention;
Figure 5 is a block diagram showing the configuration of a third embodiment of the present invention; and
Figure 6 is a block diagram showing a configuration based on current technology.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Next is a description of the preferred embodiments of the present invention, with reference to the drawings.

First embodiment

A block diagram showing the configuration of a disk apparatus according to a first embodiment of the present invention is shown in Figure 1. As is shown in Figure 1, a disk apparatus 101 comprises a host device interface 112 for sending and receiving data to and from a plurality of host devices, a data storage device (data storage unit) 105 for storing data to be sent to a host device, and a control device 106 for controlling the writing of data to, and the reading of data from, the data storage device 105.

The control device 106 comprises: an address registration unit 104, in which the host address of each host device has been registered for authorizing access, a command interpretation and execution unit 102 which on receipt of a command from a host device via the host device interface outputs the host address of the host device based on the command, and an address verification unit 103 for verifying the host address output from the command interpretation and execution unit 102 against the host address registered in the address registration unit 104, and for determining whether or not the particular host device has access authorization.

The command interpretation and execution unit 102 incorporates an authorization pending function, so that on receipt of a command from a host device, the command is interpreted and executed only after access is authorized by the address verification unit 103.

The command interpretation and execution unit 102 first receives a command from a host device, extracts the host address from the command and outputs it to the address verification unit 103. The address verification unit 103 reads the host addresses stored in the address registration unit 104 for the purpose of determining access authorization and verifies the host address sent from the command interpretation and execution unit 102. The access authorization information generated as a result of this verification process is then relayed back to the command interpretation and execution unit 102 by the address verification unit 103.

In those cases where access is authorized, the command interpretation and execution unit 102 sends the command received from the host device to the data storage unit 105, and the disk apparatus command, such as a data read/write command, is carried out in the same manner as for conventional disks.

The technique for determining access authorization could for example involve the registration of the host addresses of those host devices for which access is authorized in the address registration unit 104 and comparison of these address with the host address extracted from each command, with authorization being given in the case of a matching address. Alternatively, the host addresses of those host devices for which access is not authorized could be registered in the address registration unit 104, and authorization given if the host address extracted from the command did not match any of the registered addresses.

With the above example it was assumed that the host address was imbedded in the command, but in practice, the host address can sometimes be identified in exchanges prior to, or after the command. An example is presented in way of an explanation below.

For example in the case of a SCSI, the bus phase can be roughly divided up as shown in Figure 2. With a SCSI generally the host device interface is the initiator and the disk apparatus interface the target. When sending a command to the disk apparatus, the host device interface, the initiator, secures the bus in the arbitration phase, selects the disk apparatus in the selection phase, and then enters the information transfer phase for sending the command or data.

Within this series of phases, the initiator outputs its own ID and the ID of the target it is aiming to select in the selection phase. The specified disk apparatus, namely the target, on confirming it has been selected corresponds by switching the bus BSY signal to "true". At this point, the target samples the data bus and identifies the ID of the initiator.

In this way, the disk apparatus is able to ascertain the SCSI ID, namely the host address, of the other device. Further details are given in "Open design No. 1" (Published by CQ, 1994), pages 4 to 19.

In the case of a Fibre Channel, because communication is serial, the host address is recorded within the frame and so once again the disk apparatus is able to ascertain the host address of the other device.

Furthermore nowadays, in addition to those mentioned above, there are other protocols (such as IP (Internet Protocol)) which although not widely used as disk interfaces, do include a host address which becomes the transmission source.

An example configuration of the above embodiment which uses a general purpose CPU (central processing unit) is shown in Figure 3. A disk apparatus 101 comprises a CPU 106 which performs the centralized function of controlling reading and writing. The CPU 106 is connected to various circuit devices via a bus 107. Of these devices, a ROM (read only memory) 108 is memory solely for reading, and stores various programs and fixed data.

A RAM (random access memory) 109 is memory which is used, as required, for temporarily storing data during execution of a program.

A non volatile memory 110 is memory which can be written to by the CPU, and the content of which is saved when the power is turned off. A disk interface 111 is an interface for exchanging data and commands between the CPU and a data storage unit 105 which will be either a disk or some other storage medium.

A host device interface 112 is an interface for exchanging commands and data from a host device with the disk apparatus 101. In the case of a disk array, a SCSI is used for both the host device interface 112 and for the disk interface 111, but generally it is acceptable for the host device interface 112 and the disk interface 111 to be of different types.

For example, a Fibre Channel could be used for the host device interface 112 and a SCSI used for the disk interface 111. In small apparatus the disk storage medium itself is used as the data storage unit 105, but in large apparatus such as disk arrays the disk drive itself can be used as the data storage unit 105.

Next is a description of the use of the hardware resources shown in Figure 3 to bring to realization the function blocks of Figure 1. The command interpretation and execution unit 102 of Figure 1 is configured using the CPU 106, the bus 107, the ROM 108, the RAM 109, the disk interface 111 and the host device interface 112 of Figure 3. Similarly, the address verification unit 103 is configured using the CPU 106, the bus 107, the ROM 108, and the RAM 109.

The address registration unit 104 can be configured using the non volatile memory 110. Moreover, a read/write capable disk drive can be used as the data storage unit 105. In those instances where a disk drive with a SCSI interface is used as the data storage unit, the commands which can be sent from the command interpretation and execution unit 102 to the data storage unit 105 are not limited to just read and write commands for data, but can also indicate commands in general retained by the SCSI interface. Furthermore, the disk drive can comprise any form which allows data storage, and can therefore be configured from memory with a power backup function or from non volatile memory.

Next is a description of the operation of a disk apparatus configured as shown in Figure 3. First, host addresses are stored in advance in the non volatile memory 110. The stored host addresses can be rewritten by the CPU 106, but will not be erased when the power is switched off. Consequently, when power is supplied to the disk apparatus 101, the host addresses which have been previously stored are able to be read out.

The command interpretation and execution unit 102 of Figure 1 receives commands from the host devices at the host device interface 112 and stores them temporarily in the RAM 109. The CPU 106 uses the programs stored in the ROM 108 for interpreting a command from a host device and extracting the host address. The thus extracted host address is then verified against the host addresses stored in the non volatile memory 110 by the CPU 106. In the method where the host addresses for those devices which are authorized for access are stored in the non volatile memory 110, access is authorized when the host address extracted from the command from the host device matches one of the host addresses stored in advance in the non volatile memory.

In those cases where access is authorized, the CPU 106 sends a command to the disk interface 111 in order to execute the command from the host device, which had been temporarily stored in the RAM 109. The disk interface 111 executes the command by sending it to the data storage unit 105. In those cases where information needs to be relayed to the host device as a result of the command being executed, the disk interface informs the CPU 106 that it has received a result.

On receiving this notification the CPU 106 receives the result from the disk interface 111, stores it temporarily in the RAM 109, and then transfers the result to the host device interface. In this way, commands from a host device are first judged as to whether access is possible, and then following execution, any result of the execution is returned to the host device.

With the above example, the host address stored temporarily in the RAM 109 and the access authorization determining host addresses stored in the non volatile memory 110 were compared, but in some cases the reading of non volatile memory is time consuming, and so it is possible to imagine a technique where on startup of the disk apparatus the access authorization determining host addresses stored in the non volatile memory 110 are transferred to the RAM 109.

Furthermore as with the invention of the first apparatus, it is possible to imagine a technique where on startup of the disk apparatus the access authorization determining host addresses are transferred from the host device which controls the disk, and then stored in the RAM 109. With this technique, the amount of non volatile memory 110 can be greatly reduced.

Second embodiment

A block diagram showing the configuration of a disk apparatus according to a second embodiment of the present invention is shown in Figure 4. This is an embodiment which allows the setting of the host address afterwards. This embodiment will be explained in terms of the login operation from a host device to obtain authorization for using the disk apparatus, and the normal access operation.

First, in the login operation, the host information sent from a host device is used to determine whether that particular host device should be authorized. A disk apparatus 113 of this embodiment comprises a command interpretation and execution unit 114 for interpreting and executing commands from host devices. The command interpretation and execution unit 114 receives a command from a host device and extracts the necessary host information required to authorize usage of the disk apparatus as well as the host address accompanying that host information, and sends it all to a host check unit 115.

In the host check unit 115, this information is verified against access authorization determining host information which has been stored in advance in a host information storage unit 116. Examples of host information include the host device name, and a password. In those cases where the comparison results in a match, the host address sent from the command interpretation and execution unit 114 is registered in an address registration unit 118 as an access authorization determining address.

Once the host address has been registered in the address registration unit 118 in this way, the remaining operation is the same as for the first embodiment. Upon receiving a command from a host device the command interpretation and execution unit 114 extracts the host address from the command. It then sends this address to an address verification unit 117 and the address verification unit 117 verifies the address against the access authorization determining host addresses stored in the address registration unit 118 and then relays an access authorized or access denied message back to the command interpretation and execution unit 114. In the case where access is authorized, the command interpretation and execution unit 114 sends a command to the data storage unit 105 in order to execute the command.

With the second embodiment, the actual circuit configuration could take the form shown in Figure 3, as was the case with the first embodiment. The command interpretation and execution unit 114 of Figure 4 could then be configured comprising the CPU 106, the bus 107, the ROM 108, the RAM 109, the disk interface 111, and the host device interface 112 of Figure 3. Similarly, the host check unit 115 and the address verification unit 117 can be configured comprising the CPU 106, the bus 107, the ROM 108, and the RAM 109. Furthermore, the host information unit 116 and the address registration unit 104 can be configured using the non volatile memory 110.

Third embodiment

A block diagram showing the configuration of a disk apparatus according to a third embodiment of the present invention is shown in Figure 5. A disk apparatus 119 of this embodiment comprises a command interpretation and execution unit 120 for interpreting and executing commands from a host device. The command interpretation and execution unit 120 extracts a host address from any disk read/write command sent from a host device and outputs it to an address offset information conversion unit 121, and also outputs a disk partition address extracted from the read/write command to an actual partition address conversion unit 122.

The technique used by the command interpretation and execution unit 120 for extracting a host address is as was outlined for the first embodiment. The host address output from the command interpretation and execution unit 120 is input into the address offset information conversion unit 121. Offset information which indicates a disk partition corresponding to each host device, has been stored in advance in the address offset information conversion unit 121, and the host address input from the command interpretation and execution unit 120 is converted to this offset information.

The actual partition address conversion unit 122 combines the disk partition address output from the command interpretation and execution unit 120 with the offset information output from the address offset information conversion unit 121, and generates an actual disk partition address which it then outputs to the command interpretation and execution unit 120. The command interpretation and execution unit 120 outputs a read/write command to the data storage unit 105 based on the actual disk partition address. The data storage unit 105 executes the command output from the actual partition address conversion unit 122 by, for example, reading out data to the host device, or receiving and storing data from the host device.

The present invention is configured and functions in the manner outlined above, with the invention of the first apparatus enabling the provision of a highly secure and advanced disk apparatus of a type not currently available, wherein determination of access authorization for a host device is based on the host address imbedded in the command sent from that particular host device, thus enabling commands to be accepted only from specified host devices.

With the invention of the second apparatus, the information registered in advance in the disk apparatus by the user is not host addresses, but rather host information. Each host address is registered prior to that host device using the disk apparatus, so that once registered, subsequent recognition of the host device can be based on the host address imbedded in normal commands. Therefore procedures can be vastly simplified in comparison with the technique where host information is exchanged each time the disk apparatus is accessed. Furthermore, because the information registered in advance in the disk apparatus does not include host addresses, even if the interface configuration or address is changed there is little effect, allowing high security to be maintained.

With the invention of the third apparatus, following disk startup the host addresses relating to access authorization are received from the host device which controls the disk apparatus, and stored internally. This offers the advantage that complicated programming relating to host address registration does not need to be provided on the disk.

With the invention of the fourth apparatus, the disk apparatus is able to identify a host device from the host address imbedded within the command sent from the host device. Moreover because a partition offset information value is stored for each host device, the disk apparatus is able to allocate a different disk partition to each host device. Consequently, a single disk apparatus can essentially appear as a different disk to each host device, enabling the efficient usage of modern large volume disk apparatus.

Claims

1. A disk apparatus comprising, a host device interface (112) for sending and receiving data to and from a plurality of host devices, data storage means (105) for storing data to be sent to said host devices, and control means (106) for controlling the writing of data to, and the reading of data from, said data storage means (105), characterized in that said control device (106) comprises: an address registration unit (104; 118), in which the host address of each host device has been registered in advance, for the purpose of authorizing access, a command interpretation and execution unit (102; 114; 120) which on receipt of a command from a host device via said host device interface (112) outputs the host address of said host device based on said command, and an address verification unit (103) for verifying the host address output from said command interpretation and execution unit (102; 114) against the host address registered in said address registration unit (104; 118), and for determining whether or not the particular host device has access authorization, and said command interpre-

tation and execution unit (102; 114; 120) incorporates an authorization pending function, so that on receipt of a command from a host device, the command is interpreted and executed only after access is authorized by said address verification unit (103).

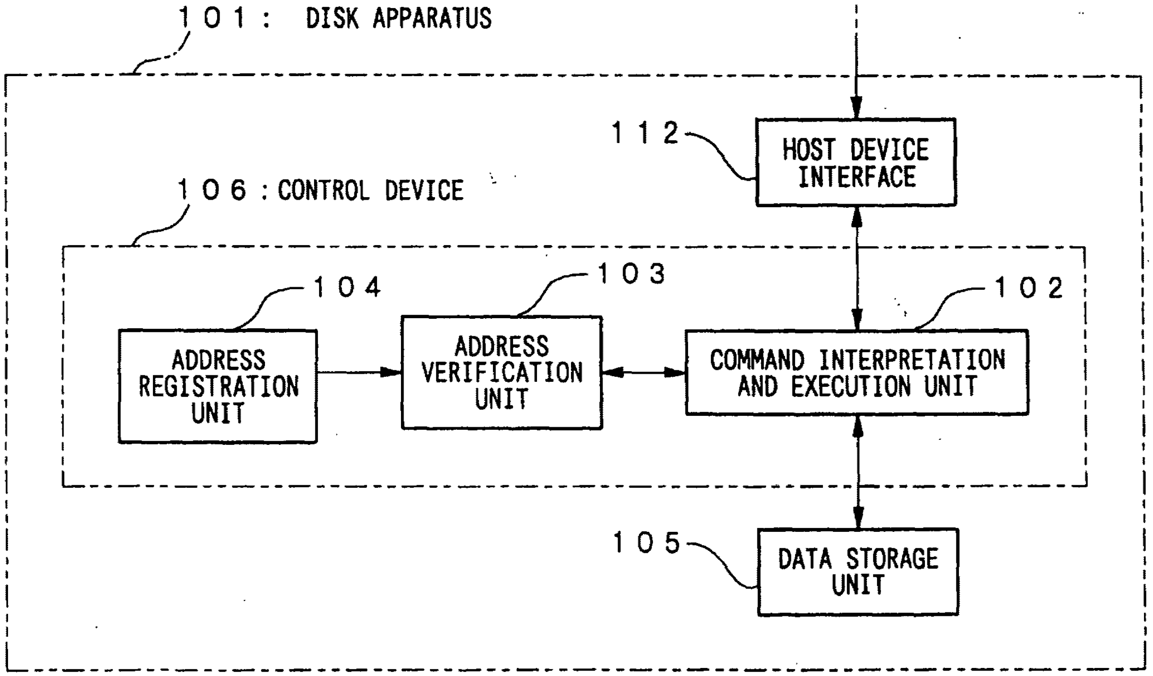
2. A disk apparatus according to claim 1, wherein a host information storage unit (116) in which information about the hosts such as host names and passwords is stored, is incorporated into said address registration unit (104; 118), and a host check unit (115) which, on receipt of host information from a host, determines whether or not that particular host has access authorization based on the host information received from the host and the host information stored in said host information storage unit (116), is incorporated into said command interpretation and execution unit (102; 114; 120), and said host check unit (115) incorporates an address registration function which registers the access authorization based on the host information, and the host address determined for the host device, in said address registration unit (104; 118).

3. A disk apparatus according to claim 2, wherein said host check unit (115) incorporates a startup setting function which requests host information from a plurality of host devices when said control means (106) is activated.

4. A disk apparatus according to claim 2, wherein said control means (106) comprises: an offset information generation unit (121), which on the basis of a host address output from said command interpretation and execution unit (102; 114; 120) generates offset information for the disk partition for that particular host device, and an actual partition address generation unit (122) which on the basis of the address for reading and writing to the disk apparatus, and the offset information, generates an actual disk partition address and then outputs that actual partition address to said command interpretation and execution unit (102; 114; 120).

5. A disk apparatus according to claim 1, wherein said command interpretation and execution unit (102; 114; 120) extracts said host address from said command received from said host device.

FIG.1



101 : DISK APPARATUS
112
HOST DEVICE INTERFACE
106 : CONTROL DEVICE
104
103
102
ADDRESS REGISTRATION UNIT
ADDRESS VERIFICATION UNIT
COMMAND INTERPRETATION AND EXECUTION UNIT
105
DATA STORAGE UNIT

FIG.2

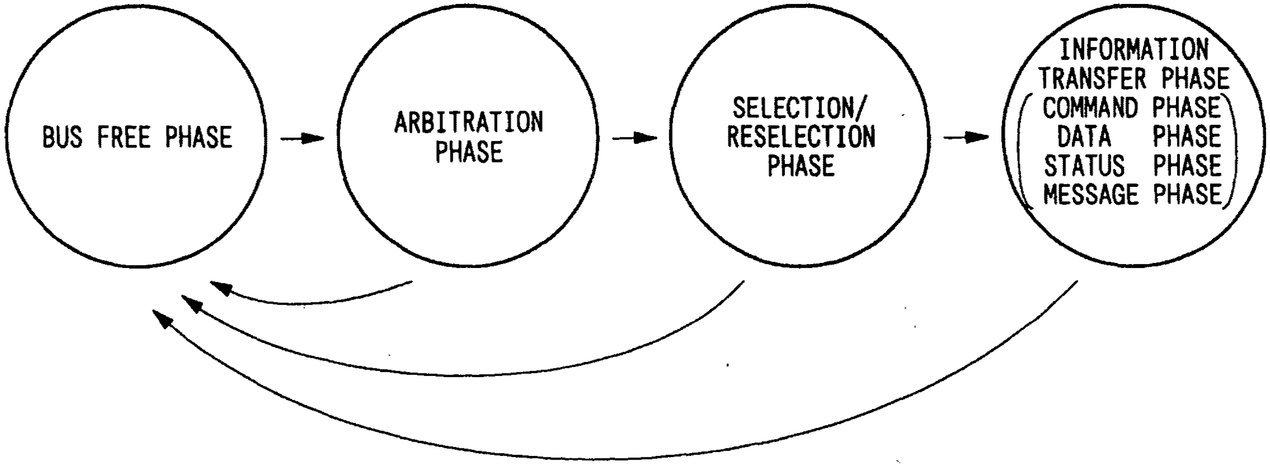

BUS FREE PHASE
ARBITRATION PHASE
SELECTION/ RESELECTION PHASE
INFORMATION TRANSFER PHASE
COMMAND PHASE
DATA PHASE
STATUS PHASE
MESSAGE PHASE

FIG.3

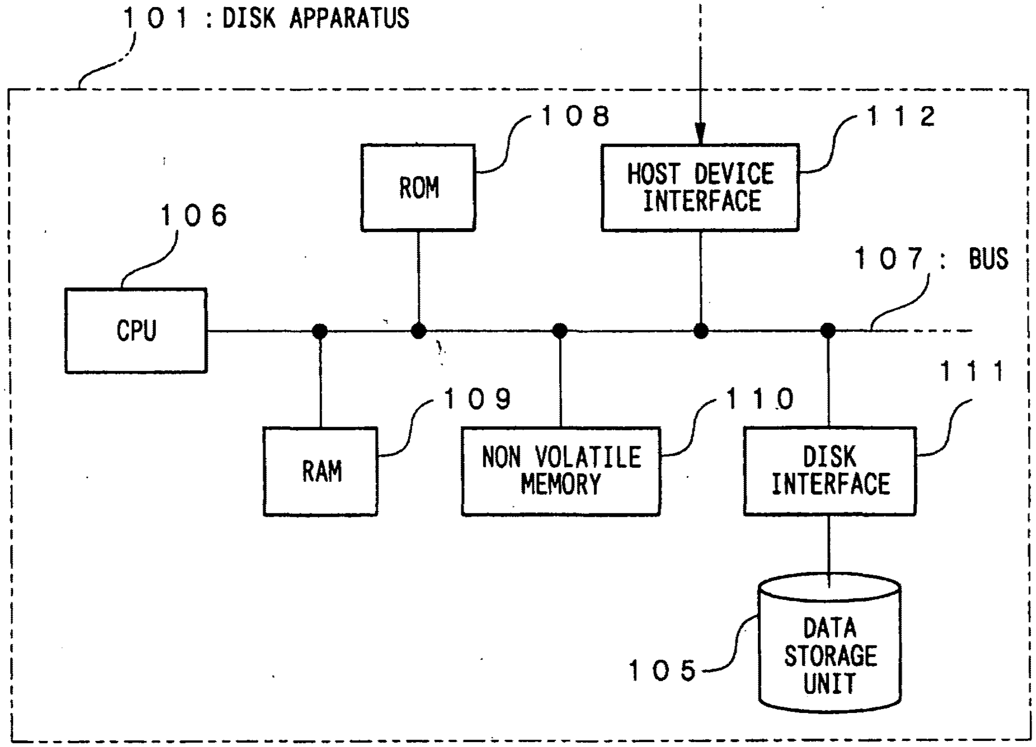
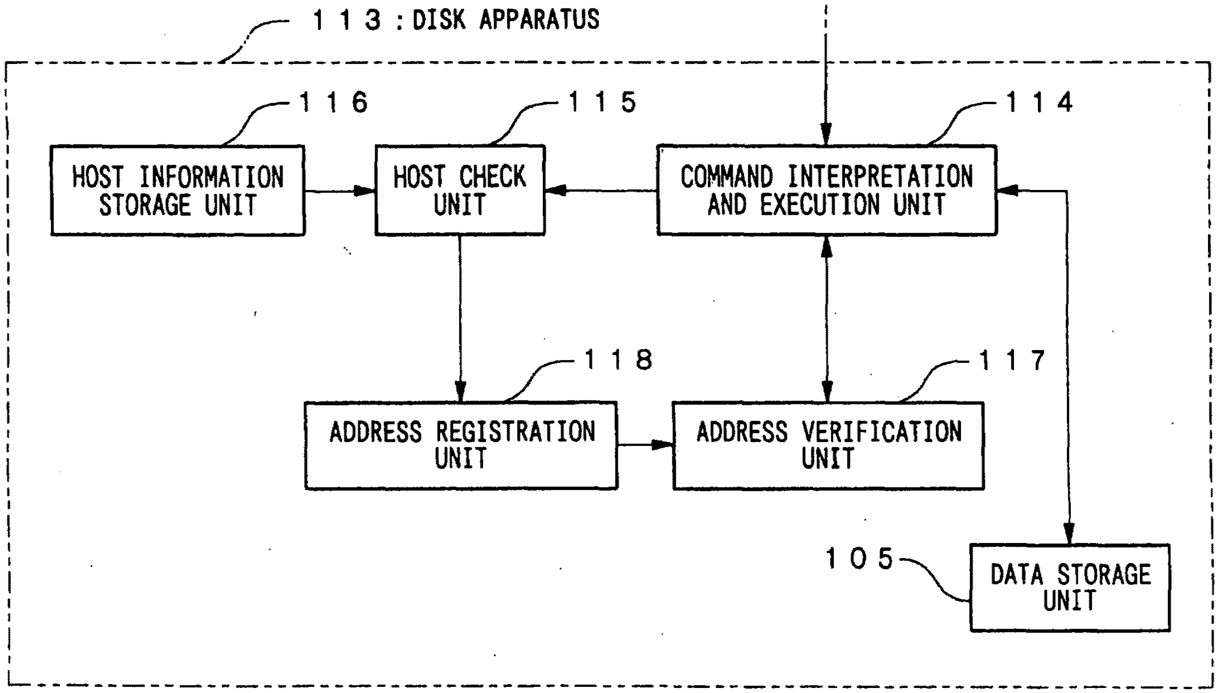

101 : DISK APPARATUS
108
ROM
112
HOST DEVICE INTERFACE
106
CPU
107 : BUS
109
RAM
110
NON VOLATILE MEMORY
111
DISK INTERFACE
105
DATA STORAGE UNIT

FIG.4



113 : DISK APPARATUS
116
115
114
HOST INFORMATION STORAGE UNIT
HOST CHECK UNIT
COMMAND INTERPRETATION AND EXECUTION UNIT
118
117
ADDRESS REGISTRATION UNIT
ADDRESS VERIFICATION UNIT
105
DATA STORAGE UNIT

FIG.5

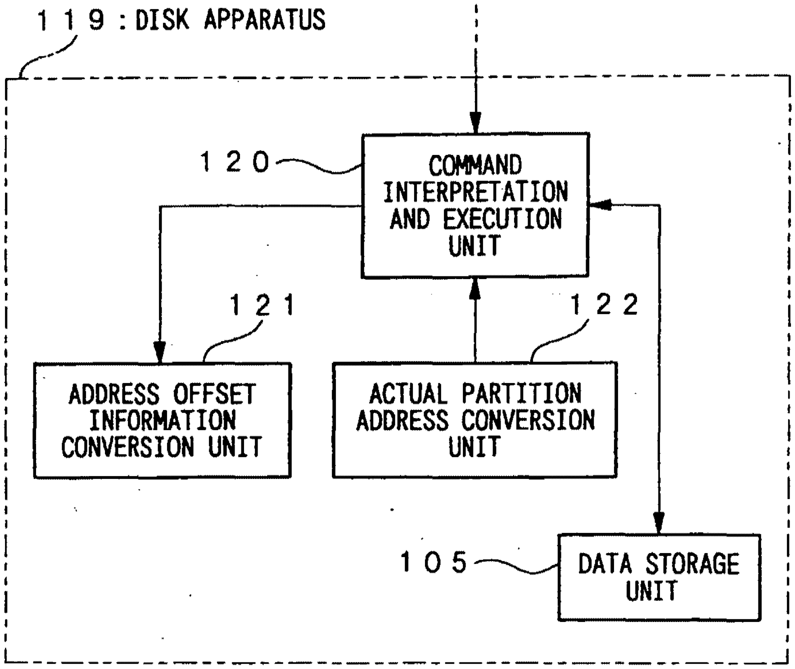
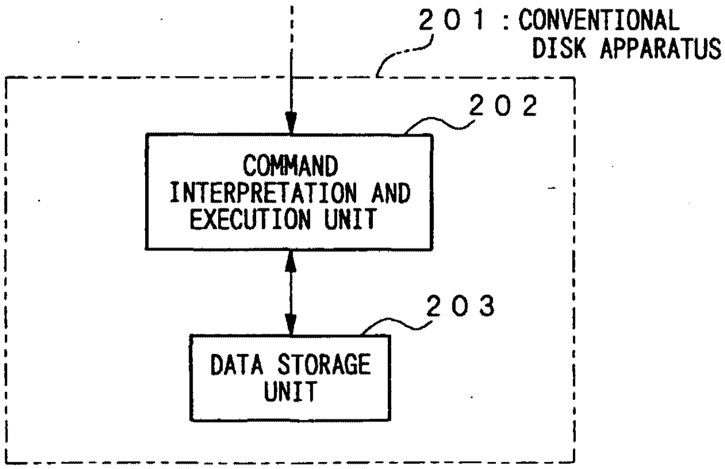

1 1 9 : DISK APPARATUS
1 2 0
COMMAND INTERPRETATION AND EXECUTION UNIT
1 2 1
ADDRESS OFFSET INFORMATION CONVERSION UNIT
1 2 2
ACTUAL PARTITION ADDRESS CONVERSION UNIT
1 0 5
DATA STORAGE UNIT

FIG.6


2 0 1 : CONVENTIONAL DISK APPARATUS
2 0 2
COMMAND INTERPRETATION AND EXECUTION UNIT
2 0 3
DATA STORAGE UNIT

86

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 13/00

A1

(11) International Publication Number: WO 99/34297

(43) International Publication Date: 8 July 1999 (08.07.99)

(21) International Application Number: PCT/US98/27689

(22) International Filing Date: 28 December 1998 (28.12.98)

(30) Priority Data:
09/001,799 31 December 1997 (31.12.97) US

(71) Applicant: CROSSROADS SYSTEMS, INC. [US/US]; Suite II-300, 9390 Research Boulevard, Austin, TX 78759 (US).

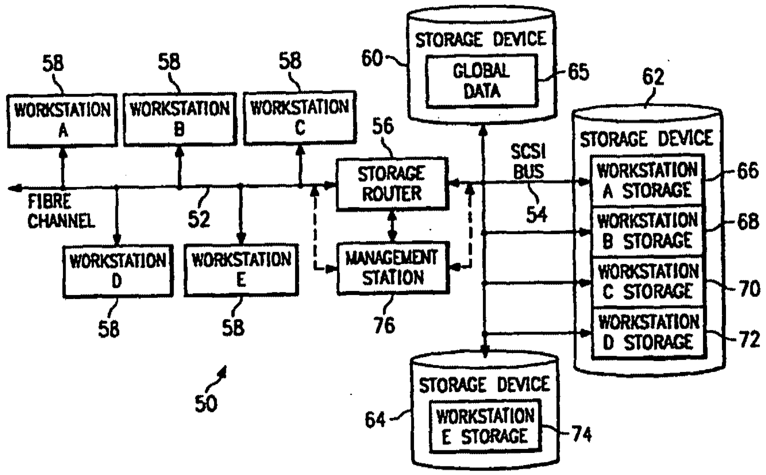
(72) Inventors: HOESE, Geoffrey, B.; 1904 Ann Arbor Avenue, Austin, TX 78704 (US). RUSSELL, Jeffry, T.; 205 Kariba Cove, Cibolo, TX 78108 (US).

(74) Agent: HULSEY, William, R., III; Gray Cary Ware & Freidenrich LLP, Suite 1440, 100 Congress Avenue, Austin, TX 78701 (US).

(81) Designated States: CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE



(57) Abstract

A storage router (56) and storage network (50) provide virtual local storage on remote SCSI storage devices (60, 62, 64) to Fibre Channel devices. A plurality of Fibre Channel devices, such as workstations (58), are connected to a Fibre Channel transport medium (52), and a plurality of SCSI storage devices (60, 62, 64) are connected to a SCSI bus transport medium (54). The storage router (56) interfaces between the Fibre Channel transport medium (52) and the SCSI bus transport medium (54). The storage router (56) maps between the workstations (58) and the SCSI storage devices (60, 62, 64) and implements access controls for storage space on the SCSI storage devices (60, 62, 64). The storage router (56) then allows access from the workstations (58) to the SCSI storage devices (60, 62, 64) using native low level, block protocol in accordance with the mapping and the access controls.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav	TM	Turkmenistan
BF	Burkina Faso	GR	Greece		Republic of Macedonia	TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's	NZ	New Zealand		
CM	Cameroon		Republic of Korea	PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

STORAGE ROUTER AND METHOD FOR PROVIDING VIRTUAL LOCAL STORAGE

TECHNICAL FIELD OF THE INVENTION

This invention relates in general to network storage devices, and more particularly to a storage router and method for providing virtual local storage on remote SCSI storage devices to Fibre Channel devices

BACKGROUND OF THE INVENTION

Typical storage transport mediums provide for a relatively small number of devices to be attached over relatively short distances. One such transport medium is a Small Computer System Interface (SCSI) protocol, the structure and operation of which is generally well known as is described, for example, in the SCSI-1, SCSI-2 and SCSI-3 specifications. High speed serial interconnects provide enhanced capability to attach a large number of high speed devices to a common storage transport medium over large distances. One such serial interconnect is Fibre Channel, the structure and operation of which is described, for example, in *Fibre Channel Physical and Signaling Interface (FC-PH)*, ANSI X3.230 *Fibre Channel Arbitrated Loop (FC-AL)*, and ANSI X3.272 *Fibre Channel Private Loop Direct Attach (FC-PLDA)*.

Conventional computing devices, such as computer workstations, generally access storage locally or through network interconnects. Local storage typically consists of a disk drive, tape drive, CD-ROM drive or other storage device contained within, or locally connected to the workstation. The workstation provides a file system structure, that includes security controls, with access to the local storage device through native low level, block protocols. These protocols map directly to the mechanisms used by the storage device and consist of data requests without security controls. Network interconnects typically provide access for a large number of computing devices to data storage on a remote network server. The remote network server provides file system structure, access control, and other miscellaneous capabilities that include the network interface. Access to data through the network server is through network protocols that the server must translate into low level requests to the storage device. A workstation with access to the server storage must translate its file system protocols into network protocols that are used to communicate with the server. Consequently, from the perspective of a workstation, or other computing device, seeking to access such server data, the access is much slower than access to data on a local storage device.

SUMMARY OF THE INVENTION

In accordance with the present invention, a storage router and method for providing virtual local storage on remote SCSI storage devices to Fibre Channel devices are disclosed that provide advantages over conventional network storage devices and methods.

According to one aspect of the present invention, a storage router and storage network provide virtual local storage on remote SCSI storage devices to Fibre Channel devices. A plurality of Fibre Channel devices, such as workstations, are connected to a Fibre Channel transport medium, and a plurality of SCSI storage devices are connected to a SCSI bus transport medium. The storage router interfaces between the Fibre Channel transport medium and the SCSI bus transport medium. The storage router maps between the workstations and the SCSI storage devices and implements access controls for storage space on the SCSI storage devices. The storage router then allows access from the workstations to the SCSI storage devices using native low level, block protocol in accordance with the mapping and the access controls.

According to another aspect of the present invention, virtual local storage on remote SCSI storage devices is provided to Fibre Channel devices. A Fibre Channel transport medium and a SCSI bus transport medium are interfaced with. A configuration is maintained for SCSI storage devices connected to the SCSI bus transport medium. The configuration maps between Fibre Channel devices and the SCSI storage devices and implements access controls for storage space on the SCSI storage devices. Access is then allowed from Fibre Channel initiator devices to SCSI storage devices using native low level, block protocol in accordance with the configuration.

A technical advantage of the present invention is the ability to centralize local storage for networked workstations without any cost of speed or overhead. Each workstation access its virtual local storage as if it work locally connected. Further, the centralized storage devices can be located in a significantly remote position even in excess of ten kilometers as defined by Fibre Channel standards.

Another technical advantage of the present invention is the ability to centrally control and administer storage space for connected users without limiting the speed with which the users can access local data. In addition, global access to data, backups, virus scanning and redundancy can be more easily accomplished by centrally located storage devices.

A further technical advantage of the present invention is providing support for SCSI storage devices as local storage for Fibre Channel hosts. In addition, the present invention helps to provide extended capabilities for Fibre Channel and for management of storage subsystems.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the present invention and the advantages thereof may be acquired by referring to the following description taken in conjunction with the accompanying drawings, in which like reference numbers indicate like features, and wherein:

FIGURE 1 is a block diagram of a conventional network that provides storage through a network server;

FIGURE 2 is a block diagram of one embodiment of a storage network with a storage router that provides global access and routing;

FIGURE 3 is a block diagram of one embodiment of a storage network with a storage router that provides virtual local storage;

FIGURE 4 is a block diagram of one embodiment of the storage router of FIGURE 3; and

FIGURE 5 is a block diagram of one embodiment of data flow within the storage router of FIGURE 4.

DETAILED DESCRIPTION OF THE INVENTION

FIGURE 1 is a block diagram of a conventional network, indicated generally at 10, that provides access to storage through a network server. As shown, network 10 includes a plurality of workstations 12 interconnected with a network server 14 via a network transport medium 16. Each workstation 12 can generally comprise a processor, memory, input/output devices, storage devices and a network adapter as well as other common computer components. Network server 14 uses a SCSI bus 18 as a storage transport medium to interconnect with a plurality of storage devices 20 (tape drives, disk drives, etc.). In the embodiment of FIGURE 1, network transport medium 16 is an network connection and storage devices 20 comprise hard disk drives, although there are numerous alternate transport mediums and storage devices.

In network 10, each workstation 12 has access to its local storage device as well as network access to data on storage devices 20. The access to a local storage device is typically through native low level, block protocols. On the other hand, access by a workstation 12 to storage devices 20 requires the participation of network server 14 which implements a file system and transfers data to workstations 12 only through high level file system protocols. Only network server 14 communicates with storage devices 20 via native low level, block protocols. Consequently, the network access by workstations 12 through network server 14 is slow with respect to their access to local storage. In network 10, it can Also be a logistical problem to centrally manage and administer local data distributed across an organization, including accomplishing tasks such as backups, virus scanning and redundancy.

FIGURE 2 is a block diagram of one embodiment of a storage network, indicated generally at 30, with a storage router that provides global access and routing. This environment is significantly different from that of FIGURE 1 in that there is no network server involved. In FIGURE 2, a Fibre Channel high speed serial transport 32 interconnects a plurality of workstations 36 and storage devices 38. A SCSI bus storage transport medium interconnects workstations 40 and storage devices 42. A storage router 44 then serves to interconnect these mediums and provide devices on either medium global, transparent access to devices on the other medium. Storage router 44 routes requests from initiator devices on one medium to target devices on the other medium and routes data between the target and the initiator. Storage router 44 can allow initiators and targets to be on either side. In this manner, storage router 44 enhances the functionality of Fibre Channel 32 by providing access, for example, to legacy SCSI storage devices on SCSI bus 34. In the embodiment of FIGURE 2, the operation of storage router 44 can be managed by a management station 46 connected to the storage router via a direct serial connection.

In storage network 30, any workstation 36 or workstation 40 can access any storage device 38 or storage device 42 through native low level, block protocols, and vice versa. This functionality is enabled by storage router 44 which routes requests and data as a generic transport between Fibre Channel 32 and SCSI bus 34. Storage router 44 uses tables to map devices from one medium to the other and distributes requests and data across Fibre Channel 32 and SCSI bus 34 without any security access controls. Although this extension of the high speed serial interconnect provided by Fibre Channel 32 is beneficial, it is desirable to provide security controls in addition to extended access to storage devices through a native low level, block protocol.

FIGURE 3 is a block diagram of one embodiment of a storage network, indicated generally at 50, with a storage router that provides virtual local storage. Similar to that of FIGURE 2, storage network 50 includes a Fibre Channel high speed serial interconnect 52 and a SCSI bus 54 bridged by a storage router 56. Storage router 56 of FIGURE 3 provides for a large number of workstations 58 to be interconnected on a common storage transport and to access common storage devices 60, 62 and 64 through native low level, block protocols.

According to the present invention, storage router 56 has enhanced functionality to implement security controls and routing such that each workstation 58 can have access to a specific subset of the overall data stored in storage devices 60, 62 and 64. This specific subset of data has the appearance and characteristics of local storage and is referred to herein as virtual local storage. Storage router 56 allows the configuration and modification of the storage allocated to each attached workstation 58 through the use of mapping tables or other mapping techniques.

As shown in FIGURE 3, for example, storage device 60 can be configured to provide global data 65 which can be accessed by all workstations 58. Storage device 62 can be configured to provide partitioned subsets 66, 68, 70 and 72, where each partition is allocated to one of the workstations 58 (workstations A, B, C and D). These subsets 66, 68, 70 and 72 can only be accessed by the associated workstation 58 and appear to the associated workstation 58 as local storage accessed using native low level, block protocols. Similarly, storage device 64 can be allocated as storage for the remaining workstation 58 (workstation E).

Storage router 56 combines access control with routing such that each workstation 58 has controlled access to only the specified partition of storage device 62 which forms virtual local storage for the workstation 58. This access control allows security control for the specified data partitions. Storage router 56 allows this allocation of storage devices 60, 62 and 64 to be managed by a management station 76. Management station 76 can connect directly to storage router 56 via a direct connection or, alternately, can interface with storage router 56 through either Fibre Channel 52 or SCSI bus 54. In the latter case, management station 76 can be a workstation or other computing device with special rights such that storage router 56 allows access to mapping tables and shows storage devices 60, 62 and 64 as they exist physically rather than as they have been allocated.

The environment of FIGURE 3 extends the concept of a single workstation having locally connected storage devices to a storage network 50 in which workstations 58 are provided virtual local storage in a manner transparent to workstations 58. Storage router 56 provides centralized control of what each workstation 58 sees as its local drive, as well as what data it sees as global data accessible by other workstations 58. Consequently, the storage space considered by the workstation 58 to be its local storage is actually a partition (i.e., logical storage definition) of a physically remote storage device 60, 62 or 64 connected through storage router 56. This means that similar requests from workstations 58 for access to their local storage devices produce different accesses to the storage space on storage devices 60, 62 and 64. Further, no access from a workstation 58 is allowed to the virtual local storage of another workstation 58.

The collective storage provided by storage devices 60, 62 and 64 can have blocks allocated by programming means within storage router 56. To accomplish this function, storage router 56 can include routing tables and security controls that define storage allocation for each workstation 58. The advantages provided by implementing virtual local storage in centralized storage devices include the ability to do collective backups and other collective administrative functions more easily. This is accomplished without limiting the performance of workstations 58 because storage access involves native low level, block protocols and does not involve the overhead of high level protocols and file systems required by network servers.

FIGURE 4 is a block diagram of one embodiment of storage router 56 of FIGURE 3. Storage router 56 can comprise a Fibre Channel controller 80 that interfaces with Fibre Channel 52 and a SCSI controller 82 that interfaces with SCSI bus 54. A buffer 84 provides memory work space and is connected to both Fibre Channel controller 80 and to SCSI controller 82. A supervisor unit 86 is connected to Fibre Channel controller 80, SCSI controller 82 and buffer 84. Supervisor unit 86 comprises a microprocessor for controlling operation of storage router 56 and to handle mapping and security access for requests between Fibre Channel 52 and SCSI bus 54.

FIGURE 5 is a block diagram of one embodiment of data flow within storage router 56 of FIGURE 4. As shown, data from Fibre Channel 52 is processed by a Fibre Channel (FC) protocol unit 88 and placed in a FIFO queue

90. A direct memory access (DMA) interface 92 then takes data out of FIFO queue 90 and places it in buffer 84. Supervisor unit 86 processes the data in buffer 84 as represented by supervisor processing 93. This processing involves mapping between Fibre Channel 52 and SCSI bus 54 and applying access controls and routing functions. A DMA interface 94 then pulls data from buffer 84 and places it into a buffer 96. A SCSI protocol unit 98 pulls data from buffer 96 and communicates the data on SCSI bus 54. Data flow in the reverse direction, from SCSI bus 54 to Fibre Channel 52, is accomplished in a reverse manner.

The storage router of the present invention is a bridge device that connects a Fibre Channel link directly to a SCSI bus and enables the exchange of SCSI command set information between application clients on SCSI bus devices and the Fibre Channel links. Further, the storage router applies access controls such that virtual local storage can be established in remote SCSI storage devices for workstations on the Fibre Channel link. In one embodiment, the storage router provides a connection for Fibre Channel links running the SCSI Fibre Channel Protocol (FCP) to legacy SCSI devices attached to a SCSI bus. The Fibre Channel topology is typically an Arbitrated Loop (FC_AL).

In part, the storage router enables a migration path to Fibre Channel based, serial SCSI networks by providing connectivity for legacy SCSI bus devices. The storage router can be attached to a Fibre Channel Arbitrated Loop and a SCSI bus to support a number of SCSI devices. Using configuration settings, the storage router can make the SCSI bus devices available on the Fibre Channel network as FCP logical units. Once the configuration is defined, operation of the storage router is transparent to application clients. In this manner, the storage router can form an integral part of the migration to new Fibre Channel based networks while providing a means to continue using legacy SCSI devices.

In one implementation (not shown), the storage router can be a rack mount or free standing device with an internal power supply. The storage router can have a Fibre Channel and SCSI port, and a standard, detachable power cord can be used, the FC connector can be a copper DB9 connector, and the SCSI connector can be a 68-pin type. Additional modular jacks can be provided for a serial port and a 802.3 10BaseT port, i.e. twisted pair Ethernet, for management access. The SCSI port of the storage router an support SCSI direct and sequential access target devices and can support SCSI initiators, as well. The Fibre Channel port can interface to SCSI-3 FCP enabled devices and initiators.

To accomplish its functionality, one implementation of the storage router uses: a Fibre Channel interface based on the HEWLETT-PACKARD TACHYON HPFC-5000 controller and a GLM media interface; an Intel 80960RP processor, incorporating independent data and program memory spaces, and associated logic required to implement a stand alone processing system; and a serial port for debug and system configuration. Further, this implementation includes a SCSI interface supporting Fast-20 based on the SYMBIOS 53C8xx series SCSI controllers, and an operating system based upon the WIND RIVERS SYSTEMS VXWORKS or IXWORKS kernel, as determined by design. In addition, the storage router includes software as required to control basic functions of the various elements, and to provide appropriate translations between the FC and SCSI protocols.

The storage router has various modes of operation that are possible between FC and SCSI target and initiator combinations. These modes are: FC Initiator to SCSI Target; SCSI Initiator to FC Target; SCSI Initiator to SCSI Target; and FC Initiator to FC Target. The first two modes can be supported concurrently in a single storage router device are discussed briefly below. The third mode can involve two storage router devices back to back and can serve primarily as a device to extend the physical distance beyond that possible via a direct SCSI connection. The last mode can be used to carry FC protocols encapsulated on other transmission technologies (e.g. ATM, SONET), or to act as a bridge between two FC loops (e.g. as a two port fabric).

The FC Initiator to SCSI Target mode provides for the basic configuration of a server using Fibre Channel to communicate with SCSI targets. This mode requires that a host system have an FC attached device and associated device drivers and software to generate SCSI-3 FCP requests. This system acts as an initiator using the storage router to communicate with SCSI target devices. The SCSI devices supported can include SCSI-2 compliant direct or sequential access (disk or tape) devices. The storage router serves to translate command and status information and transfer data between SCSI-3 FCP and SCSI-2, allowing the use of standard SCSI-2 devices in a Fibre Channel environment.

The SCSI Initiator to FC Target mode provides for the configuration of a server using SCSI-2 to communicate with Fibre Channel targets. This mode requires that a host system has a SCSI-2 interface and driver software to control SCSI-2 target devices. The storage router will connect to the SCSI-2 bus and respond as a target to multiple target IDs. Configuration information is required to identify the target IDs to which the bridge will respond on the SCSI-2 bus. The storage router then translates the SCSI-2 requests to SCSI-3 FCP requests, allowing the use of FC devices with a SCSI host system. This will also allow features such as a tape device acting as an initiator on the SCSI bus to provide full support for this type of SCSI device.

In general, user configuration of the storage router will be needed to support various functional modes of operation. Configuration can be modified, for example, through a serial port or through an Ethernet port via

SNMP (simple network management protocol) or a Telnet session. Specifically, SNMP manageability can be provided via an 802.3 Ethernet interface. This can provide for configuration changes as well as providing statistics and error information. Configuration can also be performed via TELNET or RS-232 interfaces with menu driven command interfaces. Configuration information can be stored in a segment of flash memory and can be retained across resets and power off cycles. Password protection can also be provided.

In the first two modes of operation, addressing information is needed to map from FC addressing to SCSI addressing and vice versa. This can be 'hard' configuration data, due to the need for address information to be maintained across initialization and partial reconfigurations of the Fibre Channel address space. In an arbitrated loop configuration, user configured addresses will be needed for AL_PAs in order to insure that known addresses are provided between loop reconfigurations.

With respect to addressing, FCP and SCSI 2 systems employ different methods of addressing target devices. Additionally, the inclusion of a storage router means that a method of translating device IDs needs to be implemented. In addition, the storage router can respond to commands without passing the commands through to the opposite interface. This can be implemented to allow all generic FCP and SCSI commands to pass through the storage router to address attached devices, but allow for configuration and diagnostics to be performed directly on the storage router through the FC and SCSI interfaces.

Management commands are those intended to be processed by the storage router controller directly. This may include diagnostic, mode, and log commands as well as other vendor-specific commands. These commands can be received and processed by both the FCP and SCSI interfaces, but are not typically bridged to the opposite interface. These commands may also have side effects on the operation of the storage router, and cause other storage router operations to change or terminate.

A primary method of addressing management commands though the FCP and SCSI interfaces can be through peripheral device type addressing. For example, the storage router can respond to all operations addressed to logical unit (LUN) zero as a controller device. Commands that the storage router will support can include INQUIRY as well as vendor-specific management commands. These are to be generally consistent with SCC standard commands.

The SCSI bus is capable of establishing bus connections between targets. These targets may internally address logical units. Thus, the prioritized addressing scheme used by SCSI subsystems can be represented as follows: BUS:TARGET:LOGICAL UNIT. The BUS identification is intrinsic in the configuration, as a SCSI initiator is attached to only one bus. Target addressing is handled by bus arbitration from information provided to the arbitrating device. Target addresses are assigned to SCSI devices directly, though some means of configuration, such as a hardware jumper, switch setting, or device specific software configuration. As such, the SCSI protocol provides only logical unit addressing within the Identify message. Bus and target information is implied by the established connection.

Fibre Channel devices within a fabric are addressed by a unique port identifier. This identifier is assigned to a port during certain well-defined states of the FC protocol. Individual ports are allowed to arbitrate for a known, user defined address. If such an address is not provided, or if arbitration for a particular user address fails, the port is assigned a unique address by the FC protocol. This address is generally not guaranteed to be unique between instances. Various scenarios exist where the AL-PA of a device will change, either after power cycle or loop reconfiguration.

The FC protocol also provides a logical unit address field within command structures to provide addressing to devices internal to a port. The FCP_CMD payload specifies an eight byte LUN field. Subsequent identification of the exchange between devices is provided by the FQXID (Fully Qualified Exchange ID).

FC ports can be required to have specific addresses assigned. Although basic functionality is not dependent on this, changes in the loop configuration could result in disk targets changing identifiers with the potential risk of data corruption or loss. This configuration can be straightforward, and can consist of providing the device a loop-unique ID (AL_PA) in the range of "01h" to "EFh." Storage routers could be shipped with a default value with the assumption that most configurations will be using single storage routers and no other devices requesting the present ID. This would provide a minimum amount of initial configuration to the system administrator. Alternately, storage routers could be defaulted to assume any address so that configurations requiring multiple storage routers on a loop would not require that the administrator assign a unique ID to the additional storage routers.

Address translation is needed where commands are issued in the cases FC Initiator to SCSI Target and SCSI Initiator to FC Target. Target responses are qualified by the FQXID and will retain the translation acquired at the beginning of the exchange. This prevents configuration changes occurring during the course of execution of a command from causing data or state information to be inadvertently misdirected. Configuration can be required in cases of SCSI Initiator to FC Target, as discovery may not effectively allow for FCP targets to consistently be found. This is due to an FC arbitrated loop supporting addressing of a larger number of devices than a SCSI bus and the possibility of FC devices changing their AL-PA due to device insertion or other loop initialization.

In the direct method, the translation to BUS:TARGET:LUN of the SCSI address information will be direct. That is, the values represented in the FCP LUN field will directly map to the values in effect on the SCSI bus. This provides a clean translation and does not require SCSI bus discovery. It also allows devices to be dynamically added to the SCSI bus without modifying the address map. It may not allow for complete discovery by FCP initiator devices, as gaps between device addresses may halt the discovery process. Legacy SCSI device drivers typically halt discovery on a target device at the first unoccupied LUN, and proceed to the next target. This would lead to some devices not being discovered. However, this allows for hot plugged devices and other changes to the loop addressing.

In the ordered method, ordered translation requires that the storage router perform discovery on reset, and collapses the addresses on the SCSI bus to sequential FCP LUN values. Thus, the FCP LUN values 0-N can represent N+1 SCSI devices, regardless of SCSI address values, in the order in which they are isolated during the SCSI discovery process. This would allow the FCP initiator discovery process to identify all mapped SCSI devices without further configuration. This has the limitation that hot-plugged devices will not be identified until the next reset cycle. In this case, the address may also be altered as well.

In addition to addressing, according to the present invention, the storage router provides configuration and access controls that cause certain requests from FC Initiators to be directed to assigned virtual local storage partitioned on SCSI storage devices. For example, the same request for LUN 0 (local storage) by two different FC Initiators can be directed to two separate subsets of storage. The storage router can use tables to map, for each initiator, what storage access is available and what partition is being addressed by a particular request. In this manner, the storage space provided by SCSI storage devices can be allocated to FC initiators to provide virtual local storage as well as to create any other desired configuration for secured access.

Although the present invention has been described in detail, it should be understood that various changes, substitutions, and alterations can be made hereto without departing from the spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A storage router for providing virtual local storage on remote SCSI storage devices to Fibre Channel devices, comprising:

a buffer providing memory work space for the storage router;

a Fibre Channel controller operable to connect to and interface with a Fibre Channel transport medium;

a SCSI controller operable to connect to and interface with a SCSI bus transport medium; and

a supervisor unit coupled to the Fibre Channel controller, the SCSI controller and the buffer, the supervisor unit operable:

to maintain a configuration for SCSI storage devices connected to the SCSI bus transport medium that maps between Fibre Channel devices and SCSI storage devices and that implements access controls for storage space on the SCSI storage devices; and

to process data in the buffer to interface between the Fibre Channel controller and the SCSI controller to allow access from Fibre Channel initiator devices to SCSI storage devices using native low level, block protocol in accordance with the configuration.

2. The storage router of Claim 1, wherein the configuration maintained by the supervisor unit includes an allocation of subsets of storage space to associated Fibre Channel devices, wherein each subset is only accessible by the associated Fibre Channel device.

3. The storage router of Claim 2, wherein the Fibre Channel devices comprise workstations.

4. The storage router of Claim 2, wherein the SCSI storage devices comprise hard disk drives.

5. The storage router of Claim 1, wherein the Fibre Channel controller comprises:

a Fibre Channel (FC) protocol unit operable to connect to the Fibre Channel transport medium;

a first-in-first-out queue coupled to the Fibre Channel protocol unit; and

a direct memory access (DMA) interface coupled to the first-in-first-out queue and to the buffer.

6. The storage router of Claim 1, wherein the SCSI controller comprises:

a SCSI protocol unit operable to connect to the SCSI bus transport medium;

an internal buffer coupled to the SCSI protocol unit; and

a direct memory access (DMA) interface coupled to the internal buffer and to the buffer of the storage router.

7. A storage network, comprising:

a Fibre Channel transport medium;

a SCSI bus transport medium;

a plurality of workstations connected to the Fibre Channel transport medium;

a plurality of SCSI storage devices connected to the SCSI bus transport medium; and

a storage router interfacing between the Fibre Channel transport medium and the SCSI bus transport medium, the storage router providing virtual local storage on the SCSI storage devices to the workstations and operable:

to map between the workstations and the SCSI storage devices;

to implement access controls for storage space on the SCSI storage devices; and

to allow access from the workstations to the SCSI storage devices using native low level, block protocol in accordance with the mapping and access controls.

8. The storage network of Claim 7, wherein the access controls include an allocation of subsets of storage space to associated workstations, wherein each subset is only accessible by the associated workstation.

9. The storage network of Claim 7, wherein the SCSI storage devices comprise hard disk drives.

10. The storage network of Claim 7, wherein the storage router comprises:

a buffer providing memory work space for the storage router;

a Fibre Channel controller operable to connect to and interface with a Fibre Channel transport medium, the Fibre Channel controller further operable to pull outgoing data from the buffer and to place incoming data into the buffer;

a SCSI controller operable to connect to and interface with a SCSI bus transport medium, the SCSI controller further operable to pull outgoing data from the buffer and to place incoming data into the buffer; and

a supervisor unit coupled to the Fibre Channel controller, the SCSI controller and the buffer, the supervisor unit operable:

to maintain a configuration for the SCSI storage devices that maps between Fibre Channel devices and SCSI storage devices and that implements the access controls for storage space on the SCSI storage devices; and

to process data in the buffer to interface between the Fibre Channel controller and the SCSI controller to allow access from workstations to SCSI storage devices in accordance with the configuration.

11. A method for providing virtual local storage on remote SCSI storage devices to Fibre Channel devices, comprising:

interfacing with a Fibre Channel transport medium;

interfacing with a SCSI bus transport medium;

maintaining a configuration for SCSI storage devices connected to the SCSI bus transport medium that maps between Fibre Channel devices and the SCSI storage devices and that implements access controls for storage space on the SCSI storage devices; and

allowing access from Fibre Channel initiator devices to SCSI storage devices using native low level, block protocol in accordance with the configuration.

12. The method of Claim 11, wherein maintaining the configuration includes allocating subsets of storage space to associated Fibre Channel devices, wherein each subset is only accessible by the associated Fibre Channel device.

13. The method of Claim 12, wherein the Fibre Channel devices comprise workstations.

14. The method of Claim 12, wherein the SCSI storage devices comprise hard disk drives.

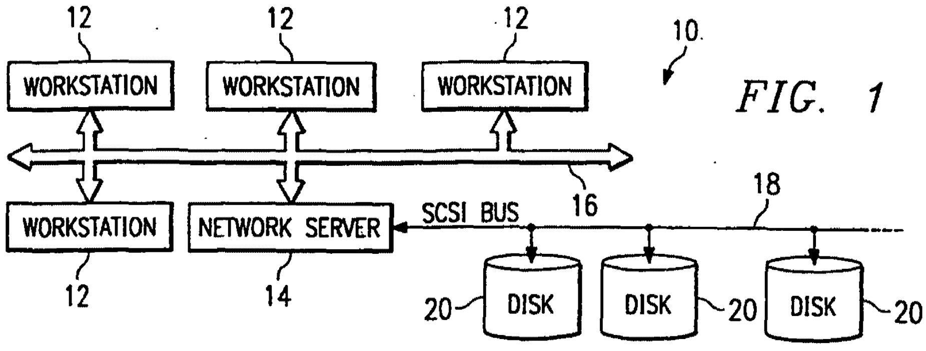


FIG. 1

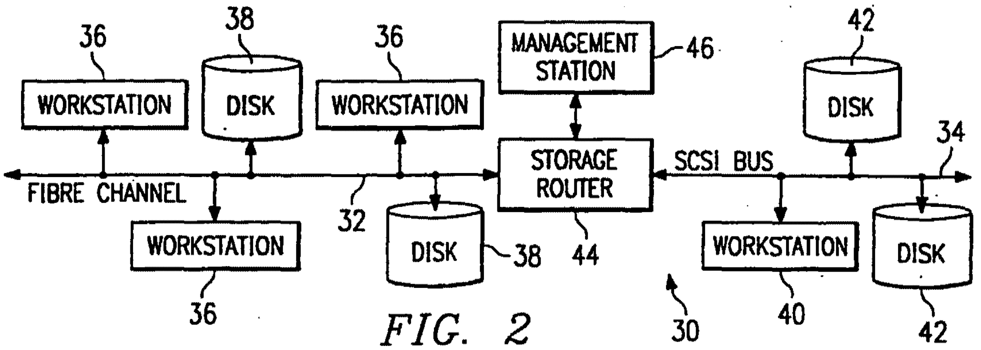


FIG. 2

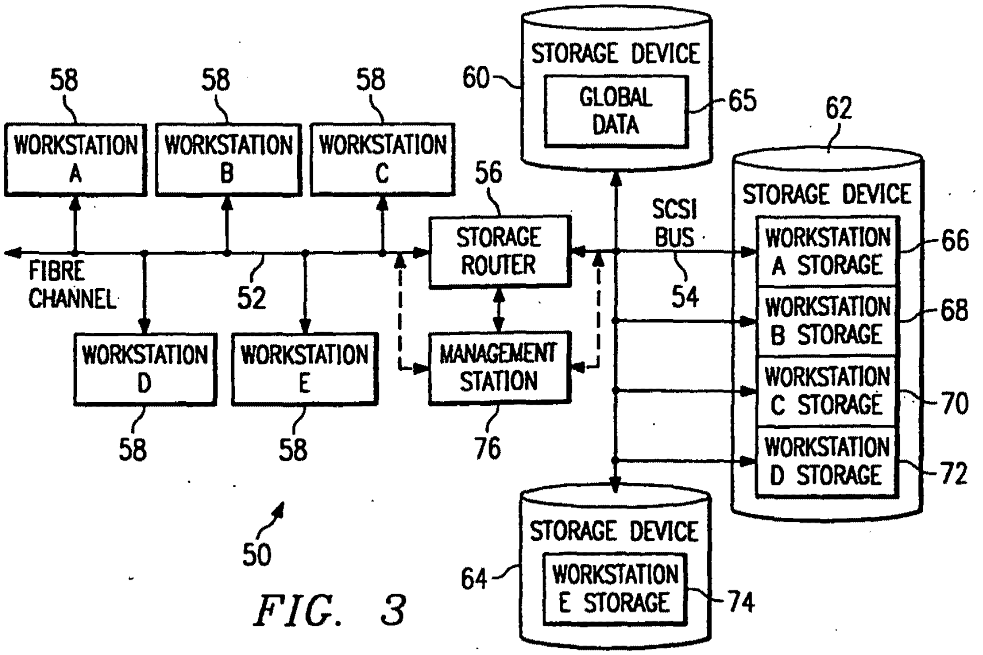


FIG. 3

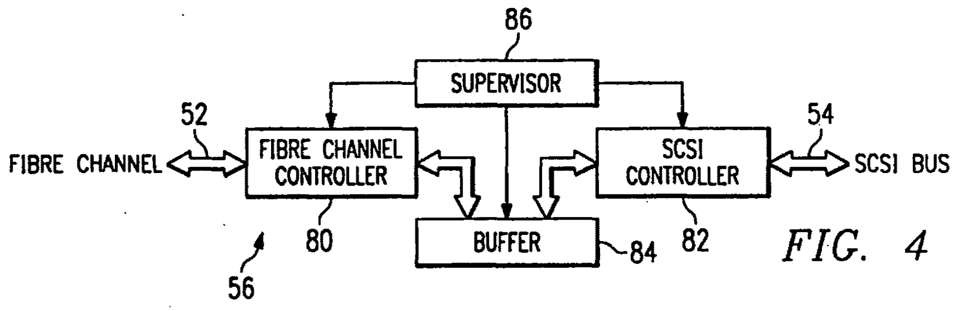

86
SUPERVISOR
52
FIBRE CHANNEL
FIBRE CHANNEL CONTROLLER
SCSI CONTROLLER
54
SCSI BUS
80
BUFFER
84
82
56

FIG. 4

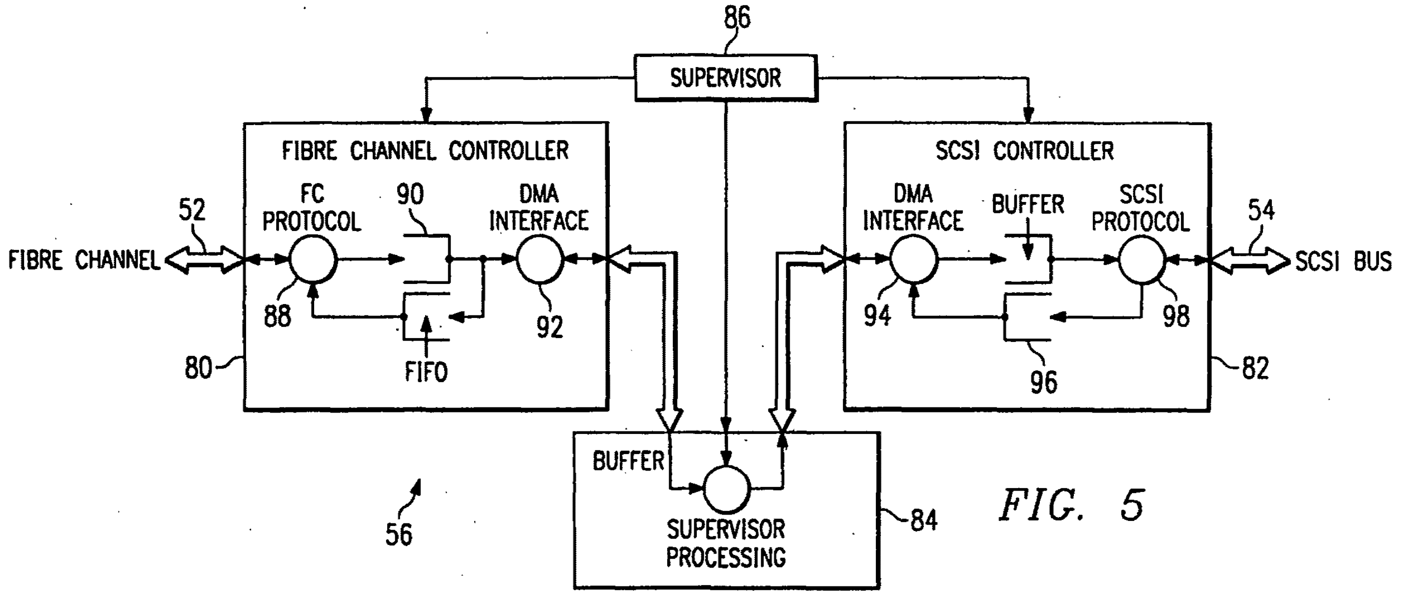

86
SUPERVISOR
FIBRE CHANNEL CONTROLLER
52
FIBRE CHANNEL
FC PROTOCOL
90
DMA INTERFACE
88
92
FIFO
80
SCSI CONTROLLER
DMA INTERFACE
BUFFER
SCSI PROTOCOL
54
SCSI BUS
94
98
96
82
BUFFER
SUPERVISOR PROCESSING
84
56

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27689

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 13/00
US CL :710/129, 128, 2
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 710/129, 128, 2

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

STN, APS, DIALOG

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,748,924 A (LLORENS et al.) 05 May 1998, entire document	1-14
A	US 5,835,496 A (YEUNG et al.) 10 November 1998, entire document	1-14
A	US 5,768,623 A (JUDD et al.) 16 June 1998, entire document	1-14
A	US 5,809,328 A (NOGALES et al.) 15 September 1998, entire document	1-14
A	US 5,812,754 A (LUI et al.) 22 September 1998, entire document	1-14

□ Further documents are listed in the continuation of Box C. □ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 12 MARCH 1999	Date of mailing of the international search report 05 APR 1999
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer CHRISTOPHER SHIN Telephone No. (703) 305-9600

B7

(12) UK Patent Application (19) GB (11) 2 341 715 (13) A

(43) Date of A Publication 22.03.2000

(21) Application No 9820213.8

(22) Date of Filing 17.09.1998

(71) Applicant(s)
Springtek Limited
(Incorporated in the United Kingdom)
Unit 3 Ashbrook Mews, Westbrook Street,
BLEWBURY, Oxon, OX11 9QA, United Kingdom

(72) Inventor(s)
Andrew Paul George Randall

(74) Agent and/or Address for Service
Atkinson Burrington
The Technology Park, 60 Shirland Lane, SHEFFIELD,
S9 3SP, United Kingdom

(51) INT CL^7
G11B 20/18

(52) UK CL (Edition R)
G5R RB33 RGB

(56) Documents Cited
EP 0795812 A1 EP 0717357 A2 EP 0569313 A2
EP 0569236 A2 EP 0485110 A2 EP 0450801 A2
WO 93/18455 A1 WO 91/20076 A1 WO 91/14982 A1
US 5651132 A

(58) Field of Search
UK CL (Edition P) G4A AES , G5R RAC RB33 RGB
INT CL^6 G06F 11/10 , G11B 20/18
EDOC WPI

(54) Abstract Title
Magnetic disk redundant array

(57) A plurality of magnetic disk drives (301, 302, 303) are configured to store machine readable data in a protected way such that data is recoverable in the event of a single disk failure. The array of disks is housed for application directly into an existing disk bay of a computer (101). The array is connectable to the computer as if it were a single conventional computer disk and the drives are controlled by an operating system on the computer as if they were a single storage volume.

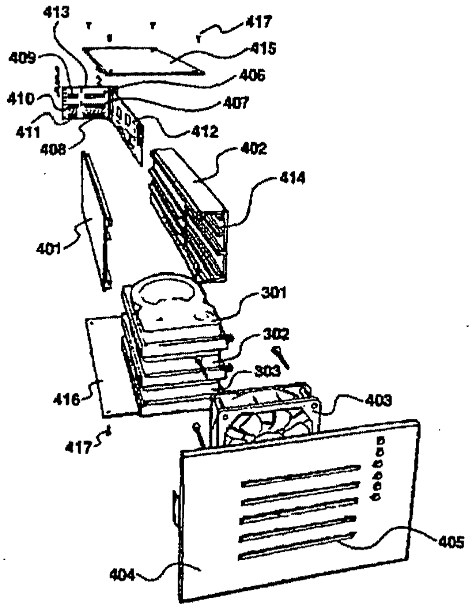


Figure 4

At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

This print takes account of replacement documents submitted after the date of filing to enable the application to comply with the formal requirements of the Patents Rules 1995

GB 2 341 715 A

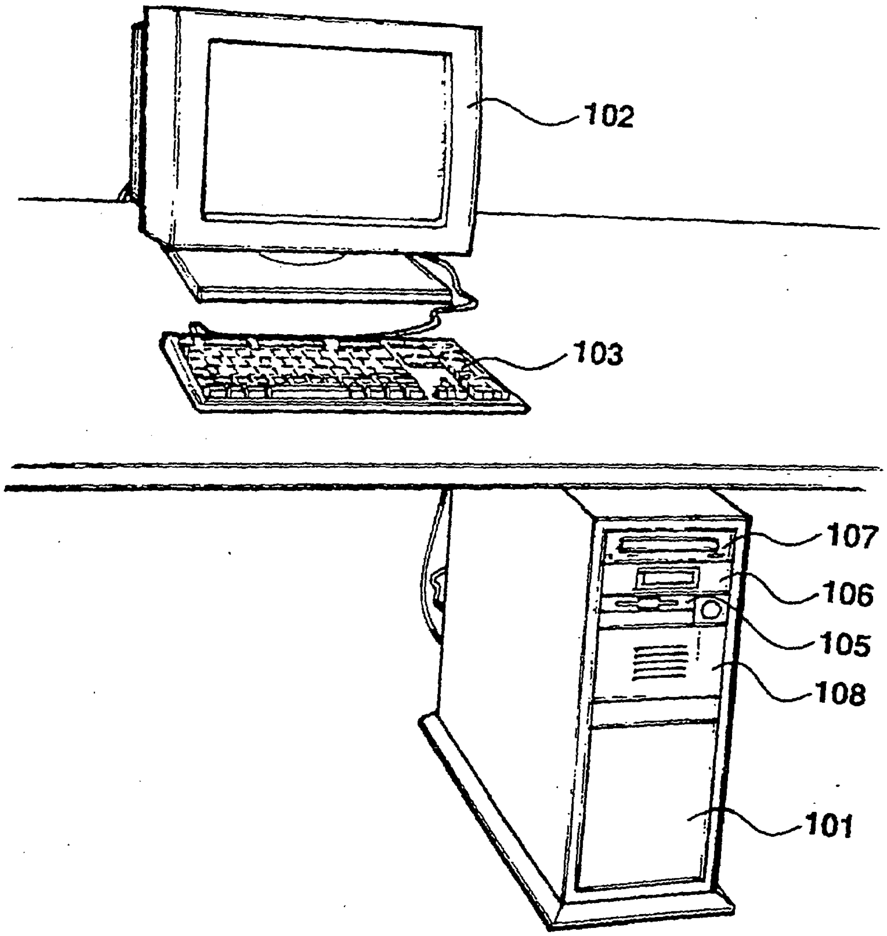


Figure 1

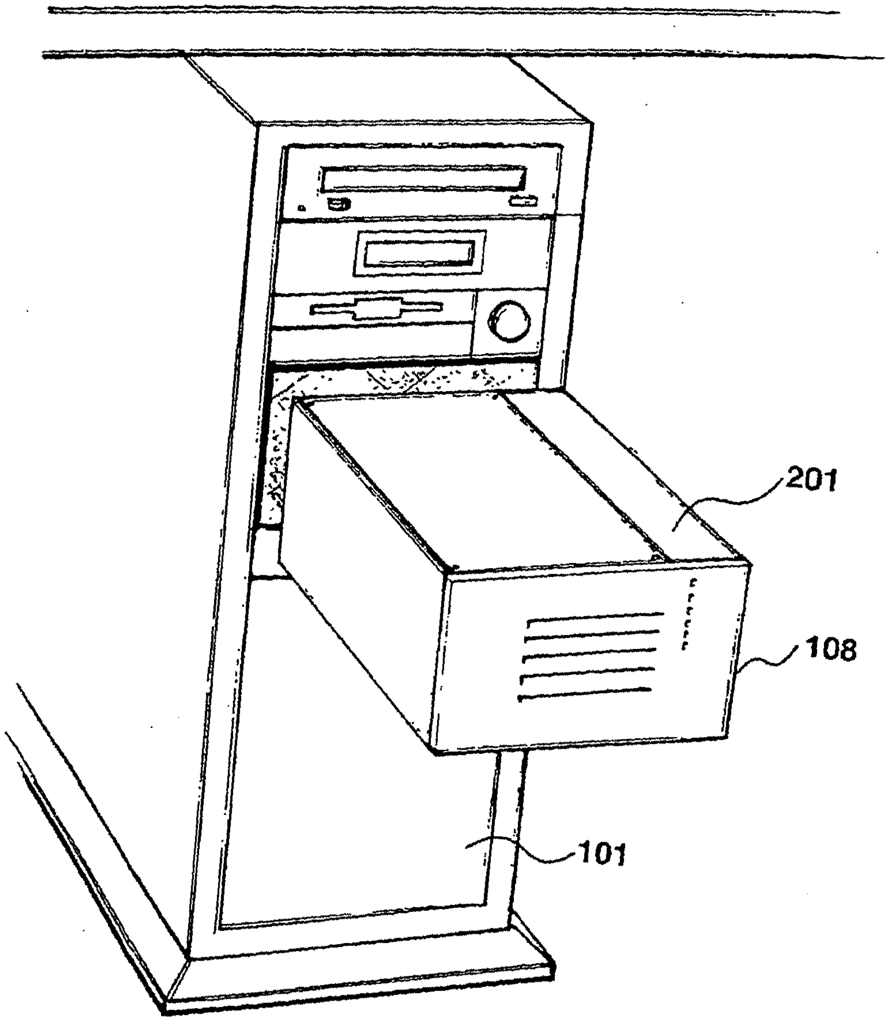


Figure 2

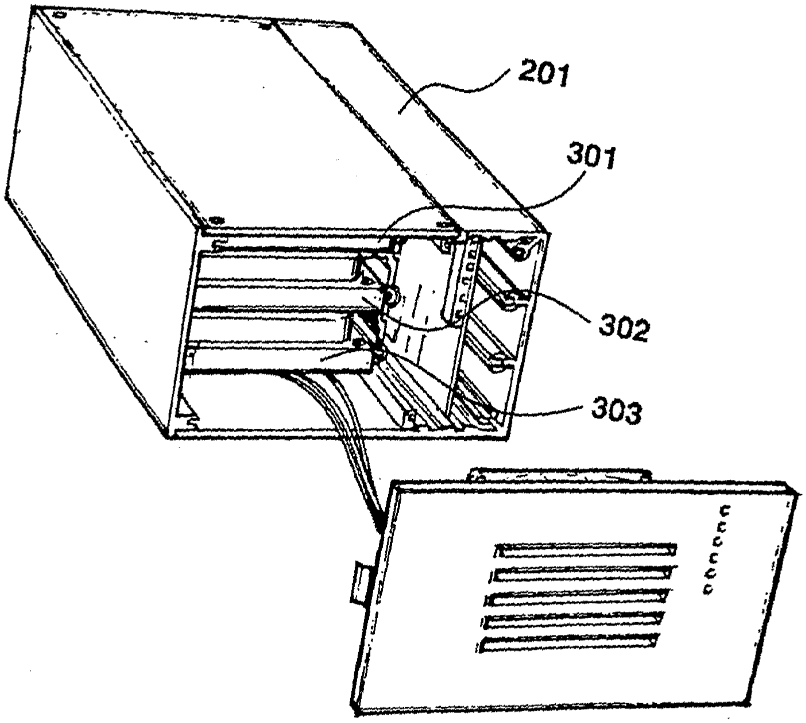


Figure 3

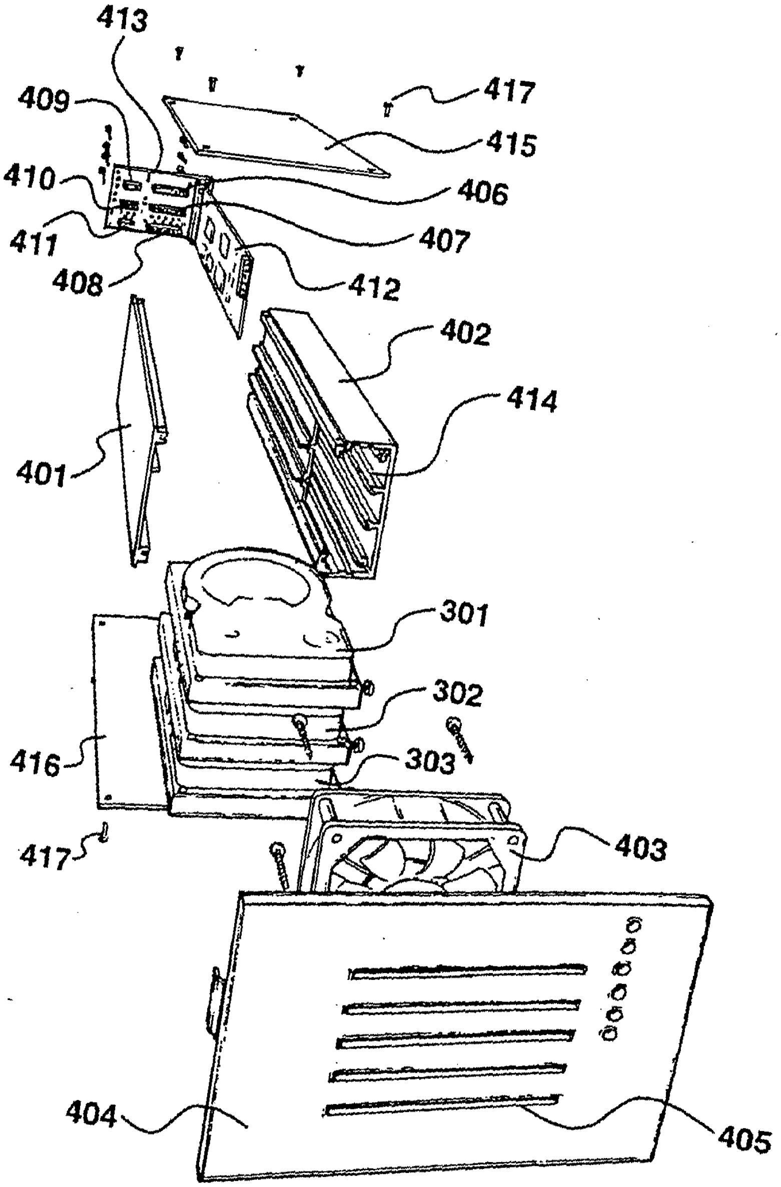

413
409
417
415
410
406
411
407
408
412
402
414
401
301
302
416
303
417
403
404
405

Figure 4

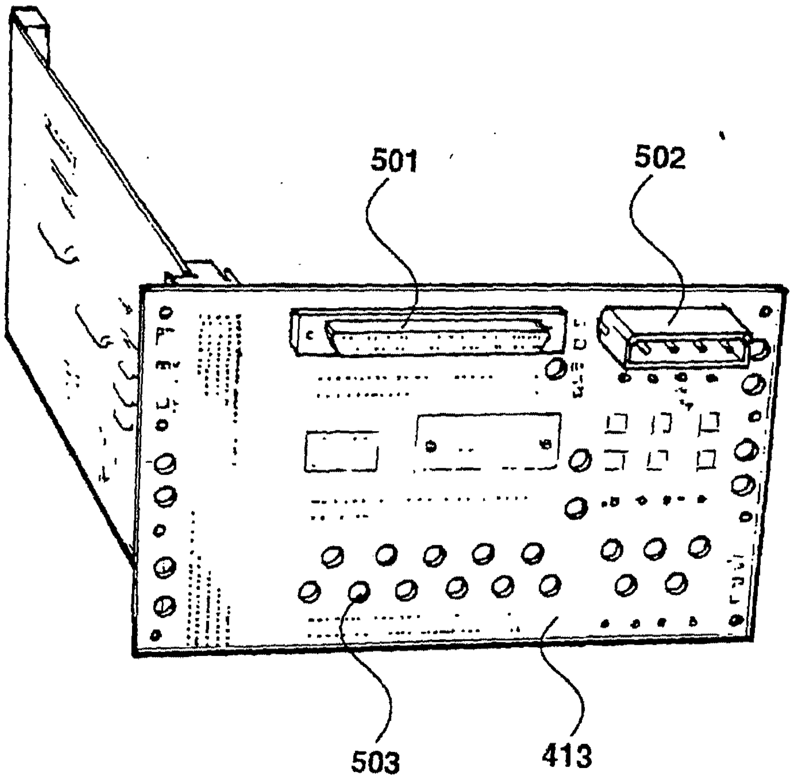


Figure 5

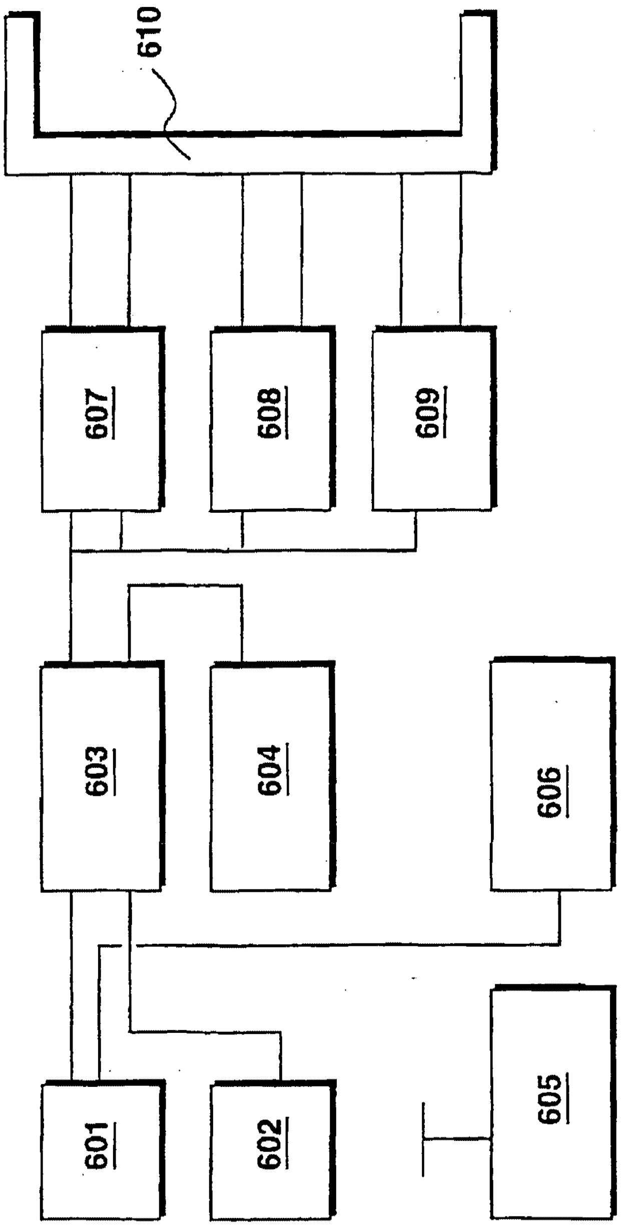


Figure 6

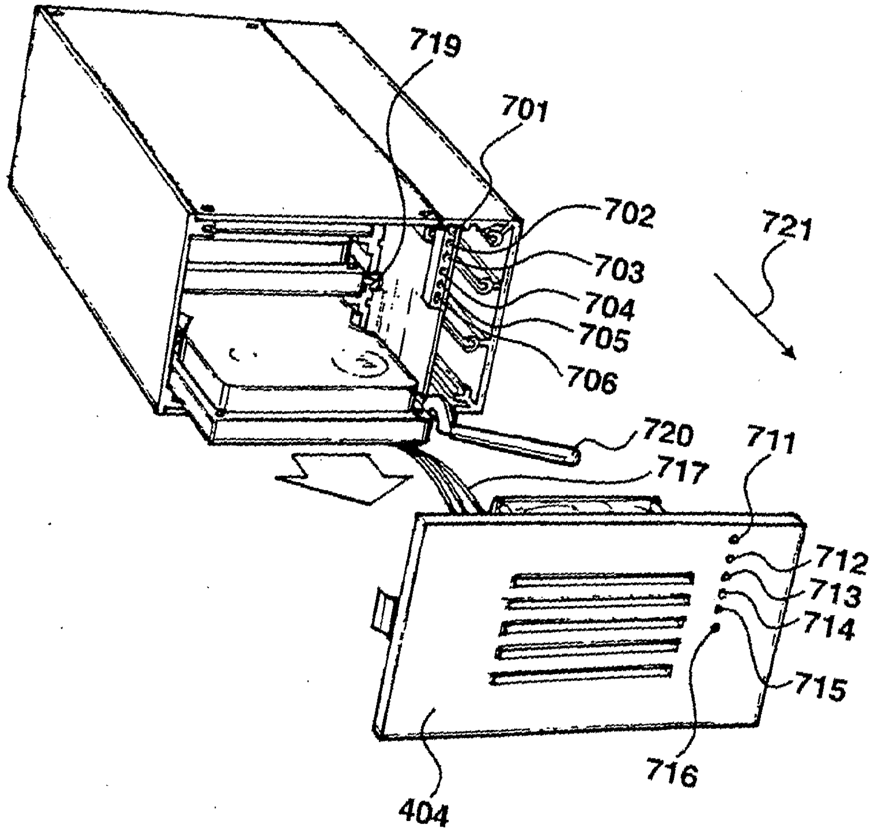

719
701
702
703
704
705
706
721
720
717
711
712
713
714
715
716
404

Figure 7

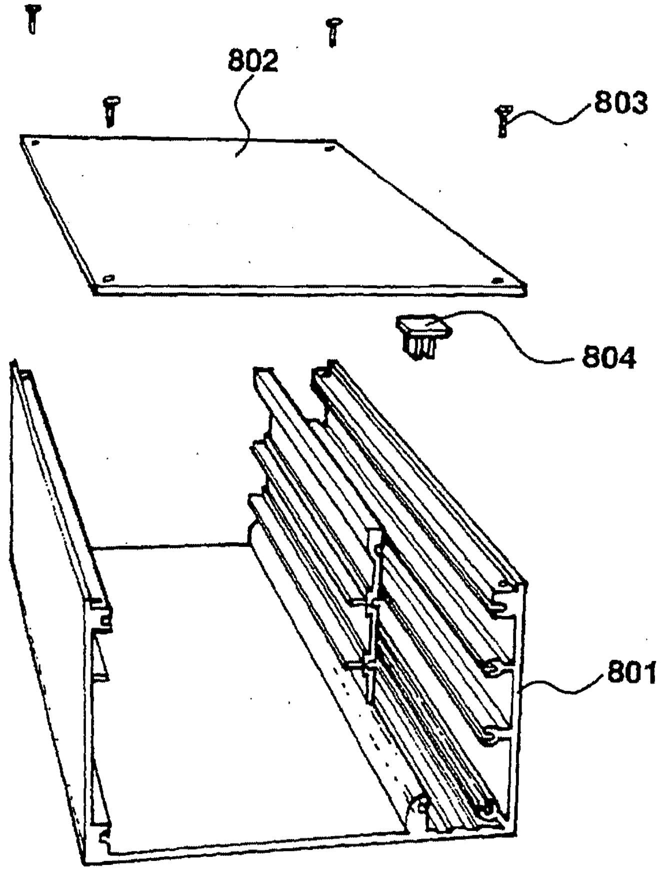


Figure 8

Data Storage

The present invention relates to an array of magnetic disks configured to store machine readable data in a protected way, such that data is recoverable in the event of disk failure.

Arrays configured to store machine readable data in a protected way are known and are often referred to as a redundant array of inexpensive disks, usually abbreviated to the acronym "RAID". Several RAID procedures are known and most of these share the approach of generating redundant data by an exclusive ORing process from which, in the event of any of the disks failing, all of the data can be reconstituted from the remaining operational disks.

When all of the disks are operational, the array is said to be working in its protected mode. In the event of one disk failure, the system may still remain operational, in that data may be read from the disks, but the data ceases to be protected and a further disk failure would result in data loss. With a single disk failure the system is said to be working in an unprotected mode at which point an operator would be advised that disk replacement is required and that the lost data needs to be reconstituted. Thus, a disk would be physically removed, replaced and then the lost data would be reconstituted on to the new disk.

As personal computer systems and workstations become more powerful, allowing more sophisticated software applications to be executed and the degree of data storage available in such systems increases, with disks containing several gigabytes of data now becoming widely used, a greater demand has been created for the installation of protected systems using disk redundancy. Complete RAID subsystems may be purchased for external connection but a problem with such known systems is that the cost can be very prohibitive. In many situations, the cost of such a RAID system tends to be higher than the cost of a personal computer system. Thus, there is a requirement for providing RAID protection at reduced cost.

Personal computer systems are usually housed in desktop units or tower units having spare bays allowing additional disks to be received. Thus, it is possible for many hard disk drives to be included within a tower housing and additional interface cards may be provided if required. Thereafter, it is possible for the RAID calculations to be effected by the resident host CPU, such that the additional extra cost is quite modest. However, a major problem with such a configuration is that a significant processor overhead is required in order to perform the RAID calculations, resulting in a severe degradation in overall system performance.

According to a first aspect of the present invention, there is provided a plurality of data storage devices configured to store machine readable data in a protected way such that data is recoverable in the event of a single device failure, wherein the devices are housed for application directly into an existing disk bay for a computer; the devices are connectable to a disk interface as if they were a single conventional storage volume; and said devices are controlled by an operating system installed on a computer as if they were a single storage volume.

In a preferred embodiment, the disks are interfaced to an IDE connection and three disks may be received in respective IDE connections.

Preferably, the array presents a SCSI interface to a host computer and the array may be configured to be housed in two or more five and one quarter inch drive bays.

According to a second aspect of the present invention, there is provided a method of equipping a personal computer with a plurality of data storage devices configured as a redundant array by interfacing said devices to conventional five and one quarter inch drive bays, such that protected machine readable data is recoverable in the event of a single disk failure, comprising the steps of supporting the array within an existing disk bay for a computer, connecting the array to the computer as if it were a single conventional computer disk; and controlling said drives by an operating system installed on a computer as if it were a single storage volume.

The invention will now be described by way of example only, with reference to the accompanying drawings, in which:

Figure 1 shows a personal computer system;

Figure 2 shows an array of disks being inserted into a computer system;

Figure 3 details the array shown in *Figure 2*;

Figure 4 shows an exploded view of the array identified in *Figure 3*;

Figure 5 shows a rear face view of the array back plane;

Figure 6 shows a circuit for implementing RAID calculations; and

Figure 7 illustrates the removal of a damaged disk from the array; and

Figure 8 shows an alternative embodiment for the extrusion identified in *Figure 4*.

A personal computer system is shown in *Figure 1* in which a main system tower **101** supplies visual information to a visual display unit **102** and receives manual commands via a keyboard **103**. The main system tower houses a central processing unit, memory circuits and other standard associated electronics as is well known in the art. The personal computer system may be an IBM PC type system, a Mackintosh system or any other computer type equipment used for individual use, possibly in a networked configuration. Alternatively, the main system tower **101** may constitute a network server, possibly running an appropriate server operating system, such as Windows NT server.

Tower **101** includes conventional five and one quarter inch disk bays. Within these disk bays a plurality of devices have been mounted, including a three and a half inch floppy disk drive **105**, a tape streamer **106**, a CD ROM drive **107** and an array of magnetic disks **108**, embodying the present invention.

Array **108** is detailed in *Figure 2* and is shown being installed into the main system tower **101**. The array **108** of magnetic disks is configured to store machine readable data in a protected way such that data is recoverable in the event of a single disk failure. The array of disks is housed for application directly into an existing disk bay of a computer, such as the main system tower **101**. The array is connectable to the computer as if it were a single conventional computer disk and the array is operated by an operating system installed on the computer as if it were a single disk.

Each empty drive bay is protected by a removable plastic cover and unit **107** locates within an aperture equivalent to the width of two bays, requiring the removal of two such covers. The array includes a housing **201**, locatable within the two bay aperture and towards its rear includes conventional power and data connectors; such that the housing as a whole is connected to the main system tower using a conventional SCSI connection. Thus, the main system perceives the disk array as if it were a single disk and the operating system, executed by the main system, controls the operation of the array using equivalent commands to those required for the operation of a single storage volume.

The array **107** is detailed in *Figure 3* and contains a total of three IDE drives **301**, **302** and **303**. An exploded view of the array is illustrated in *Figure 4*, which shows each of the individual IDE drives **301**, **302** and **303** being supported by aluminium extrusions, in the form of a left extrusion **401** and a right extrusion **402**. These extrusions hold the disk drives **301**, **302** and **303** firmly in place and facilitate the removal and replacement of individual disk drives when disk failure occurs.

Disk drives **301**, **302** and **303** are located in relatively close proximity and in order to maintain preferred operational temperatures, an electric fan **403** is positioned between the front of the disk drives and a front housing **404**. In this respect, the main front housing includes ventilation grilles **405**.

Each IDE drive **301**, **302** and **303** locates within a conventional IDE socket **406**, **407**, **408**, in addition to respective power supply sockets **409**, **410**, **411**. Thus, from the perspective of each IDE drive, the physical drives are located into sockets substantially similar to those found on an IDE bus of a standard computer system.

RAID calculations are performed within the device itself, using conventional hardware RAID circuitry mounted on circuit board **412**, having electrical connections to the back plane circuit board **413**. Right extrusion **402** defines a cavity **414**, configured to receive circuit board **412**. The extrusions **401** and **402** are held in position by an upper plate **415** and a lower plate **416**, secured by appropriate bolts **417**.

The rear face of back plane **413** is illustrated in *Figure 5*. The back plane includes a conventional SCSI socket **501** and a power supply socket **502**. The array therefore presents itself to the main system as a single disk drive, requiring a single disk drive connection via SCSI interface **501**.

Back plane **413** also includes rows of holes **503** to facilitate ventilation of the disks. Thus, cooling air is brought in through ventilation holes **405**, blown between the disks **301**, **302** and **303** and then exits through holes **503**.

The circuit implemented on board **412** is illustrated diagramatically in *Figure 6*. The circuit includes a central processing unit **601** which communicates with an input/output circuit **602** via a CPU bridge **603**. In addition, operation of CPU **601** is controlled by a CPU mode select circuit **604**. Power from the housing is directed to a three volt supply regulating circuit **605**, arranged to supply power to operational circuits via supply rails.

The CPU **601** receives data relating to the operational environment from an environmental detecting circuit **606**. This information may be received directly, as shown in *Figure 6*, or it may be directed via other control circuitry to allow combined environmental information to be returned to the CPU **601**.

Further output circuitry includes IDE controllers **607** and **608** and a SCSI controller **609**. These circuits communicate with the back plane sockets via a one hundred and eighty way connector **610**.

Input/output circuit **602** supplies driving current to six LED's **701**, **702**, **703**, **704**, **705** and **706** shown in *Figure* 7. Each of these LED's is visible by means of respective holes **711**, **712**, **713**, **714**, **715** and **716** in the front panel **404**. Each LED is a Hewlett Packard HSMF-C655 and actually includes a green LED and a red LED which may be operated independently.

LED **701** indicates the overall operational integrity of the system and primarily confirms that CPU **601** is operating correctly. Thus, when the system is fully operational, LED **701** is illuminated green. Alternatively, if faults have been detected within the controller, LED **701** is illuminated red.

LED **702** represents the environmental monitoring status and is primarily concerned with operational temperature. Environmental circuit **606** includes a temperature sensor and a fault condition is generated if this sensor detects that operational temperatures have become excessive. In addition, a tachometer is associated with fan **403** and a fault condition is generated if this detects that rotation of the fan has ceased. Malfunction of fan **403** represents a serious problem in that this could result in all three drives being permanently damaged such that no protection is offered by the RAID configuration. The system also detects the presence of appropriate voltages on voltage supply rails, as supplied by power supply unit **605** in addition to detecting appropriate terminator power on the SCSI bus.

When the supply rail voltages are correct, SCSI terminator power is correct, the fan is operational and the system is working at its optimal operational temperature, LED **702** is illuminated green. If the system encounters problems and diverges from its preferred operational characteristics, such a condition is detected and LED **702** is illuminated orange. Under these conditions, further operation of the system is permitted but warnings may be generated to the effect that a job should be closed down and that the device should be investigated. If problems continue and the situation worsens, particularly if the operational temperature becomes very high, LED **702** is illuminated red. Under these conditions, power to the drives is removed and an error condition is generated such that further access to the drives is not permitted.

LED **703** indicates that the SCSI connection is fully operational by being illuminated green. Furthermore, when the SCSI bus is actually in use, LED **703** is illuminated orange.

LED's **704**, **705** and **706** represents operational characteristics of the individual drives **301**, **302** and **303** respectively. When the drives are operational, the LED's are illuminated green and then illuminated orange when the actual data transfer takes place. Furthermore, if a disk error is detected, to the effect that an individual disk has failed, its respective LED is illuminated red.

In response to a single disk failure, it is preferable for the system to be placed off-line and for the damaged disk to be replaced immediately so that the lost data may be reconstituted and the system returned to protected mode operation. In order to replace a disk, the front panel is removed, an operation facilitated by the front panel **404** being retained simply to the main housing by means of an interference connection. Having removed the front panel **404** it is restrained by wires **717** required for supplying electrical power to fan **403**.

The disk drives include tapped holes towards their front-right corner and each of said tapped holes receives a threaded stud **719**. Stud **719** allows its respective disk **301** to **303** to be removed by the application of a stud hook **720**. Force is applied in the direction of arrow **721**, thereby forcing the respective disk drive away from its IDE and data sockets, such as sockets **406** and **409** etc.

An alternative embodiment is illustrated in *Figure 8*. In this embodiment, side panels and a base panel are fabricated as a single extrusion **801**. The housing is then completed by the application of a top panel **802**. The top panel **802** is secured to the lower extrusion **801** by means of bolts **803** and circuitry held within the extrusion is further secured by an adhesive clip **804**.

Claims

1. A plurality of data storage devices configured to store machine readable data in a protected way such that data is recoverable in the event of a single device failure, wherein

the devices are housed for application directly into an existing disk bay for a computer;

the devices are connectable to a disk interface as if they were a single conventional storage volume; and

said devices are controlled by an operating system installed on a computer as if they were a single storage volume.

2. Data storage devices according to claim 1, wherein said storage devices are magnetic disk drives.

3. Data storage devices according to claim 2, wherein the magnetic disks are interfaced to an IDE connection.

4. Data storage devices according to claim 3, wherein three disks are received in respective IDE connections.

5. Data storage devices according to any of claims 1 to 3, wherein said devices present a SCSI interface to a host computer.

6. Data storage device according to any of claims 1 to 5, configured to be housed in two or more five and one quarter inch drive bays.

7. Data storage devices according to any of claims 1 to 6, including means for detecting when said devices are operating in non-ideal conditions.

8. Data storage devices according to claim 7, including means for detecting when said devices are operating at excessive temperatures.

9. Data storage devices according to claim 7 or claim 8, including means for detecting non-operation of a cooling fan.

10. Data storage devices according to claim 7 or claim 8, including means for directly detecting an excessive operational temperature.

11. Data storage devices according to any of claims 7 to 10, including means for removing drive power to said devices upon detecting a non-ideal operating condition.

12. Data storage devices according to any of claims 1 to 11, including a detachable front panel and a cooling fan secured to said front panel, including ventilation openings arranged to direct a cooling air-stream between the individual devices.

13. A plurality of data storage devices according to any of claims 1 to 12, wherein said devices are connectable in a computer housing and the devices are controlled by the operating system of said computer.

14. A method of equipping a personal computer with a plurality of data storage devices configured as a redundant array by interfacing said devices to conventional five and one quarter inch drive bays, such that protected machine readable data is recoverable in the event of a single disk failure, comprising the steps of

supporting the array within an existing disk bay for a computer;

connecting the array to the computer as if it were a single conventional computer disk; and

controlling said drives by an operating system installed on a computer as if it were a single storage volume.

15. A method according to claim **14**, wherein said data storage devices are magnetic disk drives.

16. A method according to claim **15**, wherein said magnetic disk drives are interfaced to an IDE connection.

17. A method according to claim **16**, wherein three disks are received in respective IDE connections.

18. A method according to any of claims **14** to **17**, wherein said devices present a SCSI interface to a host computer.

19. A method according to any of claims **14** to **18**, wherein said devices are housed in two or more five and one quarter inch drive bays.

20. A method according to any of claims **14** to **19**, wherein non-ideal operating conditions for said devices are detected.

21. A plurality of data storage devices substantially as herein described with reference to the accompanying Figures.

22. A method of equipping a personal computer substantially as herein described with reference to the accompanying Figures.

The Patent Office

12

INVESTOR IN PEOPLE

Application No:	GB 9820213.8	**Examiner:**	Julyan Elbro
Claims searched:	1 to 22	**Date of search:**	4 January 1999

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.Q): G5R (RGB, RB33, RAC); G4A (AES)

Int Cl (Ed.6): G06F 11/10; G11B 20/18

Other: EDOC WPI

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	EP 0795812 A1	HITACHI see figure 1 and pages 2-3.	1-20
X	EP 0717357 A2	SYMBIOSIS LOGIC see abstract and figure 2.	1-20
X	EP 0569313 A2	INTERNATIONAL BUSINESS MACHINES see abstract and figures 1 and 3.	1-20
X	EP 0569236 A2	COMPAQ see figure 2 and pages 2-4.	1-20
X	EP 0485110 A2	ARRAY TECHNOLOGY see abstract.	1-20
X	EP 0450801 A2	INTERNATIONAL BUSINESS MACHINES see abstract, column 22 line 34 to column 23 line 11, and column 27 lines 15-25.	1-20
X	WO 93/18455 A1	ARRAY TECHNOLOGY see abstract, figure 1, and page 10 lines 2-26.	1-20
X	WO 91/20076 A1	STORAGE TECHNOLOGY see abstract and figure 1.	1-20
X	WO 91/14982 A1	SF2 CORPORATION see abstract and figures 1 and 2.	1-20

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		

CHARTER MARK

The Patent Office

13

INVESTOR IN PEOPLE

Application No:	GB 9820213.8	**Examiner:**	Julyan Elbro
Claims searched:	1 to 22	**Date of search:**	4 January 1999

Category	Identity of document and relevant passage	Relevant to claims
X	US 5651132 A HITACHI see abstract and figure 1.	1-20

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		

B8

Bibliographic Fields

Document Identity

(19)【発行国】	(19) [Publication Office]
日本国特許庁(JP)	Japan Patent Office (JP)
(12)【公報種別】	(12) [Kind of Document]
公開特許公報(A)	Unexamined Patent Publication (A)
(11)【公開番号】	(11) [Publication Number of Unexamined Application]
特開平6－301607	Japan Unexamined Patent Publication Hei 6- 301607
(43)【公開日】	(43) [Publication Date of Unexamined Application]
平成6年(1994)10月28日	1994 (1994) October 28*

Public Availability

(43)【公開日】	(43) [Publication Date of Unexamined Application]
平成6年(1994)10月28日	1994 (1994) October 28*

Technical

(54)【発明の名称】	(54) [Title of Invention]
マルチアクセスI／O制御方式	**MULTI ACCESS I/O CONTROL SYSTEM**
(51)【国際特許分類第 5 版】	(51) [International Patent Classification, 5th Edition]
G06F 13/00 351 B 7368-5B	G06F 13/00 351 B 7368-5B
13/12 340 F 8133-5B	13/12 340 F 8133-5B
【請求項の数】	[Number of Claims]
5	5
【出願形態】	[Form of Application]
OL	OL
【全頁数】	[Number of Pages in Document]
10	10

Filing

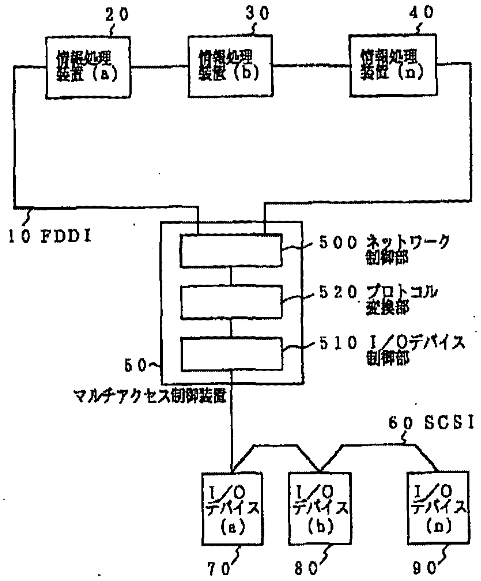
【審査請求】	[Request for Examination]
未請求	Unrequested
(21)【出願番号】	(21) [Application Number]
特願平5－86000	Japan Patent Application Hei 5- 86000
(22)【出願日】	(22) [Application Date]
平成5年(1993)4月13日	1993 (1993) April 13*

Parties

Applicants

(71)【出願人】	(71) [Applicant]

【識別番号】	[Identification Number]
000005108	000005108
【氏名又は名称】	[Name]
株式会社日立製作所	**HITACHI LTD. (DB 69-054-1503)**
【住所又は居所】	[Address]
東京都千代田区神田駿河台四丁目6番地	Tokyo Chiyoda-ku Kanda Surugadai 4-Chome 6
Inventors	
(72)【発明者】	(72) [Inventor]
【氏名】	[Name]
宇賀神 敦	*** Atsushi
【住所又は居所】	[Address]
神奈川県海老名市下今泉810番地 日立製作所 オフィスシステム事業部内	Kanagawa Prefecture Ebina City Shimoimaizumi 810address Hitachi office systems department *
Agents	
(74)【代理人】	(74) [Attorney(s) Representing All Applicants]
【弁理士】	[Patent Attorney]
【氏名又は名称】	[Name]
鈴木 誠	Suzuki *
Abstract	
(57)【要約】	(57) [Abstract]
【目的】	[Objective]
複数の情報処理装置から複数の I/O デバイスへのアクセスを可能とする。	access to I/O device of plural is made possible from information processing apparatus of plural .
【構成】	[Constitution]
複数の情報処理装置 20,30,40 とマルチアクセス制御装置 50 を FDDI10 に接続し、マルチアクセス制御装置 50 は、I/O デバイス 70,80,90 に SCSI 接続されている。	information processing apparatus 20, 30, 40 and multi access control device 50 of plural are connected to FDDI10, the multi access control device 50 SCSI is connected to I/O device 70, 80, 90.
情報処理装置は、マルチアクセス制御装置へ FDDI フレームでアクセスする。	To multi access control device access it does information processing apparatus , with FDDIframe .
ネットワーク制御部 500 は、情報処理装置からのデ-タを FDDI インタフェ-スで送受信した後、プロトコル変換部 520 では、SCSI プロトコルに変換し、I/O デバイス制御部 510 を介して I/O デバイスをアクセスする。	data from information processing apparatus transmission and reception after doing, in protocol conversion section 520,it converts network control unit 500, to SCSI protocol with FDDIinterface , through I/O device control unit 510,access it does I/O device . .



Claims

【特許請求の範囲】

【請求項 1】

ネットワークを介して複数の情報処理装置を接続したシステムにおいて、該ネットワークのインタフェース制御を行うネットワーク制御手段と、I/Oインタフェースを介して複数のI/Oデバイスを制御する I/O デバイス制御手段と、該ネットワーク制御手段と I/O デバイス制御手段のインタフェース変換を行うプロトコル変換手段からなるマルチアクセス制御手段を設け、前記複数の情報処理装置は該マルチアクセス制御手段を介して前記複数の I/O デバイスにアクセスすることを特徴とするマルチアクセス I/O 制御方式。

【請求項 2】

前記 I/O デバイス制御手段を前記 I/O デバイス内の制御部に内蔵することを特徴とする請求項 1 記載のマルチアクセス I/O 制御方式。

【請求項 3】

前記複数の情報処理装置が実行した処理データを、前記マルチアクセス制御手段を介して前記所定の I/O デバイスに格納し、該情報処理装置の障害発生時に予備の情報処理装置に切り

[Claim(s)]

[Claim 1]

Through network , through network control means and I/O interface which do interface control of said network in system which connects information processing apparatus of plural , the multi access control means which consists of protocol conversion means which converts I/O device control means and the said network control means and I/O device control means which control I/O device of plural interface providing, As for information processing apparatus of aforementioned plural through said multi access control means , in the I/O device of aforementioned plural access multi access I/O control system . which designatesthat it does as feature

[Claim 2]

multi access I/O control system . which is stated in Claim 1 which designates that theaforementioned I/O device control means is built in to control unit inside theaforementioned I/O device as feature

[Claim 3]

Treatment data which information processing apparatus of aforementioned plural executed, through aforementioned multi access control means , it houses in theaforementioned predetermined I/O device , changes to information processing

替え、該予備の情報処理装置は、前記処理データが格納された I/O デバイスを参照して処理を継続することを特徴とする請求項 1 記載のマルチアクセス I/O 制御方式。

apparatus of preparatory at time of damage of said information processing apparatus , as for information processing apparatus of said preparatory , referring to I/O device where aforementioned treatment data is housed, the multi access I/O control system . which it states in Claim 1 which designates that it continuestreatment as feature

【請求項 4】

[Claim 4]

前記各情報処理装置は、ローカル I/O デバイスを有し、該ローカル I/O デバイスに記録される情報を、前記マルチアクセス制御手段を介して、前記情報処理装置に対応する I/O デバイスに格納してバックアップすることを特徴とする請求項 1 記載のマルチアクセス I/O 制御方式。

information which aforementioned each information processing apparatus , possesses local I/O device , isrecorded to said local I/O device , through aforementioned multi access control means , housing in I/O device which corresponds to aforementioned information processing apparatus , backup the multi access I/O control system . which is stated in Claim 1 which designates thing which isdone as feature

【請求項 5】

[Claim 5]

前記 I/O インタフェースは、送信専用のインタフェースと受信専用のインタフェースから構成されていることを特徴とする請求項 1 記載のマルチアクセス I/O 制御方式。

As for aforementioned I/O interface , from interface of transmission dedicated and interface of reception dedicated configuration multi access I/O control system . whichis stated in Claim 1 which designates that it is done as feature

Specification

【発明の詳細な説明】

[Description of the Invention]

【0001】

[0001]

【産業上の利用分野】

[Field of Industrial Application]

本発明は、マルチアクセス I/O 制御方式に関し、特にネットワークを介して複数の情報処理装置を接続したシステムにおいて、複数の情報処理装置からアクセス可能な I/O デバイスの制御方式に関する。

this invention regards multi access I/O control system , through especially network , from the information processing apparatus of plural it regards control system of accessible I/O device in system which connects information processing apparatus of plural .

【0002】

[0002]

【従来の技術】

[Prior Art]

I/O デバイスを複数の処理装置によって共用する技術として、例えば、特開平 4-196737 号公報に記載された方式がある。

There is a system which is stated in for example Japan Unexamined Patent Publication Hei 4- 196737disclosure as technology whichshares I/O device with processing unit of plural .

この方式においては、1 台の保守用コンソールを複数台のホストコンピュータで共有するもので、ホストコンピュータからの受信データをバッファリングした後、コントロールユニットに通知し、該コントロールユニットはホスト選択用のスイッチを設定し、選択されたホストのデータを保守用コンソールに出力する。

Regarding this system , being something which shares console for the conservation of 1 with host computer of plural table , buffering after doing, itnotifies received information from host computer to control unit , said control unit sets the Switch for host selection, outputs data of host which isselected to console for conservation .

【0003】

[0003]

【発明が解決しようとする課題】

[Problems to be Solved by the Invention]

しかしながら、上記した技術は、各ホストインタフェース毎に独立にバッファを設けているので、ハ

But, because technology which was inscribed in each every host interface has provided buffer in independence, amount of

ードウェア量が多くなり、また、ホスト選択スイッチのような固有のハードウェアを必要とし、さらに、ホスト数に相当する数のホストインタフェースコネクタを必要とするので、接続するホストが多くなると装置全体が大型化するとともに、複数台のホストに対して1台のコンソールを接続した構成しか採ることができないという欠点があった。

【0004】

本発明の目的は、複数の情報処理装置から複数のI/Oデバイスへのアクセスを可能とするマルチアクセスI/O制御方式を提供することにある。

【0005】

【課題を解決するための手段】

前記目的を達成するために、請求項1記載の発明では、ネットワークを介して複数の情報処理装置を接続したシステムにおいて、該ネットワークのインタフェース制御を行うネットワーク制御手段と、I/Oインタフェースを介して複数のI/Oデバイスを制御するI/Oデバイス制御手段と、該ネットワーク制御手段とI/Oデバイス制御手段のインタフェース変換を行うプロトコル変換手段からなるマルチアクセス制御手段を設け、前記複数の情報処理装置は該マルチアクセス制御手段を介して前記複数のI/Oデバイスにアクセスすることを特徴としている。

【0006】

請求項2記載の発明では、前記I/Oデバイス制御手段を前記I/Oデバイス内の制御部に内蔵することを特徴としている。

【0007】

請求項3記載の発明では、前記複数の情報処理装置が実行した処理データを、前記マルチアクセス制御手段を介して前記所定のI/Oデバイスに格納し、該情報処理装置の障害発生時に予備の情報処理装置に切り替え、該予備の情報処理装置は、前記処理データが格納されたI/Oデバイスを参照して処理を継続することを特徴としている。

【0008】

請求項4記載の発明では、前記各情報処理装置は、ローカルI/Oデバイスを有し、該ローカルI/Oデバイスに記録される情報を、前記マルチアクセス制御手段を介して、前記情報処理装置に

hardware to become many, inaddition, to need hardware of peculiar like host selection switch , becausefurthermore, host interface connector of a quantity which is suitable to quantity of host are needed, when host which is connected becomes many as device entirety does scale-up , There was a deficiency that only configuration which connects console of 1vis-a-vis host of plural table it is possible to take.

[0004]

objective of this invention is to offer multi access I/O control system which makes access to I/O device of plural possible from information processing apparatus of plural .

[0005]

[Means to Solve the Problems]

In order to achieve aforementioned objective , with invention whichis stated in Claim 1, through network , through network control means and the I/O interface which do interface control of said network in system which connects the information processing apparatus of plural , multi access control means which consists of protocol conversion means whichconverts I/O device control means and said network control means and I/O device control means which control I/O device of plural interface providing, information processing apparatus of aforementioned plural through said multi access control means , hasdesignated that access it does as feature in I/O device of theaforementioned plural .

[0006]

With invention which is stated in Claim 2, it designates thataforementioned I/O device control means is built in to control unit inside theaforementioned I/O device as feature.

[0007]

With invention which is stated in Claim 3, treatment data which information processing apparatus of aforementioned plural executed, through theaforementioned multi access control means , it houses in aforementioned predetermined I/O device , changesto information processing apparatus of preparatory at time of damage of said information processing apparatus , the information processing apparatus of said preparatory referring to I/O device where aforementionedtreatment data is housed, has designated that it continues treatmentas feature.

[0008]

With invention which is stated in Claim 4, aforementionedeach information processing apparatus , it possesses local I/O device , information which is recorded to the said local I/O device , through aforementioned multi

対応するI/Oデバイスに格納してバックアップすることを特徴としている。

【0009】

請求項5記載の発明では、前記I/Oインタフェースは、送信専用のインタフェースと受信専用のインタフェースから構成されていることを特徴としている。

【0010】

【作用】

複数の情報処理装置とマルチアクセス制御装置がFDDIに接続され、マルチアクセス制御装置は、I/OデバイスにSCSI接続されている。

マルチアクセス制御装置は、ネットワーク制御部とプロトコル変換部とI/Oデバイス制御部から構成されている。

情報処理装置は、マルチアクセス制御装置へFDDIフレームでアクセスする。

ネットワーク制御部は、情報処理装置からのデータをFDDIインタフェ-スで送受信した後、プロトコル変換部では、SCSIプロトコルに変換し、I/Oデバイス制御部を介してI/Oデバイスをアクセスする。

これにより、従来のI/Oデバイスに何ら変更を加えることなく、マルチアクセス制御装置を付加するのみで、複数の情報処理装置から複数のI/Oデバイスを制御することができる。

【0011】

【実施例】

以下、本発明の一実施例を図面を用いて具体的に説明する。

図1は、本発明の一実施例に係るシステム構成図である。

本発明のシステムは、複数の情報処理装置20、30、40とマルチアクセス制御装置50がFDDI(FiberDistributed Data Interface)10(LAN)に接続されて構成されている。

【0012】

FDDI10に接続された情報処理装置20、30、40は、マルチアクセス制御装置50へFDDIフレームでアクセスする。

マルチアクセス制御装置50は、FDDIインタフェース制御を行うネットワーク制御部500と、

access control means , housing in I/O device whichcorresponds to aforementioned information processing apparatus , it designates that backup it does as feature.

[0009]

With invention which is stated in Claim 5, as for theaforementioned I/O interface , it designates that configuration it is done asfeature from interface of transmission dedicated and interface of thereception dedicated .

[0010]

[Working Principle]

information processing apparatus and multi access control device of plural are connected by FDDI, the multi access control device SCSI is connected to I/O device .

multi access control device configuration is done from network control unit and protocol conversion section and I/O device control unit .

To multi access control device access it does information processing apparatus , with FDDIframe .

data from information processing apparatus transmission and reception after doing, in protocol conversion section,it converts network control unit , to SCSI protocol with FDDIinterface , through I/O device control unit ,access it does I/O device .

Because of this, multi access control device is added only, can control I/O device of the plural from information processing apparatus of plural without adding what modification to conventional I/O device .

[0011]

[Working Example(s)]

Below, one Working Example of this invention is explained concretely making use of drawing .

Figure 1 is system diagram which relates to one Working Example of this invention .

system of this invention is done, information processing apparatus 20, 30, 40 and multi access control device 50 of plural FDDI (FiberDistributed data interface) being connected by 10 (LAN), configuration .

[0012]

To multi access control device 50 access it does information processing apparatus 20, 30, 40 which is connected to the FDDI10, with FDDIframe .

multi access control device 50 configuration is done from protocol conversion section 520 which converts I/O device

SCSI60に接続されているI/Oデバイス70,80,90(例えば、ハードディスクなどの記憶媒体や回線などの通信手段)の制御を行うI/Oデバイス制御部510と、FDDIプロトコル及びSCSIプロトコルのインタフェース変換を行うプロトコル変換部520から構成されている。

【0013】

図2は、マルチアクセス制御装置50のブロック構成図である。

マルチアクセス制御装置50において、ネットワーク制御部500と、I/Oデバイス制御部510と、RAM523と、アクセス制御部524はI/Oバス525によって接続され、プロセッサ521と、ROM522と、アクセス制御部524はプロセッサバス526によって接続されている。

【0014】

プロトコル変換を行うためのプログラムは、ROM522に格納され、プロセッサ521上で動作する。

本実施例では、I/Oバス525の使用率を下げるためにプロセッサバス526を設けているが、情報処理装置20、30、40からのアクセス頻度が低い場合には、I/Oバスとプロセッサバスを同一バスにして構成してもよい。

【0015】

アクセス制御部524は、ネットワーク制御部500またはI/Oデバイス制御部510からプロセッサ521への割込み制御を行うと共にプロセッサ521からRAM523、ネットワーク制御部500、I/Oデバイス制御部510へのアクセス制御並びにネットワーク制御部500、I/Oデバイス制御部510からRAM523へのアクセス制御を行っている。

【0016】

ROM522には、プログラムの他にFDDIのMAC(Media Access Control)アドレスを格納する。

RAM523は、データ送信及び受信用のバッファとして使用するほかに、ネットワーク制御部500、I/Oデバイス制御部510への制御を行うためのディスクリプタ領域として使用する。

また、マルチアクセス制御装置内のステータス管理やI/Oデバイス毎の管理等のためにテーブルとして使用する。

【0017】

図3は、情報処理装置からマルチアクセス制御装置への制御フレームのフォーマットを示す図

control unit 510 and FDDIprotocol and SCSI protocol which control I/O device 70, 80, 90 (for example hard disk or other storage media and circuit or other communication means) whichis connected to network control unit 500 and SCSI 60 which do FDDIinterface control interface .

[0013]

Figure 2 is block diagram of multi access control device 50.

In multi access control device 50, network control unit 500 and I/O device control unit 510 and RAM 523 and access control section524 are connected with I/O bus 525, processor 521 and ROM 522 and access control section 524 are connected with processor bus 526.

[0014]

program in order to do protocol conversion is housed in ROM 522, operates on processor 521.

With this working example , processor bus 526 is provided in order to lower usage of I/O bus 525, but when access frequency from information processing apparatus 20, 30, 40 is low, configuration it ispossible to do with I/O bus and processor bus as same bus .

[0015]

access control section 524, as interruption control to processor 521 is done from network control unit 500 or I/O device control unit 510, from processor 521 does access control to RAM 523 from access control and network control unit 500, I/O device control unit 510 to RAM 523, network control unit 500, I/O device control unit 510.

[0016]

In ROM 522, MAC (Media access control) address of FDDI is housed to other than program .

Besides you use as buffer for data transmission and reception, youuse RAM 523, as [disukuriputa] region in order to control to network control unit 500, I/O device control unit 510.

In addition, you use management or other for every status management and I/O device inside multi access control device as table .

[0017]

Figure 3 is figure which shows format of control frame to multi access control device from information processing

である。

図 3 において、FDDI ヘッダ 100(ANSI 標準)に SNAP ヘッダ 110、IP ヘッダ 120、TCP ヘッダ 130(全て Request For Comment で規定されている)、データ 140 を付加し制御を行う。

【0018】

情報処理装置 20,30,40 とマルチアクセス制御装置 50 との間の送達確認及び順序制御は、TCP(Transmission Cotrol Protocol)により行う。

【0019】

データ 140 は、制御ブロック 1410、1450 と送信 I/O データ 1460 から構成されていて、制御ブロックは、1 乃至複数のブロックからなる。

また、送信 I/O データ 1460 は付加してもよいし、あるいは付加しなくてもよいが、最大フレーム長は、FDDI 規格に準拠する必要がある。

【0020】

制御ブロック 1410、1450 は 28 バイトから構成される。

制御ブロック 1410 において、制御ブロック長 1411 は、2 バイトのフィールドであり、制御ブロックの総バイト長を示す。

コマンドチェインビット 1412 は、1 ビットからなり、異なるコマンドの制御ブロックが連続しているか否かを示す。

“0”の時はコマンドチェインなし、“1”の時はコマンドチェインありを示す。

【0021】

デバイス ID1413 は、2 バイトのフィールドであり、SCSI_ID 4 ビット、LUN(Logical Unit Number) 4 ビット、拡張 LUN 8 ビットから構成される。

CDB フォーマット 1414 は、5 ビットのフィールドである。

CDB は、6 バイト,10 バイト,12 バイトがあるのでその種別を示している。

“0”が 6 バイト、“1”が 10 バイト、“2”が 12 バイトを示す。

【0022】

不正長抑止ビット 1415 は、1 ビットのフィールドである。

リード要求と実際の読みだしデータ長が異なってもエラー報告しないためのビットである。

apparatus .

In Figure 3 , SNAPheader 110 , IP header 120, TCP header 130 (Being stipulated with all Request For Comment, it is), it adds data 140 to FDDIheader 100 (ANSIstandard) andcontrols.

[0018]

It does sending verification and order control between information processing apparatus 20, 30, 40 and multi access control device 50, with TCP (transmission Cotrol protocol).

[0019]

As for data 140, configuration being done from control block 1410, 1450 and thetransmission I/O data 1460, as for control block , it consists of block of 1 to plural .

In addition, it is possible to add transmission I/O data 1460 it is notnecessary, and, or to add, but maximum frame length has necessity to conformto FDDIstandard .

[0020]

control block 1410, 1450 configuration is done from 28 byte .

In control block 1410, control block length 1411, with field of 2 byte , shows theentire byte length of control block .

command chain bit 1412 consists of 1 bit , shows whether or not which control block of the different command is continual.

When " 0 " being, there is a command chain and shows time of command chain none , *1&apos;&apos; .

[0021]

device ID1413, with field of 2 byte , SCSI _ID 4bit , LUN (Logical Unit Number) configuration is donefrom 4 bit , extended LUN 8bit .

CDBformat 1414 is field of 5 bit .

Because CDB are 6 byte , 10byte , 12byte , type has been shown.

" 0 " 6 byte , *1&apos;&apos; 10 byte , *2* 12 byte are shown.

[0022]

Illegitimate long control bit 1415 is field of 1 bit .

read request and actual it starts reading and data length differs and error it is a bit because it does not report.

“1”のときエラー報告せず、“0”のときエラー報告する。

【0023】

終了報告ビット1416は、1ビットのフィールドである。

“1”のとき処理終了を終了報告ブロック(図4)で報告する。

“0”の時は報告しない。

【0024】

コマンド1421は、8ビットのフィールドである。

データ受信、データ送信、マルチアクセス制御装置50に対する指示などを示す。

SCSI NO.1422は、8ビットのフィールドである。

マルチアクセス制御装置50内で複数のSCSIを制御する場合に、どのSCSIかを識別するための情報である。

シーケンスNO.1420は、16ビットのフィールドである。

情報処理装置20,30,40からの要求とマルチアクセス制御装置50からの終了報告を対応させるための情報である。

【0025】

データカウント1418は、4バイトのフィールドであり、送信または受信するデータ長を示す。

CDB1419は、本実施例では10バイトであり、SCSI規格に準拠したCDBを格納する。

【0026】

図4は、マルチアクセス制御装置から情報処理装置への終了フレームのフォーマットを示す図である。

図において、FDDIヘッダ100、SNAPヘッダ110、IPヘッダ120、TCPヘッダ130は、前述したものと同様である。

データ140は、終了報告ブロック1470と受信I/Oデータ1480から構成されている。

【0027】

終了報告ブロック1470は、16バイトから構成されている。

終了報告ブロック長1471は、16ビットのフィールであり、終了報告ブロックの総バイト数を示す。

終了報告チェインビット1472は、1ビットのフィールドであり、終了報告が複数ある場合に“1”を

" At time of 1 &apos;&apos; error it does not report, " when 0 "being, error it reports.

[0023]

End report bit 1416 is field of 1 bit .

" At time of 1 &apos;&apos; treatment end is reported with endreport block (Figure 4).

When " 0 " being, it does not report.

[0024]

command 1421 is field of 8 bit .

data reception , data transmission and indication etc for multi access control device 50 are shown.

SCSI NO.1422 is field of 8 bit .

When SCSI of plural is controlled inside multi access control device 50, it is a information in order to identify which SCSI .

sequence NO.1420 is field of 16 bit .

It is a information because end report from multi access control device 50 it corresponds withrequest from information processing apparatus 20, 30, 40.

[0025]

data count 1418 with field of 4 byte , shows data length which ittransmits or receives, or.

CDB1419 with this working example with 10 byte , houses CDB whichconforms to SCSI standard .

[0026]

Figure 4 is figure which shows format of end frame to the information processing apparatus from multi access control device .

In figure, FDDIheader 100, SNAPheader 110 , IP header 120, TCP header 130 is similar to those which are mentionedearlier.

data 140 configuration is done from end report block 1470 and thereception I/O data 1480.

[0027]

End report block 1470 configuration is done from 16 byte .

End report block length 1471, with fee jp11 of 16 bit , shows theentire number of bytes of end report block .

End report chain bit 1472, when with field of 1 bit , end report is a plural , " sets 1 &apos;&apos; .

設定する。

【0028】

ステータス1474は、16ビットのフィールドである。

このフィールドは、エラーの軽重を示すシビリティビット4ビット、エラーステータスフィールド12ビットから構成される。

SAVE DMAカウント1473は、4バイトのフィールドであり、データカウント1418と実際に処理完了したバイト数の差分を示す。

例えば、データカウント1418が1000バイトで、実際に処理したデータが1000バイトの場合、該フィールドは、0となる。

【0029】

図5は、情報処理装置20、情報処理装置30からマルチアクセス制御装置50へのアクセスシーケンスを示す。

以下、情報処理装置からI/Oデバイスへデータを書き込む場合の実施例の動作を説明する。

【0030】

情報処理装置20からマルチアクセス制御装置50へデータ書き込み指示を図3に示すフレームフォーマットで送信する。

ネットワーク制御部500はフレームを受信し、プロトコル変換部520から予め渡されたRAM523上のバッファにデータを格納する。

ネットワーク制御部500は、データ格納後、割込みをアクセス制御部524を介してプロセッサ521に通知する。

【0031】

情報処理装置20からのデータ書き込み指示の後、情報処理装置30からマルチアクセス制御装置50へ、データ書き込み指示を図3に示すフレームフォーマットで送信する。

ネットワーク制御部500はフレームを受信しプロトコル変換部520から予め渡されたRAM523上のバッファにデータを格納する。

ネットワーク制御部500は、データ格納後、割込みをアクセス制御部524を介してプロセッサ521に通知する。

但し、情報処理装置20からの処理が先であるのでその処理が終了するまで処理保留となる。

is a plural , " sets 1 &apos;&apos; .

[0028]

status 1474 is field of 16 bit .

this field shows light heavy of error , [shibiritibitto] configuration it is done from4 bit , error status field 12bit .

SAVE DMA count 1473, with field of 4 byte , shows difference of number of bytes which process end is done in data count 1418 and fact.

for example data count 1418 being 1000 byte , when data which was treated actuallyis 1000 byte , said field becomes with 0.

[0029]

Figure 5 shows access sequence to multi access control device 50 from information processing apparatus 20, information processing apparatus 30.

Below, operation of Working Example when from information processing apparatus data is writtento I/O device is explained.

[0030]

From information processing apparatus 20 to multi access control device 50 it transmits with frame format which shows data writing indication in Figure 3 .

network control unit 500 receives frame , houses data in buffer on the RAM 523 which is beforehand transferred from protocol conversion section 520.

network control unit 500, after data storage , through access control section 524, notifies theinterruption to processor 521.

[0031]

After data writing indication from information processing apparatus 20, from information processing apparatus 30 to the multi access control device 50, it transmits with frame format which shows data writing indication in Figure 3 .

network control unit 500 receives frame and houses data in buffer on the RAM 523 which is beforehand transferred from protocol conversion section 520.

network control unit 500, after data storage , through access control section 524, notifies theinterruption to processor 521.

However, because treatment from information processing apparatus 20 is ahead, until thattreatment ends, it becomes treatment reservation .

【0032】

割込みを受けたプロトコル変換部 520 は、受信したフレームのヘッダを解析し TCP、IP(Internet Protocol)処理を行う。

その後、制御ブロック 1410 を解析する。

フォーマットが正常ならば SCSI NO.1422、デバイス ID1413 が示す SCSI に対してコマンドを発行する。

コマンドの発行は、RAM523 上のディスクリプタに CDB を格納した後、I/O デバイス制御部 510 内のハードウェアレジスタに起動をかけることにより行う。

コマンドを受けた I/O デバイス制御部 510 は、SCSI 規格に従ってアービトレーション、セレクション、メッセージ、コマンドフェーズを遷移した後、情報処理装置 20 によって指定された例えば I/O デバイス 70 に対してデータ転送を行う。

【0033】

この時のデータ転送は、DMA(Direct Memory Access)で行う。

データ転送終了後、I/O デバイス 70 からステータ及びコマンドコンプリートが送られてくる。

これを受けた、I/O デバイス制御部 510 はプロセッサ 521 への割込みをアクセス制御部 524 を介して通知する。

【0034】

割込みを受けたプロセッサ 521 は、RAM523 に格納されているステータスを解析する。

その後、図 4 に示した終了報告ブロック、IP ヘッダ、TCP ヘッダ、SNAP ヘッダを RAM523 上に作成し、ネットワーク制御部 500 内のハードウェアレジスタに送信指示を書き込む。

これを受けたネットワーク制御部 500 は、FDDI プロトコルに従って終了報告を情報処理装置 20 に送信する。

【0035】

情報処理装置 20 の処理が終了後、情報処理装置 30 の処理を行う。

その動作は、前述した情報処理装置 20 の場合と同様であるので、説明は省略する。

【0036】

図 6 は、マルチアクセス制御装置と I/O デバイスを一体化させた場合の他の実施例の構成を示

[0032]

protocol conversion section 520 which receives interruption analyzes the header of frame which is received and does TCP , IP (internet protocol) treatment.

After that, control block 1410 is analyzed.

command is issued format vis-a-vis SCSI which normal mule SCSI NO.1422, device ID1413 shows.

It issues command , after housing CDB in [disukuriputa] on RAM 523,by making starting on hardware register inside I/O device control unit 510.

I/O device control unit 510 which receives command , following to SCSI standard , does the data transfer transition after doing arbitration , selection , message , command phase , vis-a-vis for example I/O device 70 which isappointed with information processing apparatus 20.

[0033]

It does data transfer at time of this , with DMA (direct memory access).

After data transfer ending, stator and [komandokonpuriito] are sent from I/O device 70.

This was received, I/O device control unit 510 through access control section 524, notifies theinterruption to processor 521.

[0034]

processor 521 which receives interruption analyzes status which ishoused in RAM 523.

After that, end report block , IP header , TCP header , SNAPheader which is shown in Figure 4 is drawnup on RAM 523, transmission indication is written to hardware register inside network control unit 500.

network control unit 500 which receives this, following to FDDIprotocol , transmits endreport to information processing apparatus 20.

[0035]

Treatment of information processing apparatus 20 after ending, treats information processing apparatus 30.

Because operation is similar to case of information processing apparatus 20 which ismentioned earlier, it abbreviates explanation.

[0036]

Figure 6 is figure which shows configuration of other Working Example when multi access control device and I/O

す図である。

すなわち、一体化によって、I/O デバイス内の制御部(SCSI コントローラ)が I/O デバイス制御部 510 を肩代わりし、従って、図 2 に示す I/O デバイス制御部 510 を設ける必要がなくなり、直接 I/O デバイス内の I/O 制御部 700 に制御ブロックを渡す処理方式を採ることになる。

【0037】

図 7 は、現用系情報処理装置から予備系情報処理装置への切替えを行う場合の他の実施例の構成を示す図である。

現用系情報処理装置 21,22 は、処理を実行する場合に、マルチアクセス制御装置 50 を介して、任意の I/O デバイス 70 内に引継ぎ情報 71 を格納処理する。

そして、現用系情報処理装置 21,22 に障害が発生したとき、予備系情報処理装置 23 は I/O デバイス 70 内の引継ぎ情報 71 を読み出して、処理を続行する。

【0038】

図 8 は、情報処理装置を I/O デバイスによってバックアップする場合の他の実施例の構成を示す図であり、各情報処理装置はローカル I/O デバイスを備えた構成を採っている。

【0039】

各情報処理装置 20,30,40 は、それぞれローカル I/O デバイス 201、301、401 にデータを書き出すとともに、情報処理装置 20 は、例えば I/O デバイス 70 に、情報処理装置 30 は I/O デバイス 80 に、情報処理装置 40 は I/O デバイス 90 にそれぞれデータを書き出し、データをバックアップする。

この書き出しは、前述した図 5 のシーケンスによって行う。

【0040】

図 9 は、マルチアクセス制御装置が 2 本の SCSI を制御する他の実施例の構成を示す。

この実施例では、一つのマルチアクセス制御装置から 2 本の SCSI を制御し、一方を送信専用とし、他方を受信専用にしている。

【0041】

図において、SCSI コントローラ 511 は送信専用

device are unified.

With namely, unification, control unit (SCSI controller) inside I/O device shoulder doesto substitute I/O device control unit 510, therefore, necessity to provide I/O device control unit 510 which is shown in Figure 2 is gone, means to take treatment system whichdirectly transfers control block to I/O control unit 700 inside I/O device .

[0037]

Figure 7 is figure which shows configuration of other Working Example when changeover to preparatory information processing apparatus is done from current system information processing apparatus .

When treatment is executed, through multi access control device 50, it takes over the current system information processing apparatus 21, 22, inside I/O device 70 of option and it houses treats information 71.

When and, fault occurs in current system information processing apparatus 21, 22, preparatory information processing apparatus 23 takingover information 71 inside I/O device 70 reading *, continues treatment.

[0038]

As for Figure 8 , information processing apparatus in figure which shows configuration of theother Working Example when backup it does, as for each information processing apparatus configuration whichhas local I/O device is taken with I/O device .

[0039]

As for each information processing apparatus 20, 30, 40, as data is written out in respective local I/O device 201, 301, 401, as for information processing apparatus 20, in for example I/O device 70, as for information processing apparatus 30 in I/O device 80,information processing apparatus 40 it writes out data respectively in I/O device 90, data backup does.

It writes out this , with sequence of Figure 5 which is mentionedearlier.

[0040]

Figure 9 shows configuration of other Working Example where multi access control device controls SCSI of 2.

With this Working Example , it controls SCSI of 2 from multi access control device of the one , on one hand makes transmission dedicated , designates other as reception dedicated .

[0041]

In figure, as for SCSI controller 511 with transmission

であり、SCSI コントローラ 512 は受信専用である。

そして、I/O デバイス 70 への書き込みは SCSI コントローラ 511 を用い、I/O デバイス 70 からの読みだしは SCSI コントロ-ラ 512 を用いる。

ただし、I/O デバイスに対するコマンドは送信受信にかかわらず全て SCSI コントローラ 511 で行う。

【0042】

本実施例の方式は、I/O デバイスが 1 台の場合に特に効果的である。

つまり、デバイスが 1 台に特定できるので、アービトレーション、セレクションを最初の 1 回のみ行い、その後のアクセス時にはアービトレーション、セレクションを省略することが出来る。

従って、SCSI のフェーズ遷移でコマンドコンプリート送信後、バスフリーすることなく、再びコマンドフェーズにすることができるので、高速なデータアクセスが可能となる。

【0043】

なお、本実施例は上記したものの他に、ブロードキャスト機能を用いることによって、複数の I/O デバイスに同一のデータを配布するように構成することができ、またネットワーク、インタフェースは上記した FDDI,SCSI に限定されず、他のネットワーク、インタフェースであってもよい。

【0044】

【発明の効果】

以上、説明したように、請求項 1 記載の発明によれば、ネットワーク制御手段と I/O デバイス制御手段とプロトコル変換手段からなるマルチアクセス制御手段を設けているので、I/O デバイスを変更することなく、複数の情報処理装置から複数の I/O デバイスへのアクセスが可能になる。

【0045】

請求項 2 記載の発明によれば、I/O デバイス制御部と I/O デバイス内の SCSI コントロ-ラとを共用化しているので、装置構成を簡単化できる。

【0046】

請求項 3 記載の発明によれば、複数の情報処理装置が実行した処理データを I/O デバイスに格納しているので、障害発生時に高速に予備切替を行うことができる。

dedicated , as for SCSI controller 512 it is a reception dedicated .

And, it starts reading writing to I/O device 70 from I/O device 70 makinguse of SCSI controller 511, SCSI controller 512 uses.

However, command for I/O device does with all SCSI controller 511 regardless oftransmit receive.

[0042]

system of this working example , when I/O device 1 is, is especially effective .

In other words, because specific is possible device to 1, only the initial one time does arbitration , selection , after that it is possible at time of the access to abbreviate arbitration , selection .

Therefore, after [komandokonpuriito] transmitting, without BASF Lee doing with the phase transition of SCSI , because again it can make command phase , high speed data access becomes possible.

[0043]

Furthermore, this working example can do in order by fact that for other thanthose which were inscribed, broad cast function is used, distribution fabric to do same data to I/O device of plural , configuration , inaddition network , interface is not limited in FDDI, SCSI which was inscribed, isgood even with other network , interface .

[0044]

[Effects of the Invention]

As above, explained, according to invention which is stated in the Claim 1, because multi access control means which consists of network control means and I/O device control means and protocol conversion means is provided, from information processing apparatus of plural access to the I/O device of plural becomes possible without modifying I/O device .

[0045]

According to invention which is stated in Claim 2, because the SCSI controller inside I/O device control unit and I/O device is converted commonly, equipment configuration can be simplified.

[0046]

According to invention which is stated in Claim 3, because thetreatment data which information processing apparatus of plural executed is housed in the I/O device , it is possible at time of damage to do preparatory changeover in the high

【0047】

請求項4記載の発明によれば、バックアップデータを一元管理することができ、特にDATの如き着脱可能なI/Oデバイスを用いた場合、I/Oデバイス毎にバックアップする情報処理装置を特定することにより、メディア管理が容易になる。

【0048】

請求項5記載の発明によれば、SCSIを送信インタフェースと受信インタフェースに分離しているので、高スループットのI/Oデバイスアクセスを実現することができる。

【図面の簡単な説明】

【図1】

本発明の一実施例に係るシステム構成図である。

【図2】

マルチアクセス制御装置のブロック構成図である。

【図3】

情報処理装置からマルチアクセス制御装置への制御フレームのフォーマットを示す図である。

【図4】

マルチアクセス制御装置から情報処理装置への終了フレームのフォーマットを示す図である。

【図5】

情報処理装置からマルチアクセス制御装置へのアクセスシーケンスを示す。

【図6】

マルチアクセス制御装置とI/Oデバイスを一体化させた場合の他の実施例の構成である。

【図7】

現用系情報処理装置から予備系情報処理装置への切替えを行う場合の他の実施例の構成を示す図である。

【図8】

情報処理装置をI/Oデバイスによってバックアップする場合の他の実施例の構成を示す図である。

speed .

[0047]

According to invention which is stated in Claim 4, it ispossible to manage backup data monistically, when demountable I/O device like theespecially DAT is used, media management becomes easy by specificdoing information processing apparatus which backup is done in every I/O device .

[0048]

According to invention which is stated in Claim 5, because the SCSI is separated into transmission interface and reception interface , I/O device access of high throughput can be actualized.

[Brief Explanation of the Drawing(s)]

[Figure 1]

It is a system diagram which relates to one Working Example of this invention .

[Figure 2]

It is a block diagram of multi access control device .

[Figure 3]

It is a figure which shows format of control frame to multi access control device from information processing apparatus .

[Figure 4]

It is a figure which shows format of end frame to information processing apparatus from multi access control device .

[Figure 5]

access sequence to multi access control device is shown from information processing apparatus .

[Figure 6]

It is a configuration of other Working Example when multi access control device and I/O device areunified.

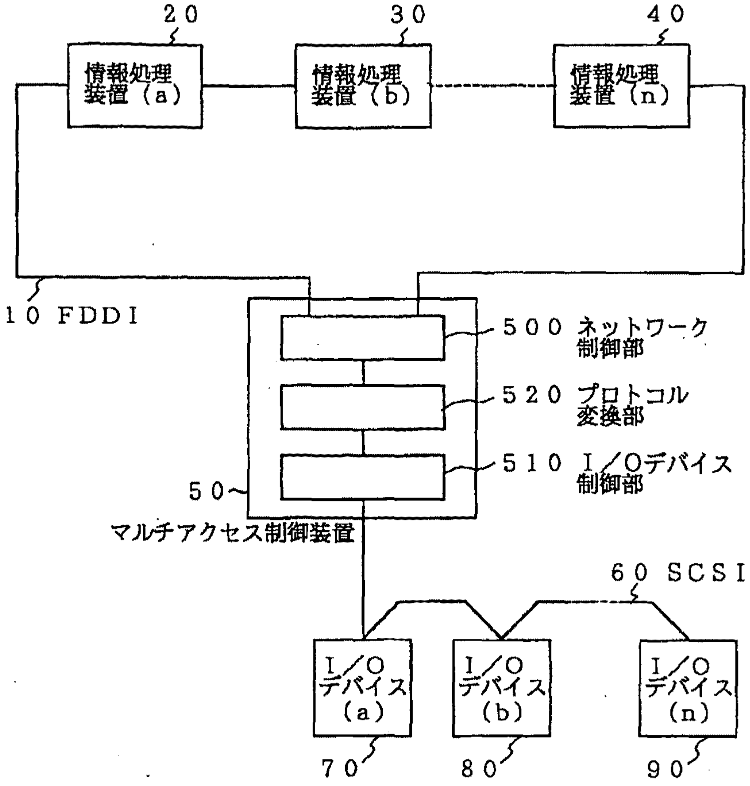
[Figure 7]

It is a figure which shows configuration of other Working Example when changeover to preparatory information processing apparatus is done from current system information processing apparatus .

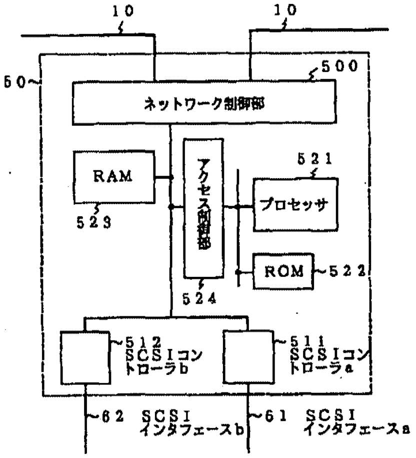
[Figure 8]

information processing apparatus it is a figure which shows configuration of other Working Example when backup it does with I/O device .

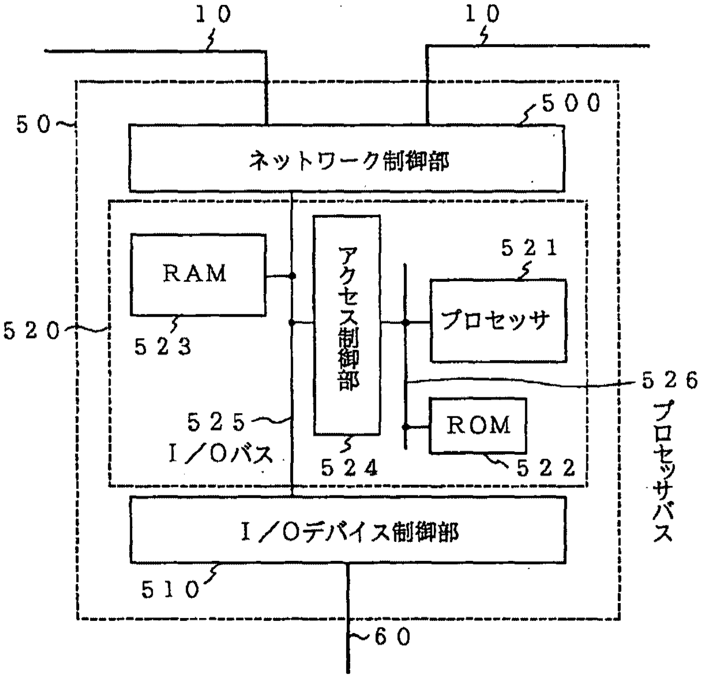
【図9】	[Figure 9]
マルチアクセス制御装置が2本のSCSIを制御する他の実施例の構成を示す図である。	It is a figure which shows configuration of other Working Example where the multi access control device controls SCSI of 2.
【符号の説明】	[Explanation of Symbols in Drawings]
10	10
FDDI	FDDI
20	20
情報処理装置	information processing apparatus
30	30
情報処理装置	information processing apparatus
40	40
情報処理装置	information processing apparatus
50	50
マルチアクセス制御装置	multi access control device
500	500
ネットワーク制御部	network control unit
510	510
I/O デバイス制御部	I/O device control unit
520	520
プロトコル変換部	protocol conversion section
60	60
SCSI	SCSI
70	70
I/O デバイス	I/O device
80	80
I/O デバイス	I/O device
90	90
I/O デバイス	I/O device
Drawings	
【図1】	[Figure 1]


20
情報処理装置（a）
30
情報処理装置（b）
40
情報処理装置（n）
10 FDDI
500 ネットワーク制御部
520 プロトコル変換部
510 I／Oデバイス制御部
50
マルチアクセス制御装置
60 SCSI
I／Oデバイス（a）
I／Oデバイス（b）
I／Oデバイス（n）
70
80
90

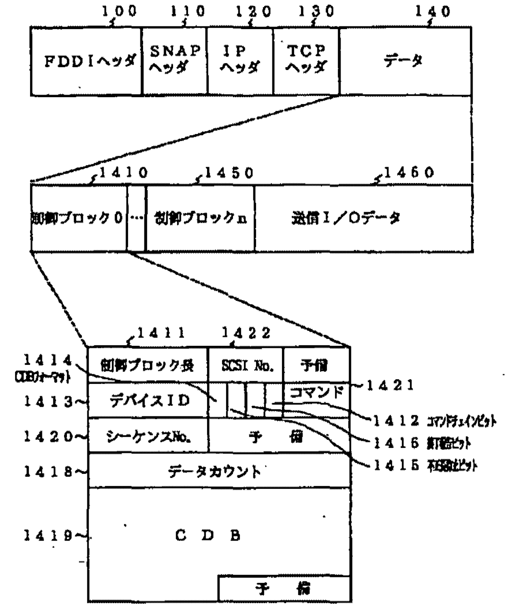
【図9】 [Figure 9]



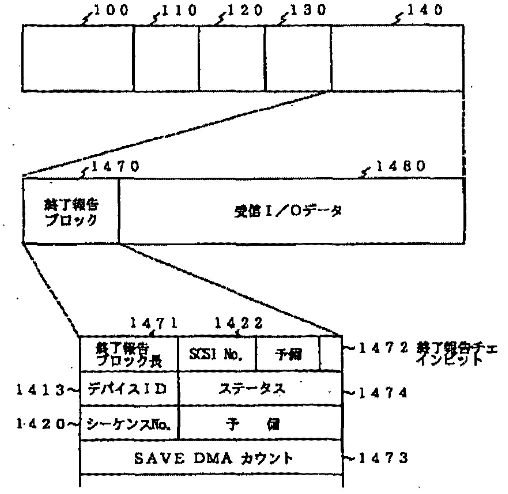
【図2】 [Figure 2]



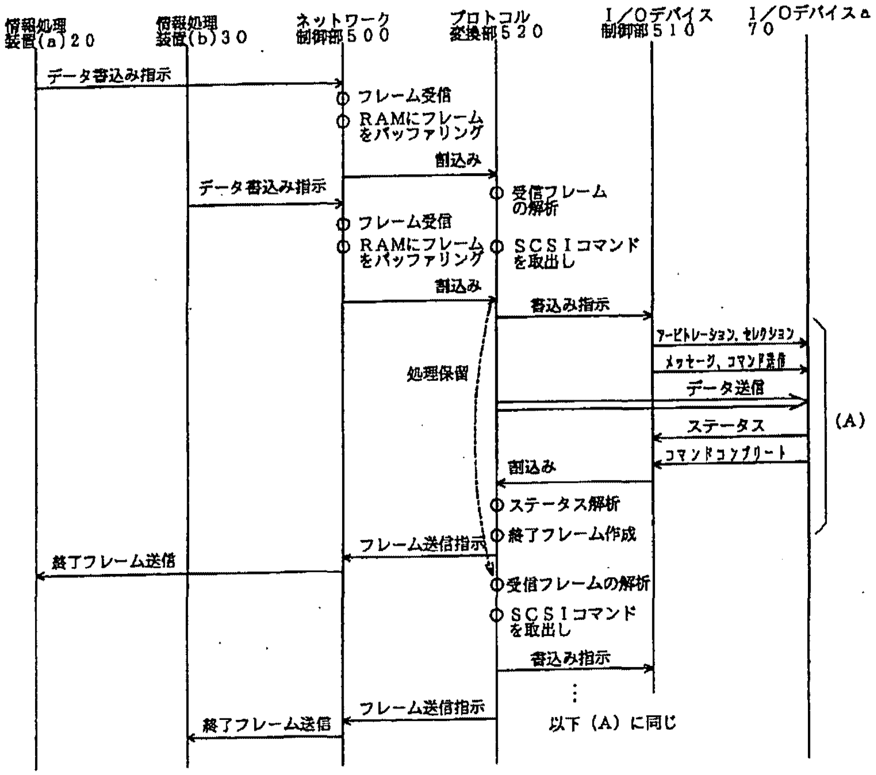
【図3】 [Figure 3]



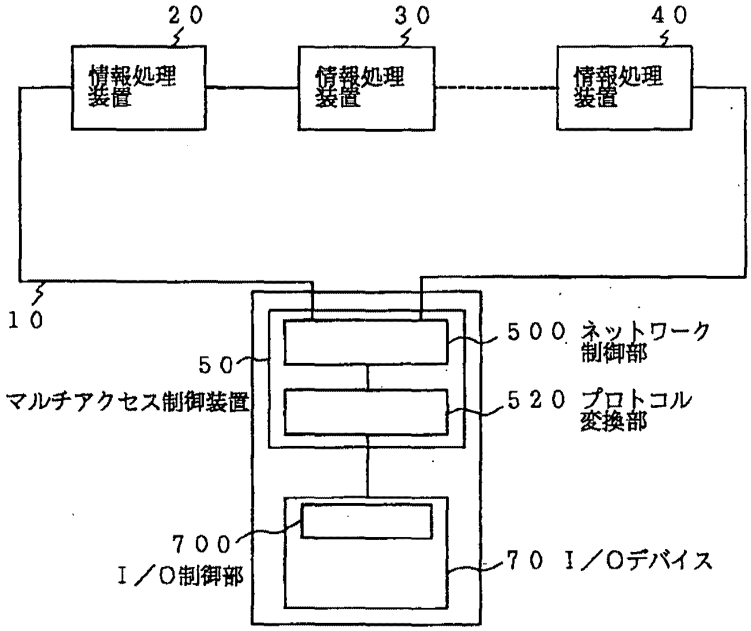
【図4】 [Figure 4]



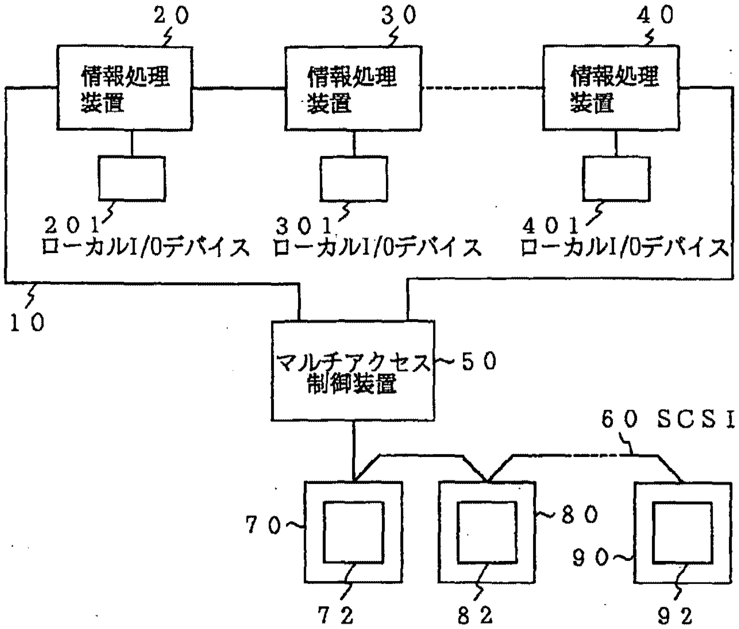
【図5】 [Figure 5]


情報処理
装置(a)20
情報処理
装置(b)30
ネットワーク
制御部500
プロトコル
変換部520
I／Oデバイス
制御部510
I／Oデバイスa
70
データ書込み指示
フレーム受信
RAMにフレーム
をバッファリング
割込み
データ書込み指示
受信フレーム
の解析
フレーム受信
RAMにフレーム
をバッファリング
SCSIコマンド
を取出し
割込み
書込み指示
アービトレーション，セレクション
メッセージ，コマンド送信
処理保留
データ送信
ステータス
(A)
コマンドコンプリート
割込み
ステータス解析
終了フレーム作成
フレーム送信指示
終了フレーム送信
受信フレームの解析
SCSIコマンド
を取出し
書込み指示
フレーム送信指示
以下（A）に同じ
終了フレーム送信

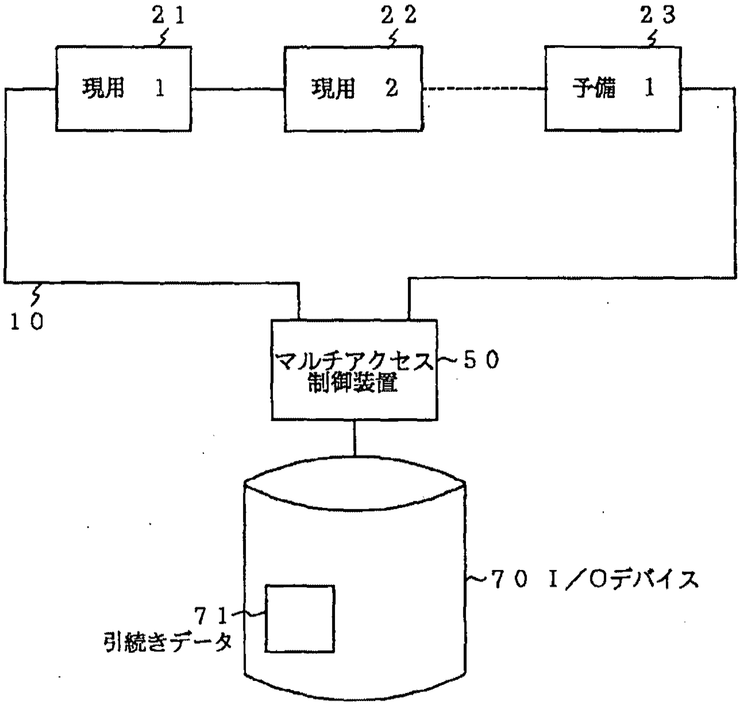
【図6】 [Figure 6]


２０
３０
４０
情報処理装置
情報処理装置
情報処理装置
１０
５０
マルチアクセス制御装置
５００ ネットワーク制御部
５２０ プロトコル変換部
７００
I／O制御部
７０ I／Oデバイス

【図8】 [Figure 8]


20
情報処理
装置
30
情報処理
装置
40
情報処理
装置
201
ローカルI/0デバイス
301
ローカルI/0デバイス
401
ローカルI/0デバイス
10
マルチアクセス
制御装置
～50
60 SCSI
70～
～80
90～
72
82
92

【図7】 [Figure 7]


２１
現用　1
２２
現用　2
２３
予備　1
１０
マルチアクセス制御装置
～５０
～７０　I／Oデバイス
７１
引続きデータ

B9

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 12/00

A1

(11) International Publication Number: WO 98/36357

(43) International Publication Date: 20 August 1998 (20.08.98)

(21) International Application Number: PCT/US98/02131

(22) International Filing Date: 5 February 1998 (05.02.98)

(30) Priority Data:
08/796,085 5 February 1997 (05.02.97) US

(71) Applicant: TRANSWITCH CORPORATION [US/US]; 3 Enterprise Drive, Shelton, CT 06484 (US).

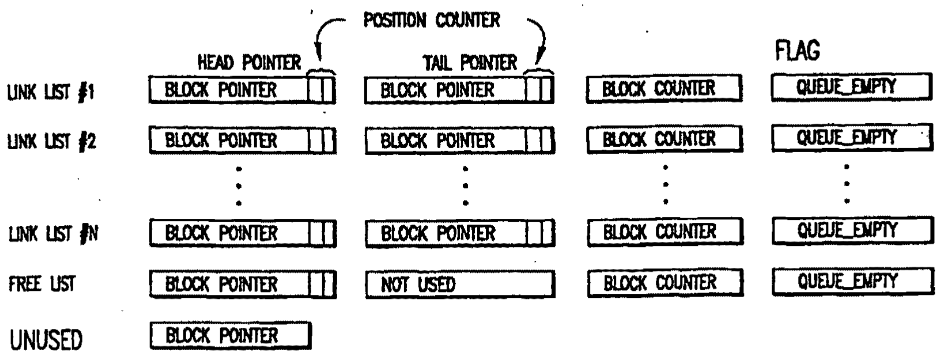
(72) Inventors: LAU, Joseph, C.; 1F, 29 Bamboo Road III, Science-based Industry Park, Hsinchu (TW). ROY, Subhash, C.; Apartment 3A, 905 Mix Avenue, Hamden, CT 06514 (US). CALLAERTS, Dirk, L., M.; Hoevestraat 13, B-3110 Rotselaar (BE). VANDEWEERD, Ivo, Edmond, Nicole; Vuurkruisenlaan 1, B-3500 Hasselt (BE).

(74) Agent: GORDON, David, P.; 65 Woods End Road, Stamford, CT 06905 (US).

(81) Designated States: CA, CN, IL, JP, MX, NO, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: SHARED MEMORY CONTROL USING MULTIPLE LINKED LISTS WITH POINTERS, STATUS FLAGS, MEMORY BLOCK COUNTERS AND PARITY



(57) Abstract

Apparatus and methods for allocating shared memory utilizing linked lists (LLs) use a management RAM which controls the flow of data to/from a shared memory (RAM), and stores information regarding a number of LLs and a free link list (FLL) in the RAM, and a block pointer to unused RAM locations. A head pointer (HP), tail pointer (TP), block counter and empty flag (EF) are stored for each data link list. The HP and TP each include a block pointer and a position counter. The block counter contains the number of blocks used in the particular queue. An EF indicates an empty queue. The FLL includes a HP, a block counter, and an EF. Each page of RAM receiving the incoming data includes locations for storing data. The last location of the last page in a block stores a next-block pointer plus parity information, and in the last block of a queue, is set to all ones. An independent agent used in the background monitors the integrity of the LL structure.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav	TM	Turkmenistan
BF	Burkina Faso	GR	Greece		Republic of Macedonia	TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's	NZ	New Zealand		
CM	Cameroon		Republic of Korea	PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

SHARED MEMORY CONTROL USING MULTIPLE LINKED LISTS WITH POINTERS, STATUS FLAGS, MEMORY BLOCK COUNTERS AND PARITY

This application is related to co-owned U.S. Serial No. 08/650,910, filed May 17, 1996, which is hereby incorporated by reference herein in its entirety.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to memory management. More particularly, the present invention relates to apparatus and methods of managing a plurality of data queues stored in linked lists in a shared common memory. The invention has particular application to the use of a very large scale integrated circuit (VLSI) for the buffering of telecommunications information such as ATM data, although it is not limited thereto.

2. State of the Art

In high speed communication networks, the management of buffer resources is one mechanism of increasing network performance. One group of methods of managing buffer resources is known as sharing, where a single RAM is simultaneously utilized as a buffer by a plurality of different channels. Various sharing methods are known (see Velamuri, R. et al., "A Multi-Queue Flexible Buffer Manager Architecture", IEEE Document No. 0-7803-0917-0/93) and each has inherent advantages coupled with inherent disadvantages in terms of blocking probability, utilization, throughput, and delay. What is common to all sharing methods, however, is that a mechanism is required to direct data into appropriate locations in the RAM in a desired order so that the data can be retrieved from the RAM appropriately. One such mechanism which is well known is the use of link lists which are used to manage multiple queues sharing a common memory buffer. Typically, a link list comprises bytes of data, where each byte has at least one pointer (forward and/or backward) attached to it, thereby identifying the location of the next byte of data in the queue. The link list typically includes extensive initialization and self-check procedures which are carried out by a microprocessor on a non-real-time basis. Thus, the use of standard prior art link list structures to manage multiplex queues sharing a common memory is not readily adaptable for VLSI implementation, and is likewise not particularly suited to the handling of very high speed telecommunications information where processing and handling are dictated by the data rate of the real-time telecommunications signal.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an apparatus and method for control of memory allocation.

It is another object of the invention to provide a new link list structure for managing queues in a shared memory.

It is a further object of the invention to provide a single VLSI which utilizes a link list structure for managing queues of high speed real time data in a shared memory.

It is an additional object of the invention to provide a link list apparatus and method for controlling the flow of Asynchronous Transfer Mode (ATM) telecommunications data into and out of a shared buffer.

Another object of the invention is to provide an apparatus and method for VLSI control of ATM data into and out of a shared RAM by utilizing a separate RAM containing information related to the plurality of link lists in the shared RAM.

In accord with the objects of the invention a management RAM contained within a VLSI is provided for controlling the flow of data into and out of a shared memory (data RAM). The management RAM is preferably structured as an x by y bit RAM which stores information regarding y-2 data link lists in the shared RAM, a free link list in the shared RAM, and a block pointer to unused shared RAM locations. Information stored in the x bits for each data link list includes a head pointer, a tail pointer, a block counter and an empty flag. In a preferred embodiment particularly applicable to the control of ATM data, the head and tail pointers are each composed of a block pointer and a position counter, with the position counter indicating a specific page in a block which is made up of a set of contiguous pages of memory, and the block pointer pointing to the block number. Regardless of how constituted, the head pointer contains the address of the first word of the first memory page of the link list, and the tail pointer preferably contains the address of the first word of the last memory page in the link list. The block counter contains the number of blocks used in the particular queue, and has a non-zero value if at least one page is used in the queue. The empty flag indicates whether the queue is empty such that the content of the link list should be ignored if the queue-empty flag indicates that the queue is empty.

Information stored in the management RAM for the free link list includes a head pointer, a block counter, and an empty flag, but does not need to include a tail pointer as free blocks are added to the top of the free list according to the preferred embodiment of the invention. As is discussed below in more detail, as data from different channels is directed into blocks of the data RAM, a link list is kept for each channel. As data is read out of the data RAM, blocks become available to receive new data. It is these freed blocks which are added to the free list. Block space can be assigned from the free list before or after the unused blocks (discussed below) are used.

To avoid excessive initialization requirements, an unused-block pointer is provided in the management RAM, as discussed above, and provides a pointer to the next unused block in memory. Initially all link lists, including the free list, are empty, and the unused block pointer is set to the number of blocks in the memory. As data is written to a block of shared RAM memory, the unused block pointer is decremented. When the unused block pointer equals zero, all of the cell blocks are included in the link lists (including the free link list).

According to a preferred aspect of the invention, each memory page of the shared data RAM receiving the incoming data (which RAM is managed by the management RAM) is composed of M contiguous memory addresses. Depending on the memory type, each address location can be of size B bits. The most common sizes are eight bits (byte), sixteen bits (word), thirty-two bits, and sixty-four bits. The first M-1 locations in the page are used to store data. The last (M'th) location of the last page in the block preferably is used to store the address of the first location of the next block of the queue plus an odd parity bit; i.e., the M'th location of the last page in the block stores a next block pointer plus parity information. If there are no more blocks in the queue, the M'th location in the last page is set to all ones.

According to another aspect of the invention, an independent agent is utilized in the background to monitor the integrity of the link list structure. The independent agent monitors the sum of the count of all of the link list block counters plus the unused blocks to ensure that it equals the total number of memory blocks in the common RAM. If not, an error is declared. Likewise, the independent agent checks each link list stored in the management RAM for the following error conditions: head and tail pointers are equal and the block counter is not of value one; head and tail pointers are different and the block counter is one; and, block counter equals zero. If desired, the independent agent can also monitor the block pointers stored in the M'th location of the last page of each block to determine parity errors and/or to determine errors using parity or CRC.

Using the methods and apparatus of the invention, four operations are defined for ATM cell management: cell write, cell read, queue clear, and link list monitoring. In the cell write operation, a cell is stored into a queue. More particularly, when an ATM cell is received at a port w so that it is to be stored in queue number n (which stores cells of priority v for port w), a determination is first made as to whether the queue is empty. If it is not empty, the queue status (i.e., the tail pointer and position counter stored in management RAM) is obtained, and a determination is made as to whether a new block will be needed to be added to the queue. If a new block is not required, the cell is written to the location indicated by the tail pointer position, and the tail pointer position counter for that queue in the management RAM is updated. If this is the last page of a block, the M'th location of the page (in the shared memory) is set to all ones. If a new block is required, either because the queue was empty or because a previous cell had been written into the last page of a block, a block must be obtained. If it is a first block of a queue, initial queue parameters are stored. If it is not the first block of the link list, a block is obtained from the free list and the free list is updated; or the block is obtained from the unused blocks and the block pointer for the unused blocks is updated. Then, the cell is written to the queue, and the tail pointer, position counter, and block counter for the queue are all updated in the management RAM.

The cell read operation is utilized where a cell is to be read from a queue. In the cell read operation, the cell indicated by the head pointer and head pointer position counter for that queue is read from the queue. After reading the cell from the queue a determination is made as to whether the cell was either the last cell in a block and/or the last cell in the queue. If it is neither, then the queue status is updated (i.e., the head pointer position counter is changed), and another cell read operation is awaited. If the cell is the last cell in the block, then the queue status preferably is checked for correctness by verifying the parity of the pointer (using a parity bit), and is updated by changing the head pointer and head pointer position counter. The free list is updated by adding the freed block to the head of the free list, and the free list and link list block counters are updated. If the cell is the last cell in the queue, the procedure for the last cell in the block is followed, and the queue empty flag is set.

The queue clear operation is a microprocessor command provided for the purpose of clearing a queue. When the queue clear operation is presented, the queue status is updated by setting the queue flag, and the blocks in the queue are added to the head of the free list which is likewise updated.

The link list monitoring operation is the agent which monitors the integrity of the link list structure whenever the cell write, cell read, and queue clear operations are not running. As set forth above, the link list monitoring operation monitors the linked lists for errors by checking that the sum of the count of all of the link list block counters plus the unused blocks equals the total number of memory blocks in the common RAM, that when head and tail pointers are equal the block counter is set to one, that when head and tail pointers are different the block counter is not set to one, etc.

Additional objects and advantages of the invention will become apparent to those skilled in the art upon reference to the detailed description taken in conjunction with the provided figures.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of an apparatus incorporating the link list memory management RAM of the invention.

Figure 2 is a chart showing the structure of the memory management RAM of Figure 1.

Figure 3a is a diagram of an example of the shared data memory of the apparatus of Figure 1.

Figure 3b is a diagram of the details of a page of one of the blocks shown in Figure 3a.

Figure 3c is a diagram of an example of the information contained in the memory management RAM of Fig. 1 for managing the shared data memory example of Figure 3a.

Figures 4a - 4d are flow charts for the write, read, queue clear, and link list monitoring operations carried out by the flow controller of the apparatus of Figure 1.

Figures 5a-5d are state machine diagrams for a write, read, clear, and monitor state machine according to the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The invention will now be described with reference to the physical layer VLSI portion of an ATM destination switch described in parent U.S. Serial No. 08/650,910, although it is not limited thereto. As seen in Fig. 1, and as discussed in the parent application, the physical layer portion 130 of the ATM destination switch 100 preferably includes a UTOPIA interface 150, a managing RAM 162, a flow controller 166, a microprocessor interface 167, channel interface buffers 170, and a RAM interface 175. The flow controller 166 is coupled to the UTOPIA interface 160, the managing RAM 162, the microprocessor interface 167, the channel interface buffers 170, and the RAM interface 175. The UTOPIA interface generally receives cells of ATM data in a byte-wide format, and passes them to the flow controller 166. Based on the destination of the cell (as discussed in the parent application), and the priority of the cell, the flow controller 166 writes the cell into an appropriate output buffer 170. The output buffer is preferably capable of storing at least two ATM cells so that one cell can be read out of the buffer as another is being read into the buffer without conflict. If buffer space is not available for a particular cell at a particular time, the flow controller 166 forwards the ATM cell via the RAM interface 175 to a desired location in a shared RAM 180 (which may be on or off chip) based on information contained in the managing RAM 162 as discussed in more detail below. When room becomes available in the output buffer 170 for the cell, the flow controller 166 reads the data out of the shared RAM 180, and places it in the buffer 170. In the background, when not receiving data from the UTOPIA interface, and when not reading data from or writing data to the shared RAM 180 or writing data to the buffers, the flow controller 166 monitors the integrity of the link list structure contained in the managing RAM, as is described in more detail below. In addition, the flow controller 166 can perform various functions in response to microprocessor command received via the microprocessor interface 167.

The managing RAM 162 may serve various functions, including providing information for assisting in the processing of the header of the ATM cell as discussed in the parent application hereto. For purposes of this invention, however, the managing RAM 162, or at least a portion thereof, is preferably provided as a x bit by y word RAM for the purpose of managing y-2 link lists which are set up in the shared RAM 180 (y-2 equalling the product of w ports times v priorities). Thus, as seen in Fig. 2, a link list information structure for y-2 data queues includes: a head pointer, a tail pointer, a block counter, and a queue empty flag for each of the y-2 data queues; a free list block pointer, block counter, and queue empty flag for a free list; and a block pointer for the unused blocks of memory. Each head pointer and tail pointer preferably includes a block pointer and a position counter, with the block pointer used for pointing to a block in the memory, and the position counter being used to track pages within a block of memory. Thus, for example, where ATM cells of fifty-three bytes of data are to be stored in the shared memory, and each cell is to be stored on a "page", a block having four contiguous pages may be arranged with the position counter being a two bit counter for referencing the page of a block. The block counter for each queue is used to reference the number of blocks contained within the queue. The queue empty flag when set indicates that the queue is empty, and that the pointers contained within the queue as well as the block count can be ignored.

As suggested above, the head pointer for each link list queue contains the address of the first word of the first memory page of the queue in the shared memory. The tail pointer for each link list queue contains the address of the first word of the last memory page in the queue. Each memory page of the shared memory is composed of M contiguous memory addresses. Depending on the memory type, each address location can be of size B bits, with common sizes being eight bits (byte), sixteen bits (word), thirty-two bits, or sixty-four bits. In accord with the preferred embodiment of the invention, the address locations are sixteen bits in length with the first M-1 locations in a page containing the stored information. The M'th location of a last page in a block is used to store a next block pointer which is set to the first location of the next block plus an odd parity bit. Where the block is the last block in the queue, the M'th location of the last page in the last block is set to all ones. Where the page is neither the last page of the block, nor the last block in the queue, the M'th location of the page is not utilized. In the preferred embodiment of the invention used with respect to ATM telecommunications data, each page is thirty-two words in length (i.e., M = 32), with each word being sixteen bits. Thus, an ATM cell of fifty-three bytes can be stored on a single page with room to spare. It should be appreciated, that in some applications, only the data payload portion of the ATM cell (i.e., forty-eight bytes), and not the overhead portion (five bytes) will be stored in the shared memory. In other applications, such as in switches where routing information is added, cells of more than fifty-three bytes may be stored. Regardless, with a thirty-two word page, system addressing is simplified.

An example of the memory organization of the shared memory is seen in Fig. 3a. In Fig. 3a, two active link list data queues are represented, as well as a free list queue and an Unused block. In particular, memory blocks 512, 124, and 122 are shown linked together for a first queue, memory blocks 511, 125, and 123 are linked together for a second queue, memory blocks 510 - 125 are linked together for the free list queue, and memory blocks 121 - 1 are Unused. It will be appreciated that in the preferred embodiment of the invention, each page contains thirty-two sixteen bit words. Thus, the thirty-second (M'th) word of memory block 512 (seen in more detail in Fig. 3b) contains a pointer (the ten least significant bits) which points to memory block 124, the thirty-second word of memory block 124 contains a pointer which points to memory block 122, and the thirty-second word of memory block 122 contains all ones, thereby indicating the last word in the queue. Likewise, the thirty-second word of memory block 511 contains a pointer which points to memory block 125, the thirty-second word of memory block 125 contains a pointer which points to memory block 123, and the thirty-second word of memory block 123 contains all ones, thereby indicating the last word of that queue.

The free list of Fig. 3a is seen extending from block 510 to block 126. The unused blocks run from block 121 to block 1.

Turning to Fig. 3c, specifics are seen of the management RAM which would be associated with managing the shared memory in the state of Fig. 3a. In particular, information for link list #1 is seen with a head pointer having a block pointer having a value equal to 512 and a position counter set at "00" to indicate a first page of memory in the block storing data. The tail pointer of the link list #1 information has a block pointer having a value equal to 122 and a position counter set to "11" to indicate that all pages of block 122 are being used. The block counter of the information for link list #1 is set to a value of three, and the queue empty flag is not set (i.e., equals zero). Information for link list #2 is seen with a head pointer having a block pointer having a value equal to 511 and a position counter set at "01" to indicate that the data first occurs at a second page of the block (i.e., the first page already having been read from the block). The tail pointer of the link list #2 information has a block pointer having a value equal to 123 and a position counter set at "10" which indicates that there is no data in the last page of the block. The block counter of the link list #2 information is also set to a value of three, and the queue empty flag is not set. The value of the head and tail pointers and block count for the information of link list #N are not indicated, as the queue empty flag of link list #N is set (equals one), thereby indicating that the pointers and block counter do not store valid data. Likewise, while details of information for other link lists are not shown, the only data of interest would be that the queue empty flags related to all of those link lists would equal one to indicate that no valid data is being stored with reference to those link lists. The head pointer of the free list information has a block pointer set to a value 510, and a block count of 385. The queue empty flag of the free list is not set, as the free list contains data. Finally, the block pointer relating to the Unused queue is shown set to a value of 121. It is noted that in order to increase performance, the free list head pointer and block counter information is preferably implemented in a series of flip-flops, and is thus readily available for purposes discussed below with reference to Figs. 4a-4d. The queue empty flags are also preferably similarly implemented.

It should be appreciated that by providing the queue empty flags and an Unused block pointer, excessive initialization requirements are eliminated. As suggested above, the queue empty flag indicates that there is no valid data for a link list and that the head and tail pointers for that link list and the block counter of that link list can be ignored. The Unused block pointer is provided to point to the next unused block in shared memory. As memory pages are written or used, the Unused block pointer is decremented until a value of zero is reached. At that point, all cell blocks are included in the link lists (including the free list). As previously mentioned, when a block is read from the shared memory, the available block is added to the free list. When a new block is required for adding to a link list, the block space may be taken from either the free list or from the Unused blocks, and available blocks from the free list may be taken either before or after the Unused blocks are used.

Turning now to Figure 4a, a flow chart of operations of the flow controller 166 of the apparatus 100 of Figure 1 is seen with respect to writing data to the shared memory. It is noted that while the operations are shown in flow chart form, in accord with the preferred embodiment of the invention, the operations are carried out in hardware. When the flow controller 166 determines that it is receiving an ATM cell which cannot be written into a buffer directly, the flow controller makes a determination at 200 (by checking the management RAM queue empty flag associated with that queue) as to whether the queue which should receive that cell is empty. If the queue is not empty, at 202 the queue status (i.e., the tail pointer and position counter) for that queue is obtained, and at 204 a determination is made as to whether a new block will be needed to be added to the queue (i.e., is the position counter equal to "11"). If a new block is not required, at 206 the cell is written to the shared RAM location indicated by the tail pointer position counter for that queue (stored in management RAM), and at 208 the tail pointer position counter for that queue is updated. At 210, a determination is made as to whether the cell is being written into the last page of a block. If so, at 212 the flow controller writes a word of all ones into the M'th location of the page (in the shared memory).

If it is determined that a new block of shared RAM is required to store the incoming cell because at 200 the queue was empty, at 214, a block is obtained from the either the free list or from unused RAM. If the block is obtained from the free list, at 216, the free list information is updated by changing the head pointer of the free list (i.e., setting the head pointer to the value stored in the M'th location of the last page of the obtained block), and by decrementing the block counter associated with the free list. If the block is obtained from the unused RAM, the block pointer for the unused RAM is decremented at 216. Regardless, at 218, the cell is written to the queue, and at 220, the tail pointer and block counter for the queue are both updated in the management RAM (with the block counter being set to the value one), and the queue empty flag is changed.

If it is determined that a new block of shared RAM is required to store the incoming cell because at 204 the tail pointer position counter of the link list indicated that the entire tail block is storing data, at 222, a block is obtained from the either the free list or from unused RAM. If the block is obtained from the free list, at 224, the free list is updated by changing the head pointer of the free list (i.e., setting the head pointer to the value stored in the M'th location of the last page of the obtained block), and by decrementing the block counter associated with the free list. If the free list becomes empty because a block is removed, the queue empty flag of the free list is set. If the block is obtained from the unused RAM, the block pointer for the unused RAM is decremented at 224. Regardless, at 228, the cell is written to the queue, and at 230, the tail pointer and block counter for the queue are both updated in the management RAM.

The details of the flow controller operation with respect to a cell read operation (i.e., where a cell is to be read from a queue because a buffer is available to receive the cell) is seen in Fig. 4b. In particular, when a data buffer becomes available, the flow controller at 250 reads the head pointer and tail pointer in the management RAM for the link list associated with the available data buffer. Then, at 252, the flow controller reads from shared memory the cell at the location in the shared memory indicated by the head pointer, and provides the cell to the data buffer. After the data has been read, the flow controller determines at 254 (based on the head pointer and tail pointer) whether the cell was the last cell in the queue, and at 256 (based on the head pointer position counter) whether the cell was the last cell in a block. If it is neither, then at 258 the queue status is updated (i.e., the head pointer position counter is changed), and another cell read operation is awaited. If at 254 it is determined that the cell is the last cell in the queue, at 260, the head pointer for the free list (obtained from the management RAM) is inserted into the last word of the last page of the freed block. Then at 262, the free list in the management RAM is updated by adding the freed block to the head of the free list; i.e., by updating the free list block pointer and block counter. At 264, the queue empty flag is set for the link list which now has no blocks. If the free list was empty prior to adding the freed block, the free list must be initialized (with appropriate head pointer and block counter) and the queue empty flag changed at 264. In addition, in the case were the free list was empty prior to adding the freed block, the last word in the freed block in the shared RAM should be set to all ones.

If at 256 it is determined that the cell which has been read out of shared memory is the last in a block, then at 266, the head pointer for the free list as obtained from the management RAM is inserted into the last word of the last page of the freed block. Then, at 268, the queue status for the link list is updated by changing the block pointer and position counter of the head pointer (to the value contained in the last word of the page of memory being read out of the shared memory), and by decrementing the block counter. Again, it is noted that if the free list was empty prior to adding the freed block, the free list must be initialized (with appropriate head pointer and block counter) and the queue empty flag changed, and the last word in the freed block in the shared RAM should be set to all ones. It is also noted, that upon obtaining the pointer in the M'th location of the last page of the block, according to the preferred embodiment of the invention, at 270, a parity check is done on the pointer. At 272, the calculated parity value is compared to the parity bit stored along with the pointer. Based on the comparison, at 274, a parity error condition can be declared, and sent as an interrupt message via the microprocessor interface port 167 (Fig. 1) to the microprocessor (not shown). Preferably, when a parity error is found, the microprocessor treats the situation as a catastrophic error and reinitializes the management and data RAMs.

Figure 4c sets out the operation with respect to the queue clear microprocessor command (received via the microprocessor interface 167). When the queue clear operation is presented, at 270 the queue status for the link list is updated by setting the queue empty flag, and at 272, the blocks in the queue are added to the head of the free list which is updated in a manner discussed above (Fig. 4b) with reference to the cell read operation.

The link list monitoring operation seen in Fig. 4d is the hardware agent which monitors the integrity of the link list structure whenever the cell write, cell read, and queue clear operations are not running. The link list monitoring operation preferably monitors four different error conditions. In particular, at 280, the counts of all of the link list block counters (including the free list) where the queue empty flag for those link lists are not set are summed together with the unused blocks and compared the total number of memory blocks in the common RAM. If the sum does not equal the total number of memory blocks in the common RAM, at 281, an error condition is declared by triggering a microprocessor interrupt bit. At 282, the head and tail block pointers of each link list are compared. If at 284 the head and tail block pointers are determined to be equal, at 286 the block counter is checked, and if not equal to one, at 287 an error condition is declared. If the head and tail block pointers are not equal when compared at 284, at 288 the block counter is checked, and if the block count is equal to one, at 289 an error condition is declared. At 290, the block counter for each link list whose queue empty flag is not set is checked; and if the block counter equals zero, at 291 an error condition is declared.

According to the preferred embodiment of the invention, the write, read, clear, and monitoring operations of the flow controller are carried out in hardware which may be generated by using HDL code to synthesize hardware gates via use a VHDL compiler. Figures 5a-5d are state machines diagrams corresponding to the HDL code, including a write state machine (Fig. 5a), a read state machine (Fig. 5b), a clear state machine (Fig. 5c), and a monitoring state machine (Fig. 5d). The gates created using the code may be standard cell technology or gate array technology.

It should be appreciated that the invention is not intended to be limited to a strictly hardware implementation, but is also intended to apply to memory management utilizing a microprocessor with associated firmware (e.g., a ROM).

There have been described and illustrated herein an apparatus and method for management of shared memory. While particular embodiments of the invention have been described, it is not intended that the invention be limited thereto, as it is intended that the invention be as broad in scope as the art will allow and that the specification be read likewise. Thus, while the invention has been described with reference to VLSI implemented ATM equipment, it will be appreciated that the invention has broader applicability. Also, while specific details of RAM sizes, etc. have been disclosed, it will be appreciated that the details could be varied without deviating from the scope of the invention. For example, while a management of RAM of size x bits by y words has been described for managing y-2 link lists of data, it will be appreciated that the management RAM could assume different sizes. Thus, for example, instead of using a separate word for the unused block pointer, the unused block pointer could be located in the "tail pointer" location of the free list (which itself does not use a tail pointer), thereby providing a management RAM of x bits by y words for managing y-1 link lists of data. In addition, rather than providing the information related to the link lists with the head pointer, tail pointer, block counter, and queue empty flag in that order, the variables of the link list could be reordered. Similarly, instead of providing a shared memory having pages of thirty-two words in depth, each word being sixteen bits in length, it will be appreciated that memories of different lengths and depths could be utilized. Also, rather than locating the pointer to the next block in the last word of the last page of a previous block, it will be appreciated that the pointer could be located in a different location. Further yet, while specific flow charts have been disclosed with respect to various operations, it will be appreciated that various aspects of the operations can be conducted in different orders. In addition, while particular code has been disclosed for generating gate arrays which conduct the operations in hardware, it should be appreciated by those skilled in the art that other code can be utilized to generate hardware, and that hardware and/or firmware can be generated in different manners. Furthermore, while the invention was described with respect to separate RAMs for the management RAM and the shared data RAM, it will be appreciated that both memories may be part of a larger single memory means. It will therefore be appreciated by those skilled in the art that yet other modifications could be made to the provided invention without deviating from its spirit and scope as so claimed.

Claims:

1. Apparatus for managing the storage of data in a memory, comprising:

a) a shared memory means having a plurality of data storage locations;

b) control means for receiving said data and forwarding said data to desired of said plurality of data storage locations in said shared memory means, wherein said data is stored in said plurality of data storage locations in the form of a plurality of link lists, each link list having a head;

c) management memory means for storing information regarding each of said plurality of link lists, said information including a head pointer and a queue empty flag for each link list, said head pointer for each particular respective link list pointing to a location of a respective said head of that particular link list, and said queue empty flag for a link list indicating that that link list has no valid data contained therein.

2. An apparatus according to claim 1, wherein:

said control means reads data from said shared memory means,

at least a plurality of said data storage locations are in the form of a free link list, said free link list relating to data storage locations from which data has been read by said control means, and

said management memory means includes a pointer and a queue empty flag for said free link list.

3. An apparatus for managing the storage of data in a memory, comprising:

a) a shared memory means having a plurality of data storage locations;

b) control means for receiving said data and forwarding said data to desired of said plurality of data storage locations in said shared memory means, and for reading data from said shared memory means, wherein said data is stored in said plurality of data storage locations in the form of a plurality of link lists, each link list having a head;

c) management memory means for storing information regarding each of said plurality of link lists, said information including a head pointer for each link list queue, said head pointer for each particular respective link list pointing to a location of a respective said head of that particular link list,

wherein upon initialization, at least a plurality of said data storage locations of said shared memory means are unused, and after utilization, at least a plurality of said data storage locations are in the form of a free link list, said free link list relating to data storage locations from which data has been read by said control means, and

wherein said management memory means includes a pointer to at least one of said unused data storage locations, and said management memory means includes a pointer for said free link list.

4. An apparatus according to any preceding claim, wherein:

at least upon initialization, at least a plurality of said data storage locations of said shared memory means are unused, and

said management memory means includes a pointer to at least one of said unused data storage locations.

5. An apparatus according to any previous claim, wherein:

said shared memory means is arranged in a plurality of blocks with each block having a plurality of said data storage locations, and

said information stored in said management memory means regarding each of said plurality of link list queues includes a block counter for each of said plurality of link list queues, each block counter counting the number of blocks contained in that link list queue.

6. An apparatus according to claim 5, wherein:

each of said plurality of blocks is arranged as a plurality of contiguous pages with each page having a plurality of said data storage locations, and each said head pointer comprises a block pointer which points to a block and a page counter which points to a page in said block.

7. An apparatus according to claim 5, wherein:
each block storing data includes at least one location containing one of (i) a pointer to a next block in the link list, and (ii) an indicator which indicates that the block is the last block in the link list.

8. An apparatus according to claim 7, wherein:
said pointer to a next block in the link list includes a parity bit for said pointer.

9. An apparatus according to claim 6, wherein:
each block storing data includes at least one location in a last page of that block containing one of (i) a pointer to a next block in the link list, and (ii) an indicator which indicates that the block is the last block in the link list.

10. An apparatus according to any previous claim, wherein:
said information includes a tail pointer for each link list containing said data.

11. An apparatus according to claim 6, wherein:
said information includes a tail pointer for each link list containing said data,
each of said plurality of blocks is arranged as a plurality of contiguous pages with each page having a plurality of said data storage locations,
each said head pointer comprises a first block pointer which points to a block and a page counter which points to a page in said block, and
each said tail pointer comprises a second block pointer which points to a tail block and a page counter which points to a page in said tail block.

12. An apparatus according to claim 6, wherein:

said data comprises ATM data received in cell format, and each said page includes enough of said data storage locations to store all of the data contained in an ATM cell.

13. An apparatus according to claim 12, wherein:

each page includes thirty-two sixteen bit word locations.

14. An apparatus according to claim 5, wherein:

said control means reads data from said shared memory means,

at least a plurality of said data storage locations are in the form of a free link list, said free link list relating to data storage locations from which data has been read by said control means, and

said management memory means includes a pointer, a block counter, and a queue empty flag for said free link list,

at least a plurality of said data storage locations of said shared memory means are unused, and

said management memory means includes a pointer to said at least one of said unused data storage locations, and

said control means includes means for comparing a sum of counts of said block counters of each link list containing data, said free link list, and said unused pointer to the number of blocks in said shared memory means.

15. An apparatus according to claim 14, wherein:

said control means further comprises means for generating an error signal is said sum of counts does not equal said number of blocks in said shared memory means.

16. An apparatus according to claim 10, wherein:

said control means includes means for comparing, for each link list containing data, said tail pointer to said head pointer.

17. An apparatus according to claim 16, wherein:

said control means further comprises means for generating an error signal if said tail pointer and said head pointer for a link list containing data point to an identical block, and said block counter for said link list does not equal one.

18. An apparatus according to claim 16, wherein:

said control means further comprises means for generating an error signal if said tail pointer and said head pointer for a link list containing data point to different blocks, and said block counter for said link list equals one.

19. An apparatus according to claim 5, wherein:

said control means further comprises means for checking the count of each block counter of a link list where the queue empty flag is not set, and for generating an error signal if the count is zero and the queue empty flag is not set.

20. An apparatus according to any preceding claim, wherein:

said control means and said management memory means are contained on a single integrated circuit.

21. An apparatus according to claim 5, wherein:

said management memory means includes said pointer, a block counter, and a queue empty flag for said free link list, and

said control means includes means for comparing a sum of counts of said block counters of each link list containing data, said free link list, and said unused pointer to the number of blocks in said shared memory means, and means for generating an error signal is said sum of counts does not equal said number of blocks in said shared memory means.

22. An apparatus according to claim 10, wherein:

said control means includes means for comparing, for each link list containing data, said tail pointer to said head pointer, and means for generating an error signal if either

(i) said tail pointer and said head pointer for a link list containing data point to an identical block, and said block counter for said link list does not equal one, or

(ii) said tail pointer and said head pointer for a link list containing data point to different blocks, and said block counter for said link list equals one.

23. A method of managing the storage of data utilizing a controller, a shared memory having a plurality of data storage locations, and a management memory, said method comprising:

a) using said controller to forward received data to desired of the plurality of data storage locations in the shared memory, wherein the data is stored in the plurality of data storage locations in the form of a plurality of link lists, each link list having a head; and

b) storing information regarding each of the plurality of link lists in the management memory, said information including a head pointer and a queue empty flag for each link list, said head pointer for each particular respective link list pointing to a location of a respective said head of that particular link list, and said queue empty flag for a link list indicating that that link list has no valid data contained therein.

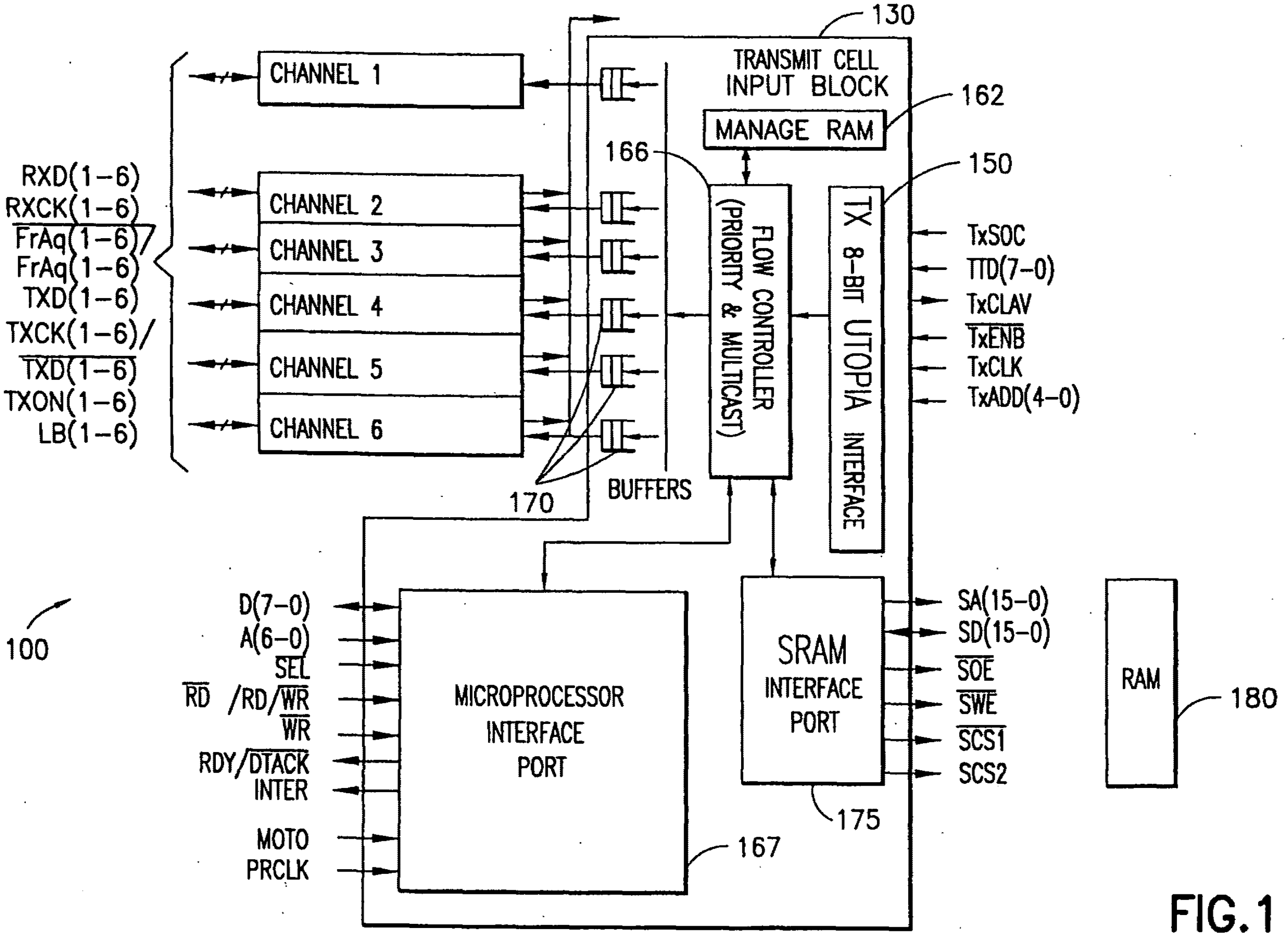

130
TRANSMIT CELL INPUT BLOCK
MANAGE RAM
162
166
150
CHANNEL 1
CHANNEL 2
CHANNEL 3
CHANNEL 4
CHANNEL 5
CHANNEL 6
RXD(1-6)
RXCK(1-6)
FrAq(1-6)/
FrAq(1-6)
TXD(1-6)
TXCK(1-6)/
TXD(1-6)
TXON(1-6)
LB(1-6)
FLOW CONTROLLER (PRIORITY & MULTICAST)
TX 8-BIT UTOPIA INTERFACE
TxSOC
TTD(7-0)
TxCLAV
TxENB
TxCLK
TxADD(4-0)
170
BUFFERS
100
D(7-0)
A(6-0)
SEL
RD /RD/WR
WR
RDY/DTACK
INTER
MOTO
PRCLK
MICROPROCESSOR INTERFACE PORT
167
SRAM INTERFACE PORT
175
SA(15-0)
SD(15-0)
SOE
SWE
SCS1
SCS2
RAM
180

FIG.1

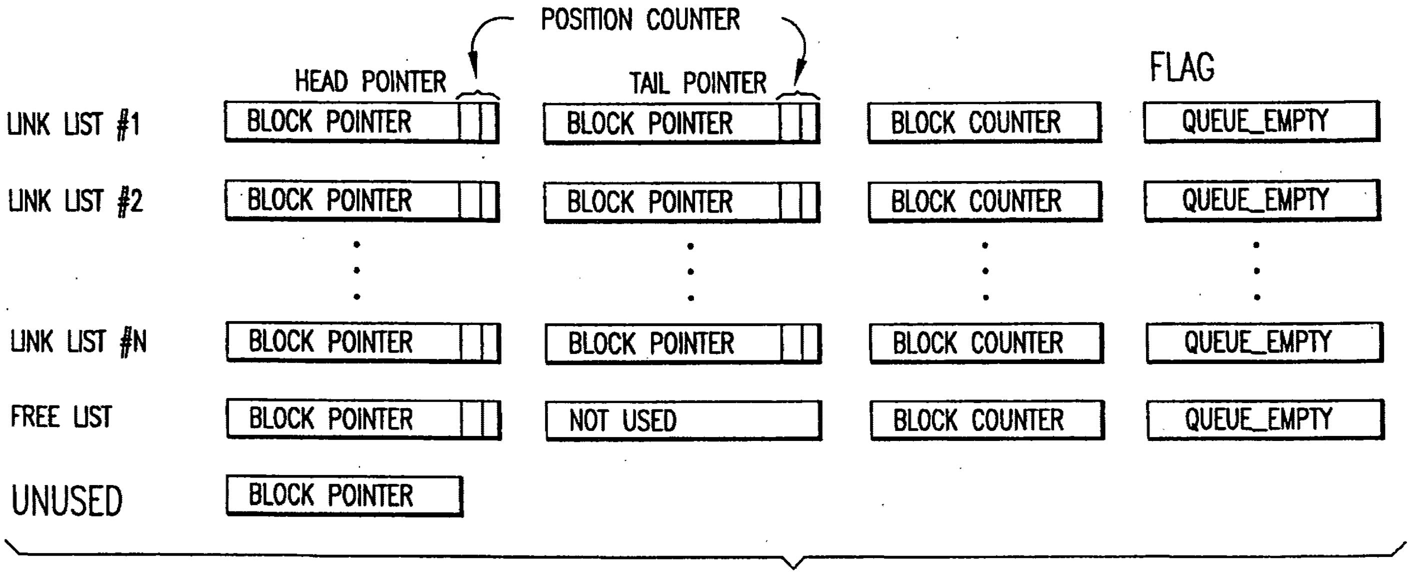

POSITION COUNTER
HEAD POINTER
TAIL POINTER
FLAG
LINK LIST #1
LINK LIST #2
LINK LIST #N
FREE LIST
UNUSED
BLOCK POINTER
BLOCK POINTER
BLOCK COUNTER
QUEUE_EMPTY
BLOCK POINTER
BLOCK POINTER
BLOCK COUNTER
QUEUE_EMPTY
BLOCK POINTER
BLOCK POINTER
BLOCK COUNTER
QUEUE_EMPTY
BLOCK POINTER
NOT USED
BLOCK COUNTER
QUEUE_EMPTY
BLOCK POINTER

FIG.2

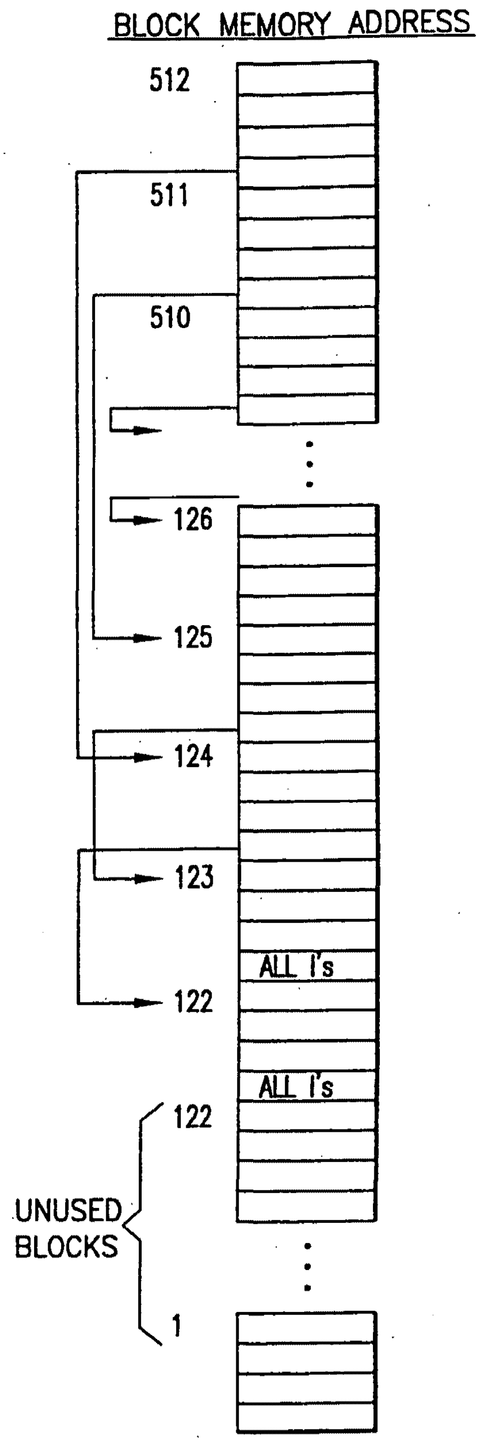

BLOCK MEMORY ADDRESS
512
511
510
126
125
124
123
ALL 1's
122
ALL 1's
122
UNUSED BLOCKS
1

FIG.3a

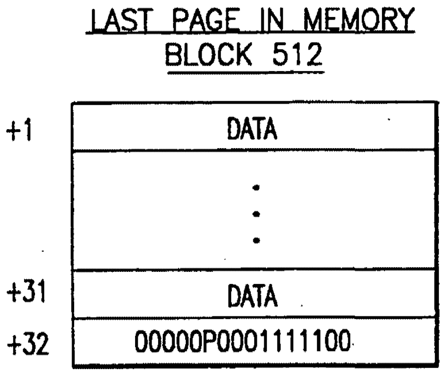
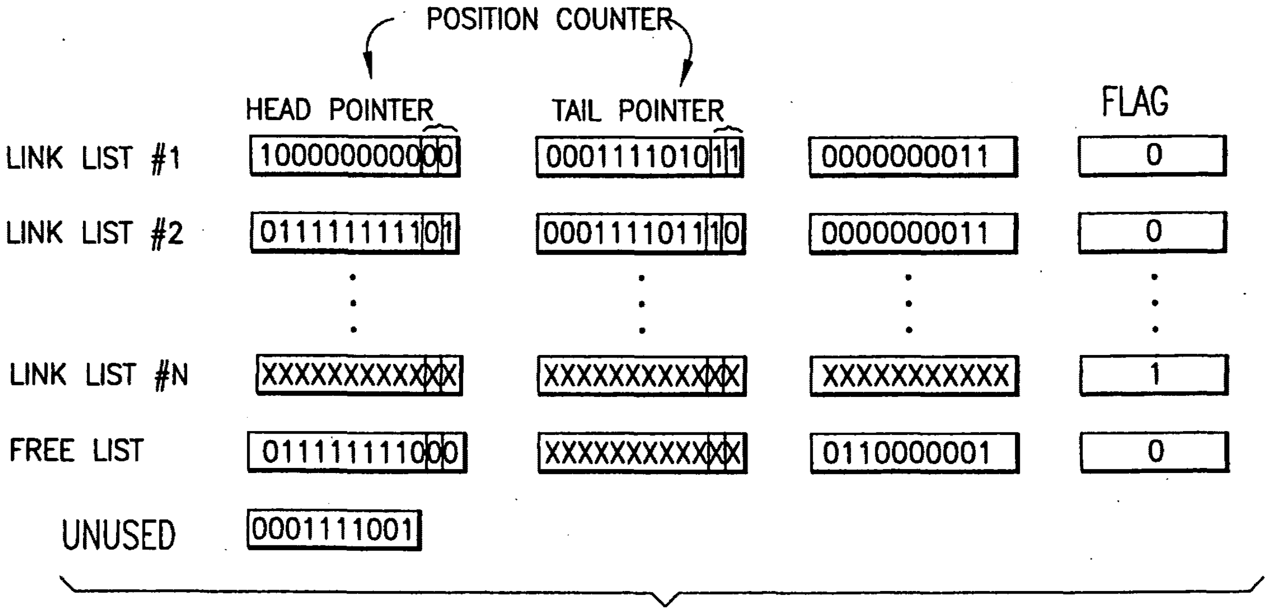

LAST PAGE IN MEMORY BLOCK 512
+1
DATA
+31
DATA
+32
00000P0001111100

FIG.3b



	HEAD POINTER	TAIL POINTER		FLAG
LINK LIST #1	100000000000	000111101011	0000000011	0
LINK LIST #2	011111111101	000111101110	0000000011	0
⋮	⋮	⋮	⋮	⋮
LINK LIST #N	XXXXXXXXXXXX	XXXXXXXXXXXX	XXXXXXXXXXX	1
FREE LIST	011111111000	XXXXXXXXXXXX	0110000001	0
UNUSED	0001111001			

FIG.3c

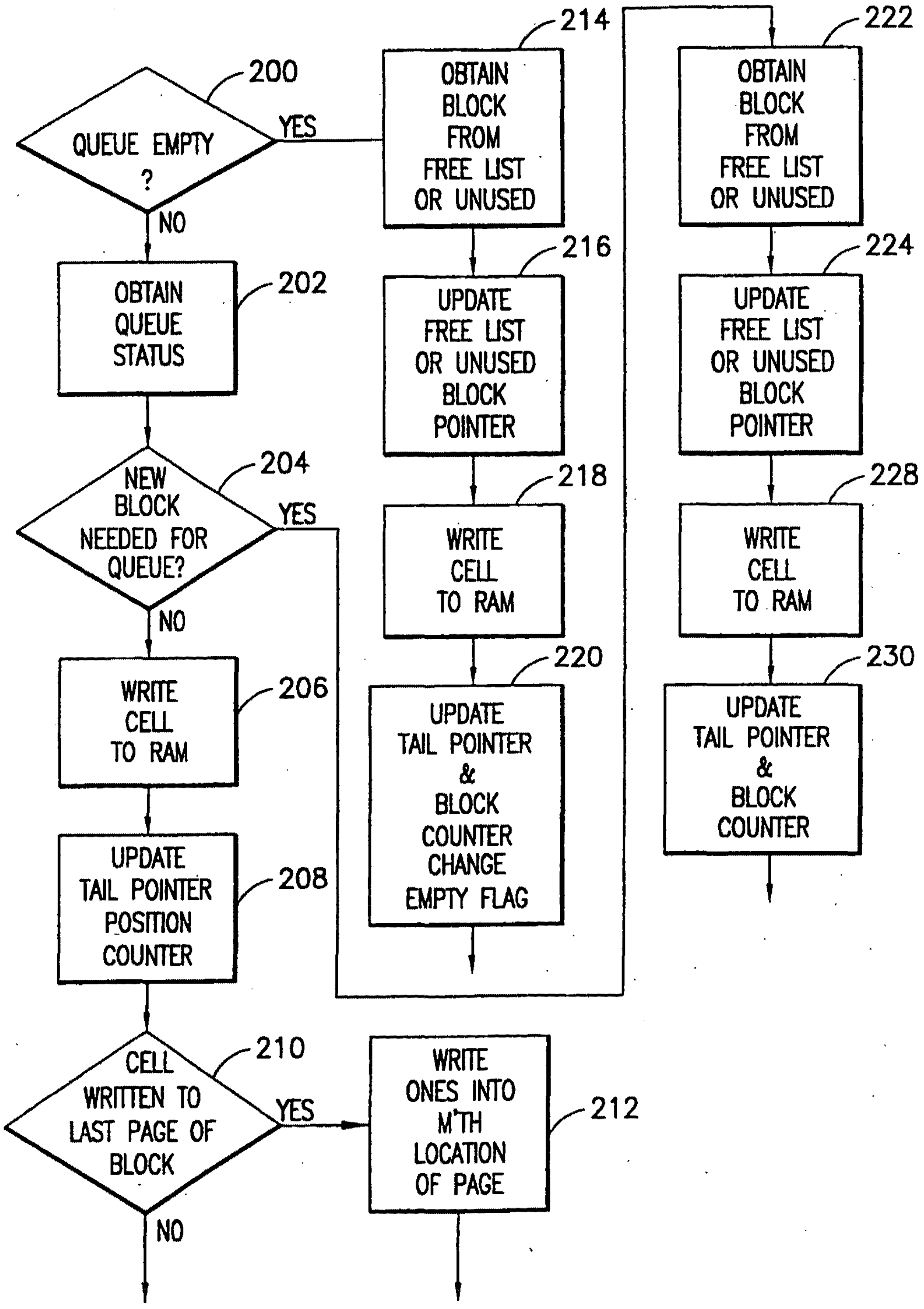

200
QUEUE EMPTY ?
YES
NO
202
OBTAIN QUEUE STATUS
204
NEW BLOCK NEEDED FOR QUEUE?
YES
NO
206
WRITE CELL TO RAM
208
UPDATE TAIL POINTER POSITION COUNTER
210
CELL WRITTEN TO LAST PAGE OF BLOCK
YES
NO
212
WRITE ONES INTO M'TH LOCATION OF PAGE
214
OBTAIN BLOCK FROM FREE LIST OR UNUSED
216
UPDATE FREE LIST OR UNUSED BLOCK POINTER
218
WRITE CELL TO RAM
220
UPDATE TAIL POINTER & BLOCK COUNTER CHANGE EMPTY FLAG
222
OBTAIN BLOCK FROM FREE LIST OR UNUSED
224
UPDATE FREE LIST OR UNUSED BLOCK POINTER
228
WRITE CELL TO RAM
230
UPDATE TAIL POINTER & BLOCK COUNTER

FIG.4a

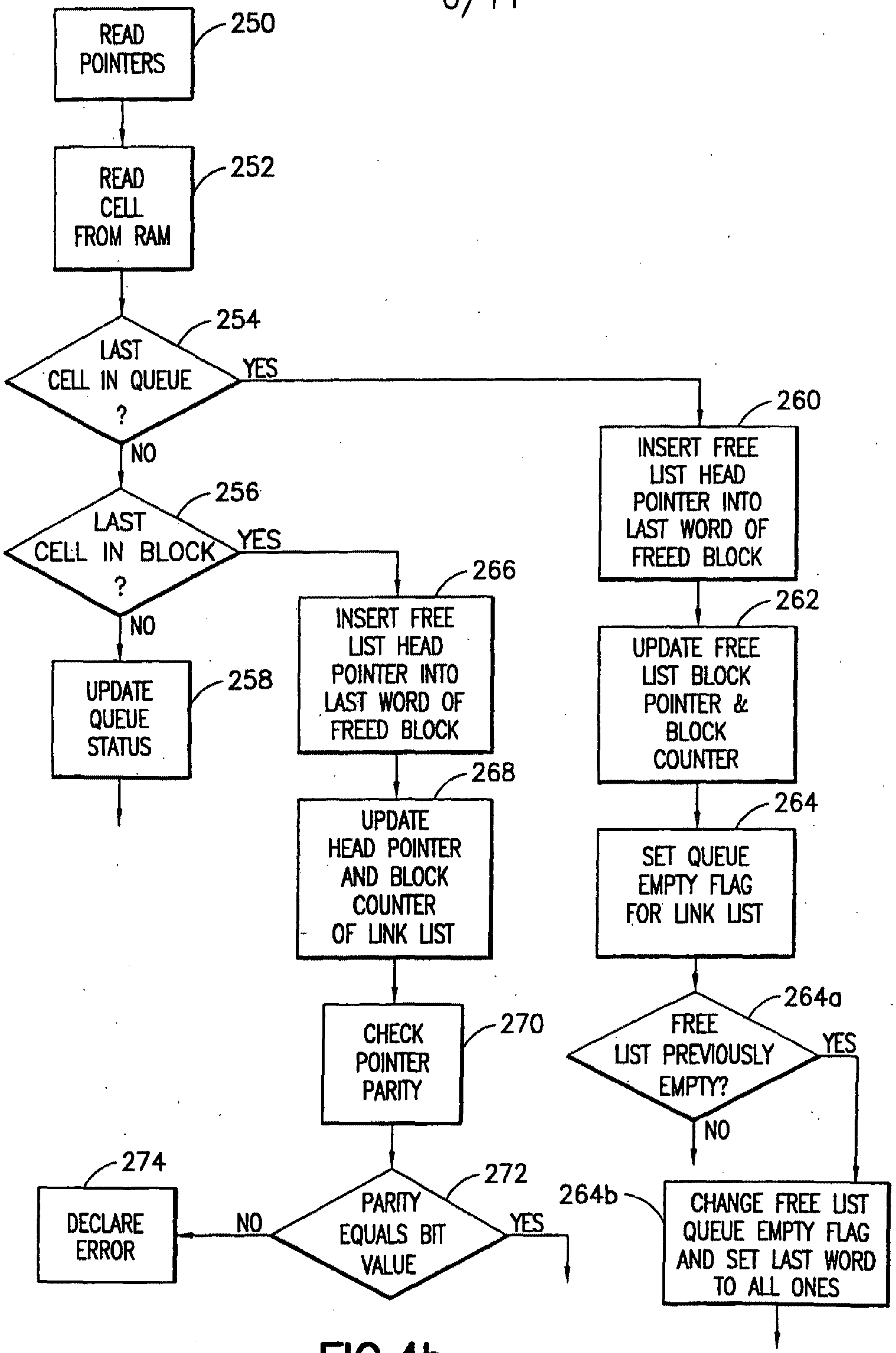

READ POINTERS
250
READ CELL FROM RAM
252
LAST CELL IN QUEUE ?
254
YES
NO
LAST CELL IN BLOCK ?
256
YES
NO
UPDATE QUEUE STATUS
258
INSERT FREE LIST HEAD POINTER INTO LAST WORD OF FREED BLOCK
266
UPDATE HEAD POINTER AND BLOCK COUNTER OF LINK LIST
268
CHECK POINTER PARITY
270
PARITY EQUALS BIT VALUE
272
NO
YES
DECLARE ERROR
274
INSERT FREE LIST HEAD POINTER INTO LAST WORD OF FREED BLOCK
260
UPDATE FREE LIST BLOCK POINTER & BLOCK COUNTER
262
SET QUEUE EMPTY FLAG FOR LINK LIST
264
FREE LIST PREVIOUSLY EMPTY?
264a
YES
NO
264b
CHANGE FREE LIST QUEUE EMPTY FLAG AND SET LAST WORD TO ALL ONES

FIG.4b

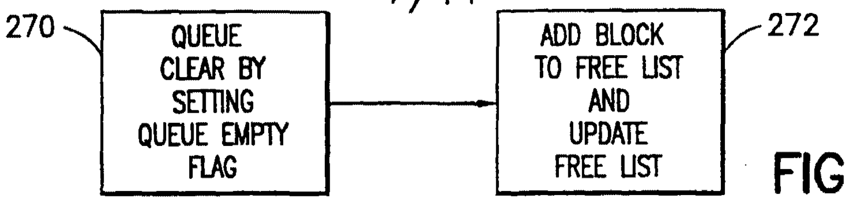

270
QUEUE CLEAR BY SETTING QUEUE EMPTY FLAG
ADD BLOCK TO FREE LIST AND UPDATE FREE LIST
272

FIG.4c

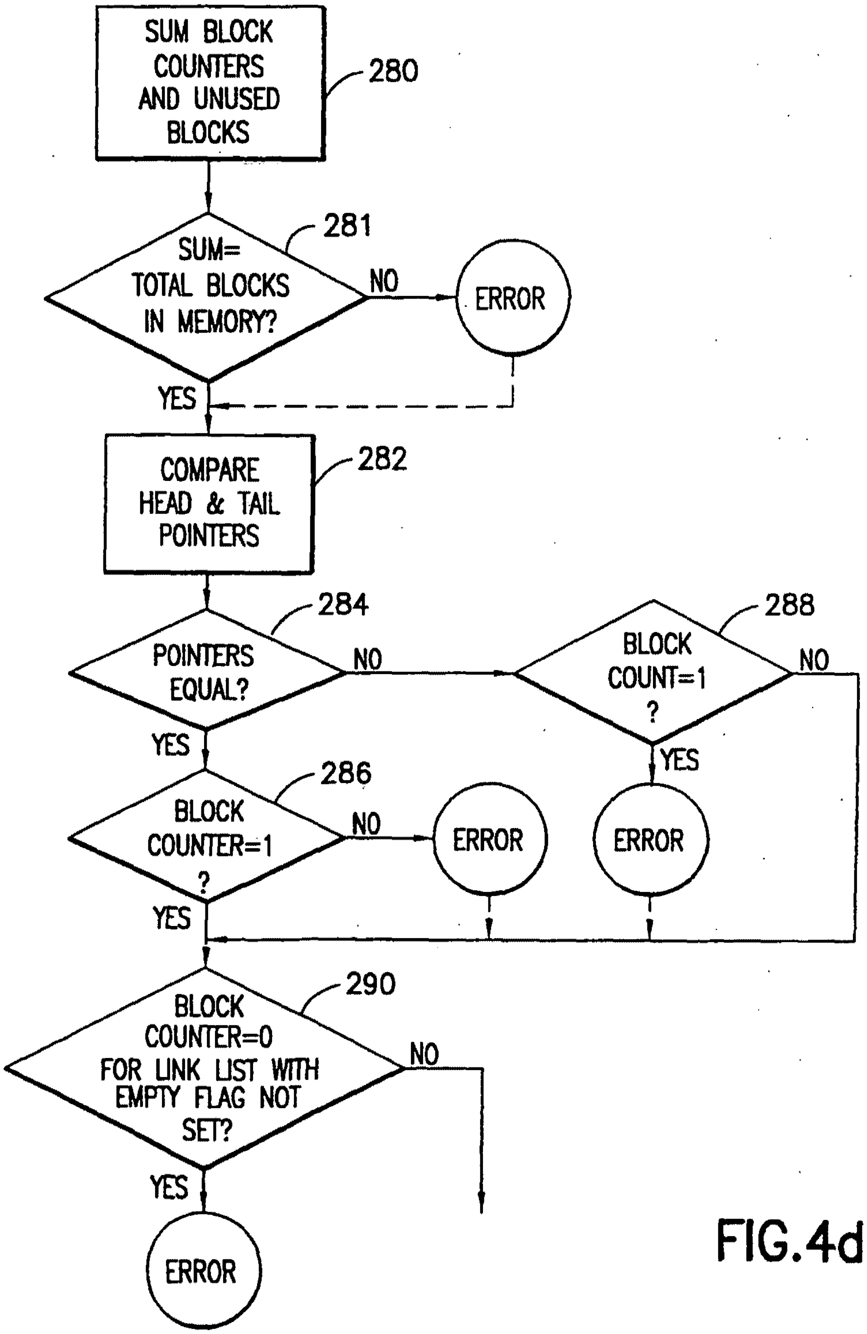

SUM BLOCK COUNTERS AND UNUSED BLOCKS
280
SUM= TOTAL BLOCKS IN MEMORY?
281
NO
ERROR
YES
COMPARE HEAD & TAIL POINTERS
282
POINTERS EQUAL?
284
NO
BLOCK COUNT=1 ?
288
NO
YES
ERROR
YES
BLOCK COUNTER=1 ?
286
NO
ERROR
YES
BLOCK COUNTER=0 FOR LINK LIST WITH EMPTY FLAG NOT SET?
290
NO
YES
ERROR

FIG.4d

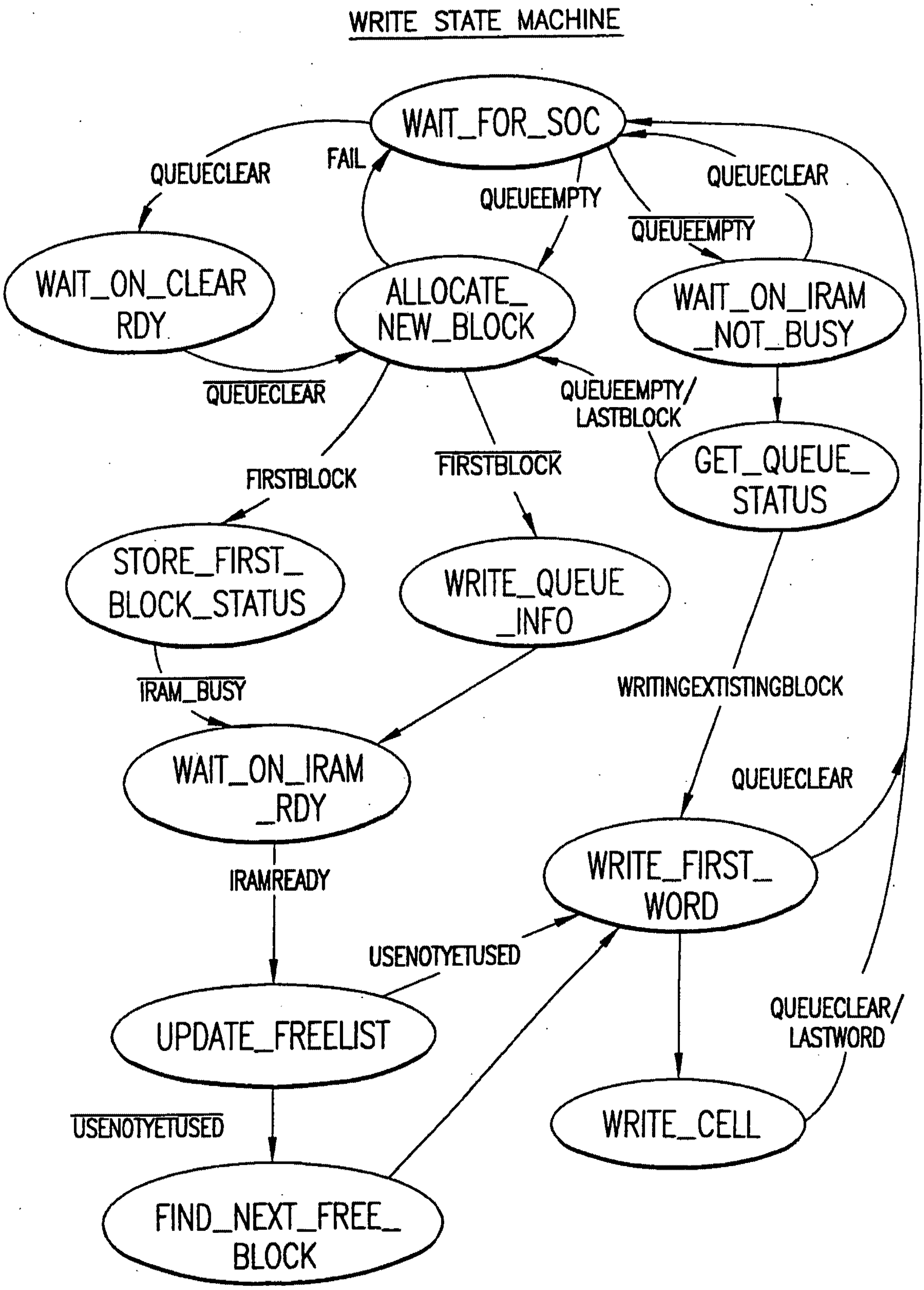

WRITE STATE MACHINE
WAIT_FOR_SOC
QUEUECLEAR
FAIL
QUEUEEMPTY
QUEUECLEAR
QUEUEEMPTY
WAIT_ON_CLEAR RDY
ALLOCATE_ NEW_BLOCK
WAIT_ON_IRAM _NOT_BUSY
QUEUECLEAR
QUEUEEMPTY/ LASTBLOCK
FIRSTBLOCK
FIRSTBLOCK
GET_QUEUE_ STATUS
STORE_FIRST_ BLOCK_STATUS
WRITE_QUEUE _INFO
IRAM_BUSY
WRITINGEXTISTINGBLOCK
WAIT_ON_IRAM _RDY
QUEUECLEAR
IRAMREADY
WRITE_FIRST_ WORD
USENOTYETUSED
UPDATE_FREELIST
QUEUECLEAR/ LASTWORD
USENOTYETUSED
WRITE_CELL
FIND_NEXT_FREE_ BLOCK

FIG.5a

READ STATE MACHINE

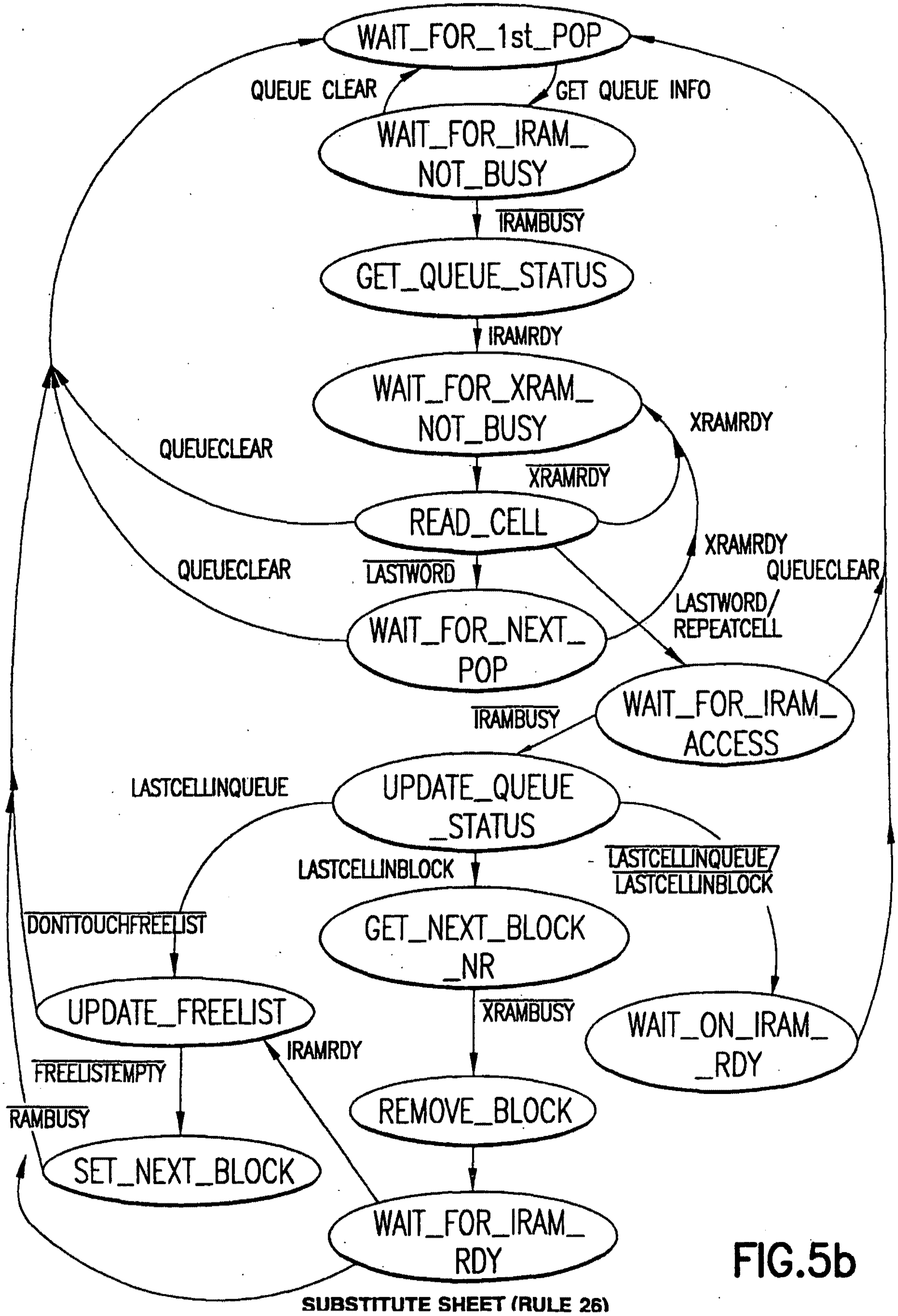


FIG.5b

CLEAR STATE MACHINE

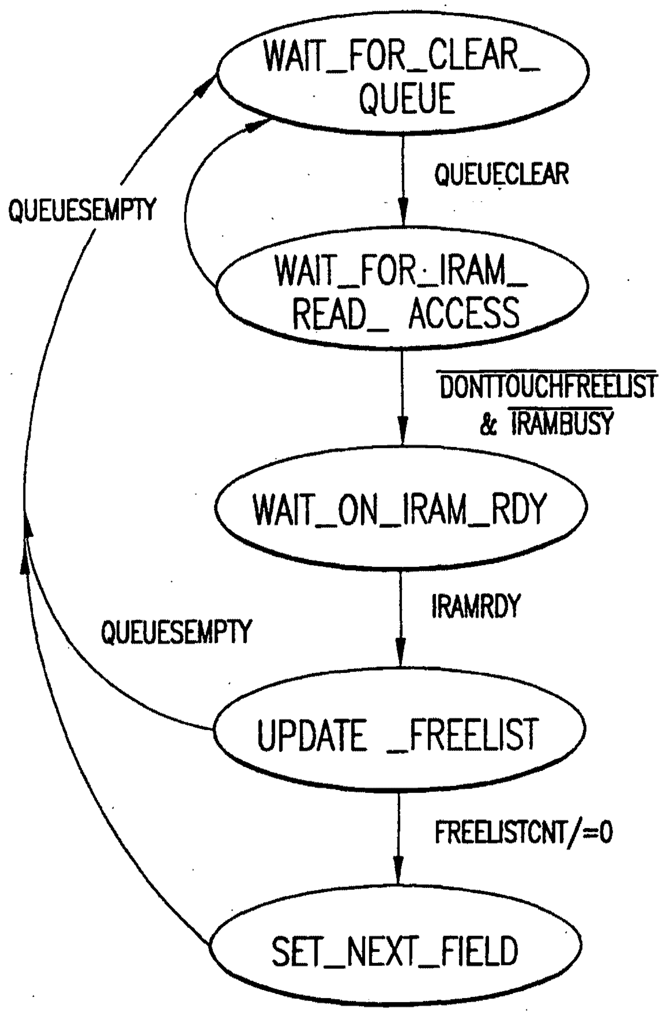


FIG.5c

MONITORING STATE MACHINE

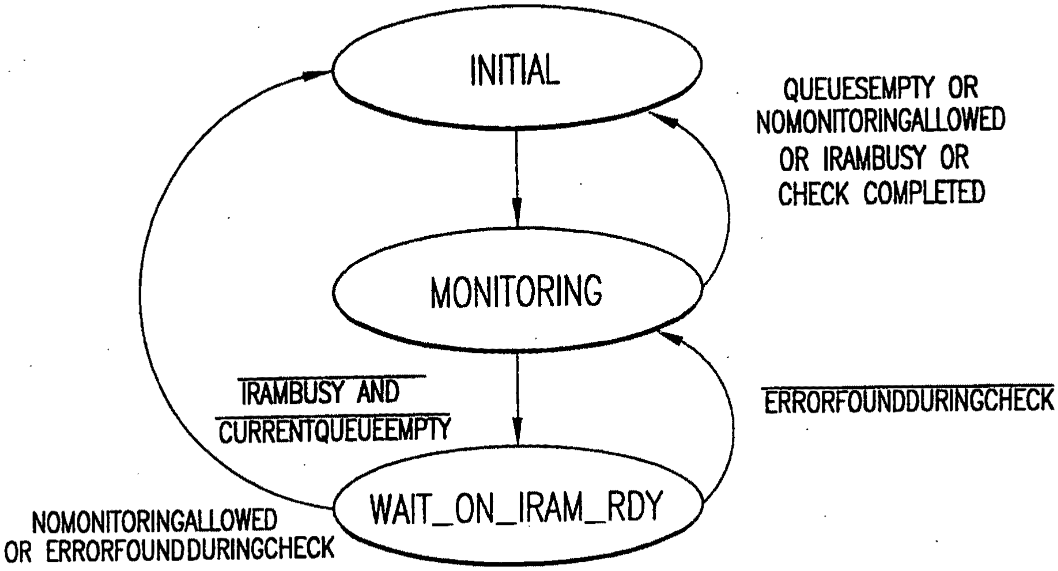

INITIAL
QUEUESEMPTY OR NOMONITORINGALLOWED OR IRAMBUSY OR CHECK COMPLETED
MONITORING
$\overline{\text{IRAMBUSY AND}}$
$\overline{\text{CURRENTQUEUEEMPTY}}$
$\overline{\text{ERRORFOUNDDURINGCHECK}}$
WAIT_ON_IRAM_RDY
NOMONITORINGALLOWED OR ERRORFOUNDDURINGCHECK

FIG.5d

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/02131

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 12/00
US CL :711/153
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 711/153, 711/207, 370/232, 370/398

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, MAYA

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X — Y	US 5,390,175 A (HILLER ET AL.) 14 February 1995, col. 45, line 10; Fig. 24; col. 37, line 20; col. 37, line 25; col. 21, line 35; col. 22, line 37; col 20, line 63; col. 54, line 34; col. 55, line 53; col. 35, line 56; col. 21, line 60; col. 37, line 19; col. 35, line 56	1-5, 7, 8, 10, 14-19, 20, 21, 22, 23 ———— 6, 9, 11-13
Y	US 5,123,101 A (SINDHU) 16 June 1992, col. 21, line 68.	6, 9, 11-13
A, P	US 5,654,962 A (ROSTOKER ET AL) 05 August 1997`	1-23

□ Further documents are listed in the continuation of Box C. □ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 30 MARCH 1998	Date of mailing of the international search report 0 4 AUG 1998
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer DAVID LANGJAHR Telephone No. (703) 305-4034

Ifw

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Atty. Docket No. (Opt.) **CROSS1120-33**
Applicant **Geoffrey B. Hoese, et al.**	
Application Number **12/690,592**	Filed **01/20/2010**
For **Storage Router and Method for Providing Virtual Local Storage**	
Group Art Unit **2111**	Examiner **Unknown**
Confirmation Number: **8115**	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Certification of Transmission Under 37 C.F.R. 1.8

I hereby certify that this correspondence is being deposited with the U.S. Postal Service as First Class Mail in a box addressed to The Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22312-1450 on 5-21, 2010

Signature

PAUL HAYGOOD
Printed Name

Dear Sir,

Applicant respectfully requests, pursuant to 37 C.F.R. §§ 1.56, 1.97 and 1.98, that the information listed on the attached SB08A/B form(s) be considered and cited in the examination of the above-identified application. A copy of U.S. Patent(s) and U.S. Patent Application Publication(s) listed on the attached SB08A form is not being submitted with this Information Disclosure Statement pursuant to the waiver of 37 C.F.R. § 1.98(a)(2)(i) by the U.S. Patent and Trademark Office. A copy of foreign patent documents as well as the information listed on the attached SB08B form is enclosed for the convenience of the Examiner.

☒ This Information Disclosure Statement is being submitted within three months of the filing date of a national application other than a continued prosecution application under 37 C.F.R. § 1.53(d).

□ This Information Disclosure Statement is being submitted within three months of the date of entry of the national stage as set forth in 37 C.F.R. § 1.491 in an international application;

☒ This Information Disclosure Statement is being submitted before the mailing of a first Office action on the merits; or

□ This Information Disclosure Statement is being submitted before the mailing of a first Office action after the filing of a request for continued examination under 37 C.F.R. § 1.114.

□ This Information Disclosure Statement is being submitted after the period specified in 37 C.F.R. § 1.97(b) and before the mailing date of any of a final action under 37 C.F.R. § 1.113, a notice of allowance under 37 C.F.R. § 1.311, or an action that otherwise closes prosecution in the application, and is accompanied by one of:

- □ The statement specified in 37 C.F.R. § 1.97(e); or
- □ The fee set forth in 37 C.F.R. § 1.17(p). Applicant hereby authorizes the Commissioner to deduct the amount of $180 from Deposit Account No. 50-3183 of Sprinkle IP Law Group for the filing fee of this Information Disclosure Statement.

□ This Information Disclosure Statement is being submitted after the period specified in 37 C.F.R. § 1.97(c) and on or before payment of the issue fee and is accompanied by:

- □ The statement specified in 37 C.F.R. § 1.97(e); and
- □ The fee set forth in 37 C.F.R. § 1.17(p). Applicant hereby authorizes the Commissioner to deduct the amount of $180 from Deposit Account No. 50-3183 of Sprinkle IP Law Group for the filing fee of this Information Disclosure Statement.

Pursuant to 37 C.F.R. § 1.97(e), Applicant hereby states:

□ That each item of information contained in the information disclosure statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of the information disclosure statement; or

□ That no item of information contained in the information disclosure statement was cited in a communication from a foreign patent office in a counterpart foreign application, and, to the knowledge of the person signing the certification after making reasonable inquiry, no item of information contained in the information disclosure statement was known to any individual designated in 37 C.F.R. § 1.56(c) more than three months prior to the filing of the information disclosure statement.

Furthermore, pursuant to 37 C.F.R. §§ 1.97(g) and (h), no representation is made that a search has been made or that this information is material to patentability of the present application.

Applicant respectfully submits that the claims of Applicant's above-referenced patent application are patentably distinguishable from the listed information.

Respectfully submitted,

Sprinkle IP Law Group
Attorneys for Applicant

John L. Adair
Reg. No. 48,828

Dated: 5-13-2010

1301 W. 25th Street, Suite 408
Austin, Texas 78705
Tel. (512) 637-9220
Fax. (512) 317-9088

CBL
B15

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 490 973 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
25.02.1998 Bulletin 1998/09

(21) Application number: 90913922.2

(22) Date of filing: 20.08.1990

(51) Int Cl.6: G06F 17/30, G06F 15/16

(86) International application number:
PCT/US90/04711

(87) International publication number:
WO 91/03788 (21.03.1991 Gazette 1991/07)

(54) PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE

DATEIENPROZESSORAUFBAU IM PARALLELEN EINGANGS/AUSGANGS NETZWERK

ARCHITECTURE DE SERVEUR DE FICHIER, EN RESEAU ENTREE/SORTIE PARALLLELE

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB IT LI LU NL SE

(30) Priority: 08.09.1989 US 404959

(43) Date of publication of application:
24.06.1992 Bulletin 1992/26

(60) Divisional application: 97112889.7

(73) Proprietor: AUSPEX SYSTEMS, INC.
Santa Clara, CA 95054 (US)

(72) Inventors:
- ROW, Edward, John
Mountain View, CA 94064 (US)
- BOUCHER, Laurence, B.
Saratoga, CA 95070 (US)
- PITTS, William, M.
Los Altos, CA 94022 (US)
- BLIGHTMAN, Stephen, E.
San Jose, CA 95133 (US)

(74) Representative: Barnard, Eric Edward et al
BROOKES & MARTIN
High Holborn House
52/54 High Holborn
London WC1V 6SE (GB)

(56) References cited:

WO-A-89/03086	US-A- 4 710 868
US-A- 4 819 159	US-A- 4 887 204
US-A- 4 897 781	

- COMPUTER STANDARDS AND INTERFACES vol. 8, no. 1, 1988, LAUSANNE CH pages 45 - 48 , XP51969 A.OSADZINSKI 'THE NETWORK FILE SYSTEM (NFS)'

Printed by Jouve, 75001 PARIS (FR)

Description

The present application is related to the following European Patent Applications:

1. MULTIPLE FACILITY -OPERATING SYSTEM ARCHITECTURE, Serial Number 90914006.3 (0490180), and

2. ENHANCED VMEBUS PROTOCOL UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA TRANSFER, Serial Number 90914333.1 (0490988).

BACKGROUND OF THE INVENTION

Field of the Invention

The invention relates to computer data networks, and more particularly, to network file server architectures for computer networks.

Description of the Related Art

Over the past ten years, remarkable increases in hardware price/performance ratios have caused a startling shift in both technical and office computing environments. Distributed workstation-server networks are displacing the once pervasive dumb terminal attached to mainframe or minicomputer. To date, however, network I/O limitations have constrained the potential performance available to workstation users. This situation has developed in part because dramatic jumps in microprocessor performance have exceeded increases in network I/O performance.

In a computer network, individual user workstations are referred to as clients, and shared resources for filing, printing, data storage and wide-area communications are referred to as servers. Clients and servers are all considered nodes of a network. Client nodes use standard communications protocols to exchange service requests and responses with server nodes.

Present-day network clients and servers usually run the DOS, Macintosh OS, OS/2, or Unix operating systems. Local networks are usually Ethernet or Token Ring at the high end, Arcnet in the midrange, or LocalTalk or StarLAN at the low end. The client-server communication protocols are fairly strictly dictated by the operating system environment -- usually one of several proprietary schemes for PCs (NetWare, 3Plus, Vines, LANManager, LANServer); AppleTalk for Macintoshes; and TCP/IP with NFS or RFS for Unix. These protocols are all well-known in the industry.

Unix client nodes typically feature a 16- or 32-bit microprocessor with 1-8 MB of primary memory, a 640 x 1024 pixel display, and a built-in network interface. A 40-100 MB local disk is often optional. Low-end examples are 80286-based PCs or 68000-based Macintosh I's; mid-range machines include 80386 PCs, Macintosh II's, and 680X0-based Unix workstations; high-end machines include RISC-based DEC, HP, and Sun Unix workstations. Servers are typically nothing more than repackaged client nodes, configured in 19-inch racks rather than desk sideboxes. The extra space of a 19-inch rack is used for additional backplane slots, disk or tape drives, and power supplies.

Driven by RISC and CISC microprocessor developments, client workstation performance has increased by more than a factor of ten in the last few years. Concurrently, these extremely fast clients have also gained an appetite for data that remote servers are unable to satisfy. Because the I/O shortfall is most dramatic in the Unix environment, the description of the preferred embodiment of the present invention will focus on Unix file servers. The architectural principles that solve the Unix server I/O problem, however, extend easily to server performance bottlenecks in other operating system environments as well. Similarly, the description of the preferred embodiment will focus on Ethernet implementations, though the principles extend easily to other types of networks.

In most Unix environments, clients and servers exchange file data using the Network File System ("NFS"), a standard promulgated by Sun Microsystems and now widely adopted by the Unix community. NFS is defined in a document entitled, "NFS: Network File System Protocol Specification," Request For Comments (RFC) 1094, by Sun Microsystems, Inc. (March 1989).

While simple and reliable, NFS is not optimal. Clients using NFS place considerable demands upon both networks and NFS servers supplying clients with NFS data. This demand is particularly acute for so-called diskless clients that have no local disks and therefore depend on a file server for application binaries and virtual memory paging as well as data. For these Unix client-server configurations, the ten-to-one increase in client power has not been matched by a ten-to-one increase in Ethernet capacity, in disk speed, or server disk-to-network I/O throughput.

The result is that the number of diskless clients that a single modem high-end server can adequately support has dropped to between 5-10, depending on client power and application workload. For clients containing small local disks for applications and paging, referred to as dataless clients, the client-to-server ratio is about twice this, or between 10-20.

Such low client/server ratios cause piecewise network configurations in which each local Ethernet contains isolated traffic for its own 5-10 (diskless) clients and dedicated server. For overall connectivity, these local networks are usually joined together with an Ethernet backbone or, in the future, with an FDDI backbone. These backbones are typically connected to the local networks either by IP routers or MAC-level bridges, coupling the local networks together directly, or by a second server functioning as a network interface, coupling servers for all the local networks together.

In addition to performance considerations, the low client-to-server ratio creates computing problems in several additional ways:

1. Sharing. Development groups of more than 5-10 people cannot share the same server, and thus cannot easily share files without file replication and manual, multi-server updates. Bridges or routers are a partial solution but inflict a performance penalty due to more network hops.
2. Administration. System administrators must maintain many limited-capacity servers rather than a few more substantial servers. This burden includes network administration, hardware maintenance, and user account administration.
3. File System Backup. System administrators or operators must conduct multiple file system backups, which can be onerously time consuming tasks. It is also expensive to duplicate backup peripherals on each server (or every few servers if slower network backup is used).
4. Price Per Seat. With only 5-10 clients per server, the cost of the server must be shared by only a small number of users. The real cost of an entry-level Unix workstation is therefore significantly greater, often as much as 140% greater, than the cost of the workstation alone.

The widening I/O gap, as well as administrative and economic considerations, demonstrates a need for higher-performance, larger-capacity Unix file servers. Conversion of a display-less workstation into a server may address disk capacity issues, but does nothing to address fundamental I/O limitations. As an NFS server, the one-time workstation must sustain 5-10 or more times the network, disk, backplane, and file system throughput than it was designed to support as a client. Adding larger disks, more network adaptors, extra primary memory, or even a faster processor do not resolve basic architectural I/O constraints; I/O throughput does not increase sufficiently.

Other prior art computer architectures, while not specifically designed as file servers, may potentially be used as such. In one such well-known architecture, a CPU, a memory unit, and two I/O processors are connected to a single bus. One of the I/O processors operates a set of disk drives, and if the architecture is to be used as a server, the other I/O processor would be connected to a network. This architecture is not optimal as a file server, however, at least because the two I/O processors cannot handle network file requests without involving the CPU. All network file requests that are received by the network I/O processor are first transmitted to the CPU, which makes appropriate requests to the disk-I/O processor for satisfaction of the network request.

In another such computer architecture, a disk controller CPU manages access to disk drives, and several other CPUs, three for example, may be clustered around the disk controller CPU. Each of the other CPUs can be connected to its own network. The network CPUs are each connected to the disk controller CPU as well as to each other for interprocessor communication. One of the disadvantages of this computer architecture is that each CPU in the system runs its own complete operating system. Thus, network file server requests must be handled by an operating system which is also heavily loaded with facilities and processes for performing a large number of other, non file-server tasks. Additionally, the interprocessor communication is not optimized for file server type requests.

In yet another computer architecture, a plurality of CPUs, each having its own cache memory for data and instruction storage, are connected to a common bus with a system memory and a disk controller. The disk controller and each of the CPUs have direct memory access to the system memory, and one or more of the CPUs can be connected to a network. This architecture is disadvantageous as a file server because, among other things, both file data and the instructions for the CPUs reside in the same system memory. There will be instances, therefore, in which the CPUs must stop running while they wait for large blocks of file data to be transferred between the system memory and the network CPU. Additionally, as with both of the previously described computer architectures, the entire operating system runs on each of the CPUs, including the network CPU.

In yet another type of computer architecture, a large number of CPUs are connected together in a hypercube topology. One or more of these CPUs can be connected to networks, while another can be connected to disk drives. This architecture is also disadvantageous as a file server because, amongst other things each processor runs the entire operating system. Interprocessor communication is also not optimal for file server applications.

US-A-4819159 describes a data control unit for use with a data network which employs a mass storage device, a file processor, a buffer memory with a cache and a storage processor unit coupleable to the mass storage device and the file processor. The file processor serves to translate file system requests into store and retrieval requests activated in the mass storage device.

WO-A-89/03086 describes a network composed of a plurality of PC's linked to a main frame computer with a data base via a number of intermediate computers. The intermediate computers serve to respond to network file requests and to additionally provide resources for the PC's.

An article entitled 'The Network File System' (NFS) by A. Osadzinski published in Computer Standards and Interfaces Vol. 8 (1988/89) No. 1 describes a NFS protocol which provides transparent file access for client work stations from a file server with a mass storage device and a host processor linked together with a network.

SUMMARY OF THE INVENTION

As is known, for example from WO-A-89/03086, the present invention provides a network server apparatus for use with a first data network and a mass storage device, including a host processor unit capable of running remote procedures defined by a client node on said network.

In accordance with the invention, the apparatus comprises an interface processor unit coupleable to the network and to the mass storage device and means in said interface processor unit for satisfying network storage requests from said network to store data from said network in said mass storage device, for satisfying network retrieval requests from said network to retrieve data from said mass storage device to said network, and for transmitting predefined categories of messages from said network to said host processor unit for processing in said host processor unit, said transmitted messages including all requests by a network client to run client-defined procedures on said network server apparatus.

An implementation of the invention involves a file server architecture comprising one or more network controllers, one or more file controllers, one or more storage processors, and a system or buffer memory, all connected over a message passing bus and operating in parallel with a Unix host processor. The network controllers each connect to one or more network, and provide all protocol processing between the network layer data format and an internal file server format for communicating client requests to other processors in the server. Only those data packets which cannot be interpreted by the network controllers, for example client requests to run a client-defined program on the server, are transmitted to the Unix host for processing. Thus the network controllers, file controllers and storage processors contain only small parts of an overall operating system, and each is optimized for the particular type of work to which it is dedicated.

Client requests for file operations are transmitted to one of the file controllers which, independently of the Unix host, manages the virtual file system of a mass storage device which is coupled to the storage processors. The file. controllers may also control data buffering between the storage processors and the network controllers, through the system memory. The file controllers preferably each include a local buffer memory for caching file control information, separate from the system memory for caching file data. Additionally, the network controllers, file processors and storage processors are all designed to avoid any instruction fetches from the system memory, instead keeping all instruction memory separate and local. This arrangement eliminates contention on the backplane between microprocessor instruction fetches and transmissions of message and file data.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described with respect to particular embodiments thereof, and reference will be made to the drawings, in which:

Fig. 1. is a block diagram of a prior art file server architecture;
Fig. 2 is a block diagram of a file server architecture according to the invention;
Fig. 3 is a block diagram of one of the network controllers shown in Fig. 2;
Fig. 4 is a block diagram of one of the file controllers shown in Fig. 2;
Fig. 5 is a block diagram of one of the storage processors shown in Fig. 2;
Fig. 6 is a block diagram of one of the system memory cards shown in Fig. 2;
Figs. 7A-C are a flowchart illustrating the operation of a fast transfer protocol BLOCK WRITE cycle; and
Figs. 8A-C are a flowchart illustrating the operation of a fast transfer protocol BLOCK READ cycle.

DETAILED DESCRIPTION

For comparison purposes and background, an illustrative prior-art file server architecture will first be described with respect to Fig. 1. Fig. 1 is an overall block diagram of a conventional prior-art Unix-based file server for Ethernet networks. It consists of a host CPU card 10 with a single microprocessor on board. The host CPU card 10 connects to an Ethernet #1 12, and it connects via a memory management unit (MMU) 11 to a large memory array 16. The host CPU card 10 also drives a keyboard, a video display, and two RS232 ports (not shown). It also connects via the MMU 11 and a standard 32-bit VME bus 20 to various peripheral devices, including an SMD disk controller 22 controlling one or two disk drives 24, a SCSI host adaptor 26 connected to a SCSI bus 28, a tape controller 30 connected to a quarter-inch tape drive 32, and possibly a network #2 controller 34 connected to a second Ethernet 36. The SMD disk controller 22 can communicate with memory array 16 by direct memory access via bus 20 and MMU 11, with either the disk controller or the MMU acting as a bus master. This configuration is illustrative; many variations are available.

The system communicates over the Ethemets using industry standard TCP/IP and NFS protocol stacks. A description of protocol stacks in general can be found in Tanenbaum, "Computer Networks" (Second Edition, Prentice Hall: 1988). File server protocol stacks are described at pages 535-546.

Basically, the following protocol layers are implemented in the apparatus of Fig. 1:

Network Layer. The network layer converts data packets between a format specific to Ethemets and a format which is independent of the particular type of network used. the Ethernet-specific format which is used in the apparatus of Fig. 1 is described in Hornig, "A Standard For The Transmission of IP Datagrams Over Ethernet Networks," RFC 894 (April 1984).

The Internet Protocol (IP) Layer. This layer provides the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. For messages to be sent from the file server to a client, a higher level in the server calls the IP module, providing the internet address of the destination client and the message to transmit. The IP module performs any required fragmentation of the message to accommodate packet size limitations of any intervening gateway, adds internet headers to each fragment, and calls on the network layer to transmit the resulting internet datagrams. The internet header includes a local network destination address (translated from the internet address) as well as other parameters.

For messages received by the IP layer from the network layer, the IP module determines from the internet address whether the datagram is to be forwarded to another host on another network, for example on a second Ethemet such as 36 in Fig. 1, or whether it is intended for the server itself. If it is intended for another host on the second network, the IP module determines a local net address for the destination and calls on the local network layer for that network to send the datagram. If the datagram is intended for an application program within the server, the IP layer strips off the header and passes the remaining portion of the message to the appropriate next higher layer. The internet protocol standard used in the illustrative apparatus of Fig. 1 is specified in Information Sciences Institute, "Internet Protocol, DARPA Internet Program Protocol Specification," RFC 791 (September 1981).

TCP/UDP Layer. This layer is a datagram service with more elaborate packaging and addressing options than the IP layer. For example, whereas an IP datagram can hold about 1,500 bytes and be addressed to hosts, UDP datagrams can hold about 64KB and be addressed to a particular port within a host. TCP and UDP are alternative protocols at this layer; applications requiring ordered reliable delivery of streams of data may use TCP, whereas applications (such as NFS) which do not require ordered and reliable delivery may use UDP.

The prior art file server of Fig. 1 uses both TCP and UDP. It uses UDP for file server-related services, and uses TCP for certain other services which the server provides to network clients. The UDP is specified in Postel, "User Datagram Protocol," RFC 768 (August 28, 1980). TCP is specified in Postel, "Transmission Control Protocol," RFC 761 (January 1980) and RFC 793 (September 1981).

XDR/RPC Layer. This layer provides functions callable from higher level programs to run a designated procedure on a remote machine. It also provides the decoding necessary to permit a client machine to execute a procedure on the server. For example, a caller process in a client node may send a call message to the server of Fig. 1. The call message includes a specification of the desired procedure, and its parameters. The message is passed up the stack to the RPC layer, which calls the appropriate procedure within the server. When the procedure is complete, a reply message is generated and RPC passes it back down the stack and over the network to the caller client. RPC is described in Sun Microsystems, Inc., "RPC: Remote Procedure Call Protocol Specification, Version 2," RFC 1057 (June 1988).

RPC uses the XDR external data representation standard to represent information passed to and from the underlying UDP layer. XDR is merely a data encoding standard, useful for transferring data between different computer architectures. Thus, on the network side of the XDR/RPC layer, information is machine-independent; on the host application side, it may not be. XDR is described in Sun Microsystems, Inc., "XDR: External Data Representation Standard," RFC 1014 (June 1987).

NFS Layer. The NFS ("network file system") layer is one of the programs available on the server which an RPC request can call. The combination of host address, program number, and procedure number in an RPC request can specify one remote NFS procedure to be called.

Remote procedure calls to NFS on the file server of Fig. 1 provide transparent, stateless, remote access to shared files on the disks 24. NFS assumes a file system that is hierarchical, with directories at all but the bottom level of files. Client hosts can call any of about 20 NFS procedures including such procedures as reading a specified number of bytes from a specified file; writing a specified number of bytes to a specified file; creating, renaming and removing specified files; parsing directory trees; creating and removing directories; and reading and setting file attributes. The location on disk to which and from which data is stored and retrieved is always specified in logical terms, such as by a file handle or Inode designation and a byte offset. The details of the actual data storage are hidden from the client.

The NFS procedures, together with possible higher level modules such as Unix VFS and UFS, perform all conversion of logical data addresses to physical data addresses such as drive, head, track and sector identification. NFS is specified in Sun Microsystems, Inc., "NFS: Network File System Protocol Specification," RFC 1094 (March 1989).

With the possible exception of the network layer, all the protocol processing described above is done in software, by a single processor in the host CPU card 10. That is, when an Ethernet packet arrives on Ethernet 12, the host CPU 10 performs all the protocol processing in the NFS stack, as well as the protocol processing for any other application which may be running on the host 10. NFS procedures are run on the host CPU 10, with access to memory 16 for both data and program code being provided via MMU 11. Logically specified data addresses are converted to a much more physically specified form and communicated to the SMD disk controller 22 or the SCSI bus 28, via the VME bus 20, and all disk caching is done by the host CPU 10 through the memory 16. The host CPU card 10 also runs procedures for performing various other functions of the file server, communicating with tape controller 30 via the VME bus 20. Among these are client-defined remote procedures requested by client workstations.

If the server serves a second Ethernet 36, packets from that Ethernet are transmitted to the host CPU 10 over the same VME bus 20 in the form of IP datagrams. Again, all protocol processing except for the network layer is performed by software processes running on the host CPU 10. In addition, the protocol processing for any message that is to be sent from the server out on either of the Ethernets 12 or 36 is also done by processes running on the host CPU 10.

It can be seen that the host CPU 10 performs an enormous amount of processing of data, especially if 5-10 clients on each of the two Ethernets are making file server requests and need to be sent responses on a frequent basis. The host CPU 10 runs a multitasking Unix operating system, so each incoming request need not wait for the previous request to be completely processed and returned before being processed. Multiple processes are activated on the host CPU 10 for performing different stages of the processing of different requests, so many requests may be in process at the same time. But there is only one CPU on the card 10, so the processing of these requests is not accomplished in a truly parallel manner. The processes are instead merely time sliced. The CPU 10 therefore represents a major bottleneck in the processing of file server requests.

Another bottleneck occurs in MMU 11, which must transmit both instructions and data between the CPU card 10 and the memory 16. All data flowing between the disk drives and the network passes through this interface at least twice.

Yet another bottleneck can occur on the VME bus 20, which must transmit data among the SMD disk controller 22, the SCSI host adaptor 26, the host CPU card 10, and possibly the network #2 controller 24.

PREFERRED EMBODIMENT-OVERALL HARDWARE ARCHITECTURE

In Fig. 2 there is shown a block diagram of a network file server 100 according to the invention. It can include multiple network controller (NC) boards, one or more file controller (FC) boards, one or more storage processor (SP) boards, multiple system memory boards, and one or more host processors. The particular embodiment shown in Fig. 2 includes four network controller boards 110a-110d, two file controller boards 112a-112b, two storage processors 114a-114b, four system memory cards 116a-116d for a total of 192MB of memory, and one local host processor 118. The boards 110, 112, 114, 116 and 118 are connected together over a VME bus 120 on which an enhanced block transfer mode as described in the ENHANCED VMEBUS PROTOCOL application identified above may be used. Each of the four network controllers 110 shown in Fig. 2 can be connected to up to two Ethernets 122, for a total capacity of 8 Ethernets 122a-122h. Each of the storage processors 114 operates ten parallel SCSI busses, nine of which can each support up to three SCSI disk drives each. The tenth SCSI channel on each of the storage processors 114 is used for tape drives and other SCSI peripherals.

The host 118 is essentially a standard SunOs Unix processor, providing all the standard Sun Open Network Computing (ONC) services except NFS and IP routing. Importantly, all network requests to run a user-defined procedure are passed to the host for execution. Each of the NC boards 110, the FC boards 112 and the SP boards 114 includes its own independent 32-bit microprocessor. These boards essentially offload from the host processor 118 virtually all of the NFS and disk processing. Since the vast majority of messages to and from clients over the Ethernets 122 involve NFS requests and responses, the processing of these requests in parallel by the NC, FC and SP processors, with minimal involvement by the local host 118, vastly improves file server performance. Unix is explicitly eliminated from virtually all network, file, and storage processing.

OVERALL SOFTWARE ORGANIZATION AND DATA FLOW

Prior to a detailed discussion of the hardware subsystems shown in Fig. 2, an overview of the software structure will now be undertaken. The software organization is described in more detail in the above-identified application entitled MULTIPLE FACILITY OPERATING SYSTEM ARCHITECTURE.

Most of the elements of the software are well known in the field and are found in most networked Unix systems, but there are two components which are not: Local NFS ("LNFS") and the messaging kernel ("MK") operating system kernel. These two components will be explained first.

The Messaging Kernel. The various processors in file server 100 communicate with each other through the use of a messaging kernel running on each of the processors 110, 112, 114 and 118. These processors do not share any instruction memory, so task-level communication cannot occur via straightforward procedure calls as it does in conventional Unix. Instead, the messaging kernel passes messages over VME bus 120 to accomplish all necessary interprocessor communication. Message passing is preferred over remote procedure calls for reasons of simplicity and speed.

Messages passed by the messaging kernel have a fixed 128-byte length. Within a single processor, messages are sent by reference; between processors, they are copied by the messaging kernel and then delivered to the destination process by reference. The processors of Fig. 2 have special hardware, discussed below, that can expediently exchange and buffer inter-processor messaging kernel messages.

The LNFS Local NFS interface. The 22-function NFS standard was specifically designed for stateless operation using unreliable communication. This means that neither clients nor server can be sure if they hear each other when they talk (unreliability). In practice, an in an Ethernet environment, this works well.

Within the server 100, however, NFS level datagrams are also used for communication between processors, in particular between the network controllers 110 and the file controller 112, and between the host processor 118 and the file controller 112. For this internal communication to be both efficient and convenient, it is undesirable and impractical to have complete statelessness or unreliable communications. Consequently, a modified form of NFS, namely LNFS, is used for internal communication of NFS requests and responses. LNFS is used only within the file server 100; the external network protocol supported by the server is precisely standard, licensed NFS. LNFS is described in more detail below.

The Network Controllers 110 each run an NFS server which, after all protocol processing is done up to the NFS layer, converts between external NFS requests and responses and internal LNFS requests and responses. For example, NFS requests arrive as RPC requests with XDR and enclosed in a UDP datagram. After protocol processing, the NFS server translates the NFS request into LNFS form and uses the messaging kernel to send the request to the file controller 112.

The file controller runs an LNFS server which handles LNFS requests both from network controllers and from the host 118. The LNFS server translates LNFS requests to a form appropriate for a file system server, also running on the file controller, which manages the system memory file data cache through a block I/O layer.

An overview of the software in each of the processors will now be set forth.

Network Controller 110

The optimized dataflow of the server 100 begins with the intelligent network controller 110. This processor receives Ethernet packets from client workstations. It quickly identifies NFS-destined packets and then performs full protocol processing on them to the NFS level, passing the resulting LNFS requests directly to the file controller 112. This protocol processing includes IP routing and reassembly, UDP demultiplexing, XDR decoding, and NFS request dispatching. The reverse steps are used to send an NFS reply back to a client. Importantly, these time-consuming activities are performed directly in the Network Controller 110, not in the host 118.

The server 100 uses conventional NFS ported from Sun Microsystems, Inc., Mountain View, CA, and is NFS protocol compatible.

Non-NFS network traffic is passed directly to its destination host processor 118.

The NCs 110 also perform their own IP routing. Each network controller 110 supports two fully parallel Ethernets. There are four network controllers in the embodiment of the server 100 shown in Fig. 2, so that server can support up to eight Ethernets. For the two Ethernets on the same network controller 110, IP routing occurs completely within the network controller and generates no backplane traffic. Thus attaching two mutually active Ethernets to the same controller not only minimizes their internet transit time, but also significantly reduces backplane contention on the VME bus 120. Routing table updates are distributed to the network controllers from the host processor 118, which runs either the gated or routed Unix demon.

While the network controller described here is designed for Ethernet LANs, it will be understood that the invention can be used just as readily with other network types, including FDDI.

File Controller 112

In addition to dedicating a separate processor for NFS protocol processing and IP routing, the server 100 also dedicates a separate processor, the intelligent file controller 112, to be responsible for all file system processing. It uses conventional Berkeley Unix 4.3 file system code and uses a binary-compatible data representation on disk. These two choices allow all standard file system utilities (particularly block-level tools) to run unchanged.

The file controller 112 runs the shared file system used by all NCs 110 and the host processor 118. Both the NCs and the host processor communicate with the file controller 112 using the LNFS interface. The NCs 110 use LNFS as described above, while the host processor 118 uses LNFS as a plug-in module to SunOs's standard Virtual File System ("VFS") interface.

When an NC receives an NFS read request from a client workstation, the resulting LNFS request passes to the FC 112. The FC 112 first searches the system memory 116 buffer cache for the requested data. If found, a reference to the buffer is returned to the NC 110. If not found, the LRU (least recently used) cache buffer in system memory 116 is freed and reassigned for the requested block. The FC then directs the SP 114 to read the block into the cache buffer from a disk drive array. When complete, the SP so notifies the FC, which in turn notifies the NC 100. The NC 110 then sends an NFS reply, with the data from the buffer, back to the NFS client workstation out on the network. Note that the SP 114 transfers the data into system memory 116, if necessary, and the NC 110 transferred the data from system memory 116 to the networks. The process takes place without any involvement of the host 118.

Storage Processor

The intelligent storage processor 114 manages all disk and tape storage operations. While autonomous, storage processors are primarily directed by the file controller 112 to move file data between system memory 116 and the disk subsystem. The exclusion of both the host 118 and the FC 112 from the actual data path helps to supply the performance needed to service many remote clients.

Additionally, coordinated by a Server Manager in the host 118, storage processor 114 can execute server backup by moving data between the disk subsystem and tape or other archival peripherals on the SCSI channels. Further, if directly accessed by host processor 118, SP 114 can provide a much higher performance conventional disk interface for Unix, virtual memory, and databases. In Unix nomenclature, the host processor 118 can mount boot, storage swap, and raw partitions via the storage processors 114.

Each storage processor 114 operates ten parallel, fully synchronous SCSI channels (busses) simultaneously. Nine of these channels support three arrays of nine SCSI disk drives each, each drive in an array being assigned to a different SCSI channel. The tenth SCSI channel hosts up to seven tape and other SCSI peripherals. In addition to performing reads and writes, SP 114 performs device-level optimizations such as disk seek queue sorting, directs device error recovery, and controls DMA transfers between the devices and system memory 116.

Host Processor 118

The local host 118 has three main purposes: to run Unix, to provide standard ONC network services for clients, and to run a Server Manager. Since Unix and ONC are ported from the standard SunOs Release 4 and ONC Services Release 2, the server 100 can provide identically compatible high-level ONC services such as the Yellow Pages, Lock Manager, DES Key Authenticator, Auto Mounter, and Port Mapper. Sun/2 Network disk booting and more general IP internet services such as Telnet, FTP, SMTP, SNMP, and reverse ARP are also supported. Finally, print spoolers and similar Unix demons operate transparently.

The host processor 118 runs the following software modules:

TCP and socket layers. The Transport Control Protocol ("TCP"), which is used for certain server functions other than NFS, provides reliable bytestream communication between two processors. Socket are used to establish TCP connections.

VFS interface. The Virtual File System ("VFS") interface is a standard SunOs file system interface. It paints a uniform file-system picture for both users and the non-file parts of the Unix operating system, hiding the details of the specific file system. Thus standard NFS, LNFS, and any local Unix file system can coexist harmoniously.

UFS interface. The Unix File System ("UFS") interface is the traditional and well-known Unix interface for communication with local-to-the-processor disk drives. In the server 100, it is used to occasionally mount storage processor volumes directly, without going through the file controller 112. Normally, the host 118 uses LNFS and goes through the file controller.

Device layer. The device layer is a standard software interface between the Unix device model and different physical device implementations. In the server 100, disk devices are not attached to host processors directly, so the disk driver in the host's device layer uses the messaging kernel to communicate with the storage processor 114.

Route and Port Mapper Demons. The Route and Port Mapper demons are Unix user-level background processes that maintain the Route and Port databases for packet routing. They are mostly inactive and not in any performance path.

Yellow Pages and Authentication Demon. The Yellow Pages and Authentication services are Sun-ONC standard network services. Yellow Pages is a widely used multipurpose name-to-name directory lookup service. The Authentication service uses cryptographic keys to authenticate, or validate, requests to insure that requestors have the proper privileges for any actions or data they desire.

Server Manager. The Server Manager is an administrative application suite that controls configuration, logs error and performance reports, and provides a monitoring and tuning interface for the system administrator. These functions can be exercised from either system console connected to the host 118, or from a system administrator's workstation.

The host processor 118 is a conventional OEM Sun central processor card, Model 3E/120. It incorporates a Motorola 68020 microprocessor and 4MB of on-board memory. Other processors, such as a SPARC-based processor, are also possible.

The structure and operation of each of the hardware components of server 100 will now be described in detail.

NETWORK CONTROLLER HARDWARE ARCHITECTURE

Fig. 3 is a block diagram showing the data path and some control paths for an illustrative one of the network controllers 110a. It comprises a 20 MHz 68020 microprocessor 210 connected to a 32-bit microprocessor data bus 212. Also connected to the microprocessor data bus 212 is a 256K byte CPU memory 214. The low order 8 bits of the microprocessor data bus 212 are connected through a bidirectional buffer 216 to an 8-bit slow-speed data bus 218. On the slow-speed data bus 218 is a 128K byte EPROM 220, a 32 byte PROM 222, and a multi-function peripheral (MFP) 224. The EPROM 220 contains boot code for the network controller 110a, while the PROM 222 stores various operating parameters such as the Ethernet addresses assigned to each of the two Ethernet interfaces on the board. Ethernet address information is read into the corresponding interface control block in the CPU memory 214 during initialization. The MFP 224 is a Motorola 68901, and performs various local functions such as timing, interrupts, and general purpose I/O. The MFP 224 also includes a UART for interfacing to an RS232 port 226. These functions are not critical to the invention and will not be further described herein.

The low order 16 bits of the microprocessor data bus 212 are also coupled through a bidirectional buffer 230 to a 16-bit LAN data bus 232. A LAN controller chip 234, such as the Am7990 LANCE Ethernet controller manufactured by Advanced Micro Devices, Inc. Sunnyvale, CA., interfaces the LAN data bus 232 with the first Ethernet 122a shown in Fig. 2. Control and data for the LAN controller 234 are stored in a 512K byte LAN memory 236, which is also connected to the LAN data bus 232. A specialized 16 to 32 bit FIFO chip 240, referred to herein as a parity FIFO chip and described below, is also connected to the LAN data bus 232. Also connected to the LAN data bus 232 is a LAN DMA controller 242, which controls movements of packets of data between the LAN memory 236 and the FIFO chip 240. The LAN DMA controller 242 may be a Motorola M68440 DMA controller using channel zero only.

The second Ethernet 122b shown in Fig. 2 connects to a second LAN data bus 252 on the network controller card 110a shown in Fig. 3. The LAN data bus 252 connects to the low order 16 bits of the microprocessor data bus 212 via a bidirectional buffer 250, and has similar components to those appearing on the LAN data bus 232. In particular, a LAN controller 254 interfaces the LAN data bus 252 with the Ethernet 122b, using LAN memory 256 for data and control,. and a LAN DMA controller 262 controls DMA transfer of data between the LAN memory 256 and the 16-bit wide data port A of the parity FIFO 260.

The low order 16 bits of microprocessor data bus 212 are also connected directly to another parity FIFO 270, and also to a control port of a VME/FIFO DMA controller 272. The FIFO 270 is used for passing messages between the CPU memory 214 and one of the remote boards 110, 112, 114, 116 or 118 (Fig. 2) in a manner described below. The VME/FIFO DMA controller 272, which supports three round-robin non-prioritized channels for copying data, controls all data transfers between one of the remote boards and any of the FIFOs 240, 260 or 270, as well as between the FIFOs 240 and 260.

32-bit data bus 274, which is connected to the 32-bit port B of each of the FIFOs 240, 260 and 270, is the data bus over which these transfers take place. Data bus 274 communicates with a local 32-bit bus 276 via a bidirectional pipelining latch 278, which is also controlled by VME/FIFO DMA controller 727, which in turn communicates with the VME bus 120 via a bidirectional buffer 280.

The local data bus 276 is also connected to a set of control registers 282, which are directly addressable across the VME bus 120. The registers 282 are used mostly for system initialization and diagnostics.

The local data bus 276 is also coupled to the microprocessor data bus 212 via a bidirectional buffer 284. When the NC 110a operates in slave mode, the CPU memory 214 is directly addressable from VME bus 120. One of the remote boards can copy data directly from the CPU memory 214 via the bidirectional buffer 284. LAN memories 236 and 256 are not directly addressed over VME bus 120.

The parity FIFOs 240, 260 and 270 each consist of an ASIC, the functions and operation of which are described in the Appendix. The FIFOs 240 and 260 are configured for packet data transfer and the FIFO 270 is configured for massage passing. Referring to the Appendix, the FIFOs 240 and 260 are programmed with the following bit settings in the Data Transfer Configuration Register:

Bit	Definition	Setting
0	WD Mode	N/A
1	Parity Chip	N/A
2	Parity Correct Mode	N/A
3	8/16 bits CPU & PortA interface	16 bits(1)
4	Invert Port A address 0	no (0)
5	Invert Port A address 1	yes (1)
6	Checksum Carry Wrap	yes (1)
7	Reset	no (0)

The Data Transfer Control Register is programmed as follows:

Bit	Definition	Setting
0	Enable PortA Req/Ack	yes (1)
1	Enable PortB Req/Ack	yes (1)
2	Data Transfer Direction	(as desired)
3	CPU parity enable	no (0)
4	PortA parity enable	no (0)
5	PortB parity enable	no (0)
6	Checksum Enable	yes (1)
7	PortA Master	yes (1)

Unlike the configuration used on FIFOs 240 and 260, the microprocessor 210 is responsible for loading and unloading Port A directly. The microprocessor 210 reads an entire 32-bit word from port A with a single instruction using two port A access cycles. Port A data transfer is disabled by unsetting bits 0 (Enable PortA Req/Ack) and 7 (PortA Master) of the Data Transfer Control Register.

The remainder of the control settings in FIFO 270 are the same as those in FIFOs 240 and 260 described above.

The NC 110a also includes a command FIFO 290. The command FIFO 290 includes an input port coupled to the local data bus 276, and which is directly addressable across the VME bus 120, and includes an output port connected to the microprocessor data bus 212. As explained in more detail below, when one of the remote boards issues a command or response to the NC 110a, it does so by directly writing a 1-word (32-bit) message descriptor into NC 110a's command FIFO 290. Command FIFO 290 generates a "FIFO not empty" status to the microprocessor 210, which then reads the message descriptor off the top of FIFO 290 and processes it. If the message is a command, then it includes a VME address at which the message is located (presumably an address in a shared memory similar to 214 on one of the remote boards). The microprocessor 210 then programs the FIFO 270 and the VME/FIFO DMA controller 272 to copy the message from the remote location into the CPU memory 214.

Command FIFO 290 is a conventional two-port FIFO, except that additional circuitry is included for generating a Bus Error signal on VME bus 120 if an attempt is made to write to the data input port while the FIFO is full. Command FIFO 290 has space for 256 entries.

A noteworthy feature of the architecture of NC 110a is that the LAN buses 232 and 252 are independent of the microprocessor data bus 212. Data packets being routed to or from an Ethernet are stored in LAN memory 236 on the LAN data bus 232 (or 256 on the LAN data bus 252), and not in the CPU memory 214. Data transfer between the LAN memories 236 and 256 and the Ethernets 122a and 122b, are controlled by LAN controllers 234 and 254, respectively, while most data transfer between LAN memory 236 or 256 and a remote port on the VME bus 120 are controlled by LAN DMA controllers 242 and 262, FIFOs 240 and 260, and VME/FIFO DMA controller 272. An exception to this rule occurs when the size of the data transfer is small, e.g., less than 64 bytes, in which case microprocessor 210 copies it directly without using DMA. The microprocessor 210 is not involved in larger transfers except in initiating them and in receiving notification when they are complete.

The CPU memory 214 contains mostly instructions for microprocessor 210, messages being transmitted to or from a remote board via FIFO 270, and various data blocks for controlling the FIFOs, the DMA controllers and the LAN controllers. The microprocessor 210 accesses the data packets in the LAN memories 236 and 256 by directly addressing them through the bidirectional buffers 230 and 250, respectively, for protocol processing. The local high-speed static RAM in CPU memory 214 can therefore provide zero wait state memory access for microprocessor 210 independent of network traffic. This is in sharp contrast to the prior art architecture shown in Fig. 1, in which all data and data packets, as well as microprocessor instructions for host CPU card 10, reside in the memory 16 and must communicate with the host CPU card 10 via the MMU 11.

While the LAN data buses 232 and 252 are shown as separate buses in Fig. 3, it will be understood that they may instead be implemented as a single combined bus.

NETWORK CONTROLLER OPERATION

In operation, when one of the LAN controllers (such as 234) receives a packet of information over its Ethernet 122a, it reads in the entire packet and stores it in corresponding LAN memory 236. The LAN controller 234 then issues an interrupt to microprocessor 210 via MFP 224, and the microprocessor 210 examines the status register on LAN controller 234 (via bidirectional buffer 230) to determine that the event causing the interrupt was a "receive packet completed." In order to avoid a potential lockout of the second Ethernet 122b caused by the prioritized interrupt handling characteristic of MFP 224, the microprocessor 210 does not at this time immediately process the received packet; instead, such processing is scheduled for a polling function.

When the polling function reaches the processing of the received packet, control over the packet is passed to a software link level receive module. The link level receive module then decodes the packet according to either of two different frame formats: standard Ethernet format or SNAP (IEEE 802 LCC) format. An entry in the header in the packet specifies which frame format was used. The link level driver then determines which of three types of messages is contained in the received packet: (1) IP, (2) ARP packets which can be handled by a local ARP module, or (3) ARP packets and other packet types which must be forwarded to the local host 118 (Fig. 2) for processing. If the packet is an ARP packet which can be handled by the NC 110a, such as a request for the address of server 100, then the microprocessor 210 assembles a response packet in LAN memory 236 and, in a conventional manner, causes LAN controller 234 to transmit that packet back over Ethernet 122a. It is noteworthy that the data manipulation for accomplishing this task is performed almost completely in LAN memory 236, directly addressed by microprocessor 210 as controlled by instructions in CPU memory 214. The function is accomplished also without generating any traffic on the VME backplane 120 at all, and without disturbing the local host 118.

If the received packet is either an ARP packet which cannot be processed completely in the NC 110a, or is another type of packet which requires delivery to the local host 118 (such as a client request for the server 100 to execute a client-defined procedure), then the microprocessor 210 programs LAN DMA controller 242 to load the packet from LAN memory 236 into FIFO 240, programs FIFO 240 with the direction of data transfer, and programs DMA controller 272 to read the packet out of FIFO 240 and across the VME bus 120 into system memory 116. In particular, the microprocessor 210 first programs the LAN DMA controller 242 with the starting address and length of the packet in LAN memory 236, and programs the controller to begin transferring data from the LAN memory 236 to port A of parity FIFO 240 as soon as the FIFO is ready to receive data. Second, microprocessor 210 programs the VME/FIFO DMA controller 272 with the destination address in system memory 116 and the length of the data packet, and instructs the controller to begin transferring data from port B of the FIFO 260 onto VME bus 120. Finally, the microprocessor 210 programs FIFO 240 with the direction of the transfer to take place. The transfer then proceeds entirely under the control of DMA controllers 242 and 272, without any further involvement by microprocessor 210.

The microprocessor 210 then sends a message to host 118 that a packet is available at a specified system memory address. The microprocessor 210 sends such a message by writing a message descriptor to a software-emulated command FIFO on the host, which copies the message from CPU memory 214 on the NC via buffer 284 and into the host's local memory, in ordinary VME block transfer mode. The host then copies the packet from system memory 116 into the host's own local memory using ordinary VME transfers.

If the packet received by NC 110a from the network is an IP packet, then the microprocessor 210 determines whether it is (1) an IP packet for the server 100 which is not an NFS packet; (2) an IP packet to be routed to a different network; or (3) an NFS packet. If it is an IP packet for the server 100, but not an NFS packet, then the microprocessor 210 causes the packet to be transmitted from the LAN memory 236 to the host 118 in the same manner described above with respect to certain ARP packets.

If the IP packet is not intended for the server 100, but rather is to be routed to a client on a different network, then the packet is copied into the LAN memory associated with the Ethernet to which the destination client is connected. If the destination client is on the Ethernet 122b, which is on the same NC board as the source Ethernet 122a, then the microprocessor 210 causes the packet to be copied from LAN memory 236 into LAN 256 and then causes LAN controller 254 to transmit it over Ethernet 122b. (Of course, if the two LAN data buses 232 and 252 are combined, then copying would be unnecessary; the microprocessor 210 would simply cause the LAN controller 254 to read the packet out of the same locations in LAN memory to which the packet was written by LAN controller 234.)

The copying of a packet from LAN memory 236 to LAN memory 256 takes place similarly to the copying described above from LAN memory to system memory. For transfer sizes of 64 bytes or more, the microprocessor 210 first programs the LAN DMA controller 242 with the starting address and length of the packet in LAN memory 236, and programs the controller to begin transferring data from the LAN memory 236 into port A of parity FIFO 240 as soon as the FIFO is ready to receive data. Second, microprocessor 210 programs the LAN DMA controller 262 with a destination address in LAN memory 256 and the length of the data packet, and instructs that controller to transfer data from parity FIFO 260 into the LAN memory 256. Third, microprocessor 210 programs the VME/FIFO DMA controller 272 to clock words of data out of port B of the FIFO 240, over the data bus 274, and into port B of FIFO 260. Finally, the microprocessor 210 programs the two FIFOs 240 and 260 with the direction of the transfer to take place. The transfer then proceeds entirely under the control of DMA controllers 242, 262 and 272, without any further involvement by the microprocessor 210. Like the copying from LAN memory to system memory, if the transfer size is smaller than 64 bytes, the microprocessor 210 performs the transfer directly, without DMA.

When each of the LAN DMA controllers 242 and 262 complete their work, they so notify microprocessor 210 by a respective interrupt provided through MFP 224. When the microprocessor 210 has received both interrupts, it programs LAN controller 254 to transmit the packet on the Ethernet 122b in a conventional manner.

Thus, IP routing between the two Ethernets in a single network controller 110 takes place over data bus 274, generating no traffic over VME bus 120. Nor is the host processor 118 disturbed for such routing, in contrast to the prior art architecture of Fig. 1. Moreover, all but the shortest copying work is performed by controllers outside microprocessor 210, requiring the involvement of the microprocessor 210, and bus traffic on microprocessor data bus 212, only for the supervisory functions of programming the DMA controllers and the parity FIFOs and instructing them to begin. The VME/FIFO DMA controller 272 is programmed by loading control registers via microprocessor data bus 212; the LAN DMA controllers 242 and 262 are programmed by loading control registers on the respective controllers via the microprocessor data bus 212, respective bidirectional buffers 230 and 250, and respective LAN data buses 232 and 252, and the parity FIFOs 240 and 260 are programmed as set forth in the Appendix.

If the destination workstation of the IP packet to be routed is on an Ethemet connected to a different one of the network controllers 110, then the packet is copied into the appropriate LAN memory on the NC 110 to which that Ethernet is connected. Such copying is accomplished by first copying the packet into system memory 116, in the manner described above with respect to certain ARP packets, and then notifying the destination NC that a packet is available. When an NC is so notified, it programs its own parity FIFO and DMA controllers to copy the packet from system memory 116 into the appropriate LAN memory. It is noteworthy that though this type of IP routing does create VME bus traffic, it still does not involve the host CPU 118.

If the IP packet received over the Ethernet 122a and now stored in LAN memory 236 is an NFS packet intended for the server 100, then the microprocessor 210 performs all necessary protocol preprocessing to extract the NFS message and convert it to the local NFS (LNFS) format. This may well involve the logical concatenation of data extracted from a large number of individual IP packets stored in LAN memory 236, resulting in a linked list, in CPU memory 214, pointing to the different blocks of data in LAN memory 236 in the correct sequence.

The exact details of the LNFS format are not important for an understanding of the invention, except to note that it includes commands to maintain a directory of files which are stored on the disks attached to the storage processors 114, commands for reading and writing data to and from a file on the disks, and various configuration management and diagnostics control messages. The directory maintenance commands which are supported by LNFS include the following messages based on conventional NFS: get attributes of a file (GETATTR); set attributes of a file (SETATTR); look up a file (LOOKUP); created a file (CREATE); remove a file (REMOVE); rename a file (RENAME); created a new linked file (LINK); create a symlink (SYMLINK); remove a directory (RMDIR); and return file system statistics (STATFS). The data transfer commands supported by LNFS include read from a file (READ); write to a file (WRITE); read from a directory (READDIR); and read a link (READLINK). LNFS also supports a buffer release command (RELEASE), for notifying the file controller that an NC is finished using a specified buffer in system memory. It also supports a VOP-derived access command, for determining whether a given type access is legal for specified credential on a specified file.

If the LNFS request includes the writing of file data from the LAN memory 236 to disk, the NC 110a first requests a buffer in system memory 116 to be allocated by the appropriate FC 112. When a pointer to the buffer is returned, microprocessor 210 programs LAN DMA controller 242, parity FIFO 240 and VME/FIFO DMA controller 272 to transmit the entire block of file data to system memory 116. The only difference between this transfer and the transfer described above for transmitting IP packets and ARP packets to system memory 116 is that these data blocks will typically have portions scattered throughout LAN memory 236. The microprocessor 210 accommodates that situation by programming LAN DMA controller 242 successively for each portion of the data, in accordance with the linked list, after receiving notification that the previous portion is complete. The microprocessor 210 can program the parity FIFO 240 and the VME/FIFO DMA controller 272 once for the entire message, as long as the entire data block is to be placed contiguously in system memory 116. If it is not, then the microprocessor 210 can program the DMA controller 272 for successive blocks in the same manner LAN DMA controller 242.

If the network controller 110a receives a message from another processor in server 100, usually from file controller 112, that file data is available in system memory 116 for transmission on one of the Ethernets, for example Ethernet

122a, then the network controller 110a copies the file data into LAN memory 236 in a manner similar to the copying of file data in the opposite direction. In particular, the microprocessor 210 first programs VME/FIFO DMA controller 272 with the starting address and length of the data in system memory 116, and programs the controller to begin transferring data over the VME bus 120 into port B of parity FIFO 240 as soon as the FIFO is ready to receive data. The microprocessor 210 then programs the LAN DMA controller 242 with a destination address in LAN memory 236 and then length of the file data, and instructs that controller to transfer data from the parity FIFO 240 into the LAN memory 236. Third, microprocessor 210 programs the parity FIFO 240 with the direction of the transfer to take place. The transfer then proceeds entirely under the control of DMA controllers 242 and 272, without any further involvement by the microprocessor 210. Again, if the file data is scattered in multiple blocks in system memory 116, the microprocessor 210 programs the VME/FIFO DMA controller 272 with a linked list of the blocks to transfer in the proper order.

When each of the DMA controllers 242 and 272 complete their work, they so notify microprocessor 210 through MFP 224. The microprocessor 210 then performs all necessary protocol processing on the LNFS message in LAN memory 236 in order to prepare the message for transmission over the Ethernet 122a in the form of Ethemet IP packets. As set forth above, this protocol processing is performed entirely in network controller 110a, without any involvement of the local host 118.

It should be noted that the parity FIFOs are designed to move multiples of 128-byte blocks most efficiently. The data transfer size through port B is always 32-bits wide, and the VME address corresponding to the 32-bit data must be quad-byte aligned. The data transfer size for port A can be either 8 or 16 bits. For bus utilization reasons, it is set to 16 bits when the corresponding local start address is double-byte aligned, and is set at 8 bits otherwise. The TCP/IP checksum is always computed in the 16 bit mode. Therefore, the checksum word requires byte swapping if the local start address is not double-byte aligned.

Accordingly, for transfer from port B to port A of any of the FIFOs 240, 260 or 270, the microprocessor 210 programs the VME/FIFO DMA controller to pad the transfer count to the next 128-byte boundary. The extra 32-bit word transfers do not involve the VME bus, and only the desired number of 32-bit words will be unloaded from port A.

For transfers from port A to port B of the parity FIFO 270, the microprocessor 210 loads port A word-by-word and forces a FIFO full indication when it is finished. The FIFO full indication enables unloading from port B. The same procedure also takes place for transfers from port A to port B of either of the parity FIFOs 240 or 260, since transfers of fewer than 128 bytes are performed under local microprocessor control rather than under the control of LAN DMA controller 242 or 262. For all of the FIFOs, the VME/FIFO DMA controller is programmed to unload only the desired number of 32-bit words.

FILE CONTROLLER HARDWARE ARCHITECTURE

The file controllers (FC) 112 may each be a standard off-the-shelf microprocessor board, such as one manufactured by Motorola Inc. Preferably, however, a more specialized board is used such as that shown in block diagram form in Fig. 4.

Fig. 4 shows one of the FCs 112a, and it will be understood that the other FC can be identical. In many aspects it is simply a scaled-down version of the NC 110a shown in Fig. 3, and in some respects it is scaled up. Like the NC 110a, FC 112a comprises a 20MHz 68020 microprocessor 310 connected to a 32-bit microprocessor data bus 312. Also connected to the microprocessor data bus 312 is a 256K byte shared CPU memory 314. The low order 8 bits of the microprocessor data bus 312 are connected through a bidirectional buffer 316 to an 8-bit slow-speed data bus 318. On slow-speed data bus 318 are a 128K byte PROM 320, and a multifunction peripheral (MFP) 324. The functions of the PROM 320 and MFP 324 are the same as those described above with respect to EPROM 220 and MFP 224 on NC 110a. FC 112a does not include PROM like the PROM 222 on NC 110a, but does include a parallel port 392. The parallel port 392 is mainly for testing and diagnostics.

Like the NC 110a, the FC 112a is connected to the VME bus 120 via a bidirectional buffer 380 and a 32-bit local data bus 376. A set of control registers 382 are connected to the local data bus 376, and directly addressable across the VME bus 120. The local data bus 376 is also coupled to the microprocessor data bus 312 via a bidirectional buffer 384. This permits the direct addressability of CPU memory 314 from VME bus 120.

FC 112a also includes a command FIFO 390, which includes an input port coupled to the local data bus 376 and which is directly addressable across the VME bus 120. The command FIFO 390 also includes an output port connected to the microprocessor data bus 312. The structure, operation and purpose of command FIFO 390 are the same as those described above with respect to command FIFO 290 on NC 110a.

The FC 112a omits the LAN data buses 323 and 352 which are present in NC 110a, but instead includes a 4 megabyte 32-bit wide FC memory 396 coupled to the microprocessor data bus 312 via a bidirectional buffer 394. As will be seen, FC memory 396 is used as a cache memory for file control information, separate from the file data information cached in system memory 116.

The file controller embodiment shown in Fig. 4 does not include any DMA controllers, and hence cannot act as a master for transmitting or receiving data in any block transfer mode, over the VME bus 120. Block transfers do occur with the CPU memory 314 and the FC memory 396, however, with the FC 112a acting as an VME bus slave. In such transfers, the remote master addresses the CPU memory 314 or the FC memory 396 directly over the VME bus 120 through the bidirectional buffers 384 and, if appropriate, 394.

FILE CONTROLLER OPERATION

The purpose of the FC 112a is basically to provide virtual file system services in response to requests provided in LNFS format by remote processors on the VME bus 120. Most requests will come from a network controller 110, but requests may also come from the local host 118.

The file related commands supported by LNFS are identified above. They are all specified to the FC 112a in terms of logically identified disk data blocks. For example, the LNFS command for reading data from a file includes a specification of the file from which to read (file system ID (FSID) and file ID (inode)), a byte offset, and a count of the number of bytes to read. The FC 112a converts that identification into physical form, namely disk and sector numbers, in order to satisfy the command.

The FC 112a runs a conventional Fast File System (FFS or UFS), which is based on the Berkeley 4.3 VAX release. This code performs the conversion and also performs all disk data caching and control data caching. However, as previously mentioned, control data caching is performed using the FC memory 396 on FC 112a, whereas disk data caching is performed using the system memory 116 (Fig. 2). Caching this file control information within the FC 112a avoids the VME bus congestion and speed degradation which would result if file control information was cached in system memory 116. The memory on the FC 112a is directly accessed over the VME bus 120 for three main purposes. First, and by far the most frequent, are accesses to FC memory 396 by an SP 114 to read or write cached file control information. These are accesses requested by FC 112a to write locally modified file control structures through to disk, or to read file control structures from disk. Second, the FC's CPU memory 314 is accessed directly by other processors for message transmissions from the FC 112a to such other processors. For example, if a data block in system memory is to be transferred to an SP 114 for writing to disk, the FC 112a first assembles a message in its local memory 314 requesting such a transfer. The FC 112a then notifies the SP 114, which copies the message directly from the CPU memory 314 and executes the requested transfer.

A third type of direct access to the FC's local memory occurs when an LNFS client reads directory entries. When FC 112a receives an LNFS request to read directory entries, the FC 112a formats the requested directory entries in FC memory 396 and notifies the requestor of their location. The requestor then directly accesses FC memory 396 to read the entries.

The version of the UFS code on FC 112a includes some modifications in order to separate the two caches. In particular, two sets of buffer headers are maintained, one for the FC memory 396 and one for the system memory 116. Additionally, a second set of the system buffer routines (GETBLK(), BRELSE(), BREAD(), BWRITE(), and BREADA ()) exist, one for buffer accesses to FC Mem 396 and one for buffer accesses to system memory 116. The UFS code is further modified to call the appropriate buffer routines for FC memory 396 for accesses to file control information, and to call the appropriate buffer routines for the system memory 116 for the caching of disk data. A description of UFS may be found in chapters 2, 6, 7 and 8 of "Kernel Structure and Flow," by Rieken and Webb of .sh consulting (Santa Clara, California: 1988).

When a read command is sent to the FC by a requestor such as a network controller, the FC first converts the file, offset and count information into disk and sector information. It then locks the system memory buffers which contain that information, instructing the storage processor 114 to read them from disk if necessary. When the buffer is ready, the FC returns a message to the requestor containing both the attributes of the designated file and an array of buffer descriptors that identify the locations in system memory 116 holding the data.

After the requestor has read the data out of the buffers, it sends a release request back to the FC. The release request is the same message that was returned by the FC in response to the read request; the FC 112a uses the information contained therein to determine which buffers to free.

A write command is processed by FC 112a similarly to the read command, but the caller is expected to write to (instead of read from) the locations in system memory 116 identified by the buffer descriptors returned by the FC 112a. Since FC 112a employs write-through caching, when it receives the release command from the requestor, it instructs storage processor 114 to copy the data from system memory 116 onto the appropriate disk sectors before freeing the system memory buffers for possible reallocation.

The READDIR transaction is similar to read and write, but the request is satisfied by the FC 112a directly out of its own FC memory 396 after formatting the requested directory information specifically for this purpose. The FC 112a causes the storage processor read the requested directory information from disk if it is not already locally cached. Also, the specified offset is a "magic cookie" instead of a byte offset, identifying directory entries instead of an absolute byte offset into the file. No file attributes are returned.

The READLINK transaction also returns no file attributes, and since links are always read in their entirety, it does not require any offset or count.

For all of the disk data caching performed through system memory 116, the FC 112a acts as a central authority for dynamically allocating, deallocating and keeping track of buffers. If there are two or more FCs 112, each has exclusive control over its own assigned portion of system memory 116. In all of these transactions, the requested buffers are locked during the period between the initial request and the release request. This prevents corruption of the data by other clients.

Also in the situation where there are two or more FCs, each file system on the disks is assigned to a particular one of the FCs. FC #0 runs a process called FC_VICE_PRESIDENT, which maintains a list of which file systems are assigned to which FC. When a client processor (for example an NC 110) is about to make an LNFS request designating a particular file system, it first sends the fsid in a message to the FC_VICE_PRESIDENT asking which FC controls the specified file system. The FC_VICE_PRESIDENT responds, and the client processor sends the LNFS request to the designated FC. The client processor also maintains its own list of fsid/FC pairs as it discovers them, so as to minimize the number of such requests to the FC_VICE_PRESIDENT.

STORAGE PROCESSOR HARDWARE ARCHITECTURE

In the file server 100, each of the storage processors 114 can interface the VME bus 120 with up to 10 different SCSI buses. Additionally, it can do so at the full usage rate of an enhanced block transfer protocol of 55MB per second.

Fig. 5 is a block diagram of one of the SPs 114a. SP 114b is identical. SP 114a comprises a microprocessor 510, which may be a Motorola 68020 microprocessor operating at 20MHz. The microprocessor 510 is coupled over a 32-bit microprocessor data bus 512 with CPU memory 514, which may include up to 1MB of static RAM. The microprocessor 510 accesses instructions, data and status on its own private bus 512, with no contention from any other source. The microprocessor 510 is the only master of bus 512.

The low order 16 bits of the microprocessor data bus 512 interface with a control bus 516 via a bidirectional buffer 518. The low order 8 bits of the control bus 516 interface with a slow speed bus 520 via another bidirectional buffer 522. The slow speed bus 520 connects to an MFP 524, similar to the MFP 224 in NC 110a (Fig. 3), and with a PROM 526, similar to PROM 220 on NC 110a. The PROM 526 comprises 128K bytes of EPROM which contains the functional code for SP 114a. Due to the width and speed of the EPROM 526, the functional code is copied to CPU memory 514 upon reset for faster execution.

MFP 524, like the MFP 224 on NC 110a, comprises a Motorola 68901 multifunction peripheral device. It provides the functions of a vectored interrupt controller, individually programmable I/O pins, four timers and a UART. The UART functions provide serial communications across an RS 232 bus (not shown in Fig. 5) for debug monitors and diagnostics. Two of the four timing functions may be used as general-purpose timers by the microprocessor 510, either independently or in cascaded fashion. A third timer function provides the refresh clock for a DMA controller described below, and the fourth timer generates the UART clock. Additional information on the MFP 524 can be found in "MC 68901 Multi-Function Peripheral Specification," by Motorola, Inc. The eight general-purpose I/O bits provided by MFP 524 are configured according to the following table:

Bit	Direction	Definition
7	input	Power Failure is Imminent - This functions as an early warning.
6	input	SCSI Attention - A composite of the SCSI. Attentions from all 10 SCSI channels.
5	input	Channel Operation Done - A composite of the channel done bits from all 13 channels of the DMA controller, described below.
4	output	DMA Controller Enable. Enables the DMA Controller to run.
3	input	VMEbus Interrupt Done - Indicates the completion of a VMEbus Interrupt.
2	input	Command Available - Indicates that the SP'S Command Fifo, described below, contains one or more command pointers.
1	output	External Interrupts Disable. Disables externally generated interrupts to the microprocessor 510.

(continued)

Bit	Direction	Definition
0	output	Command Fifo Enable. Enables operation of the SP'S Command Fifo. Clears the Command Fifo when reset.

Commands are provided to the SP 114a from the VME bus 120 via a bidirectional buffer 530, a local data bus 532, and a command FIFO 534. The command FIFO 534 is similar to the command FIFOs 290 and 390 on NC 110a and FC 112a, respectively, and has a depth of 256 32-bit entries. The command FIFO 534 is a write-only register as seen on the VME bus 120, and as a read-only register as seen by microprocessor 510. If the FIFO is full at the beginning of a write from the VME bus, a VME bus error is generated. Pointers are removed from the command FIFO 534 in the order received, and only by the microprocessor 510. Command available status is provided through I/O bit 4 of the MFP 524, and as a long as one or more command pointers are still within the command FIFO 534, the command available status remains asserted.

As previously mentioned, the SP 114a supports up to 10 SCSI buses or channels 540a-540j. In the typical configuration, buses 540a-540i support up to 3 SCSI disk drives each, and channel 540j supports other SCSI peripherals such as tape drives, optical disks, and so on. Physically, the SP 114a connects to each of the SCSI buses with an ultra-miniature D sub connector and round shielded cables. Six 50-pin cables provide 300 conductors which carry 18 signals per bus and 12 grounds. The cables attach at the front panel of the SP 114a and to a commutator board at the disk drive array. Standard 50-pin cables connect each SCSI device to the commutator board. Termination resistors are installed on the SP 114a.

The SP 114a supports synchronous parallel data transfers up to 5MB per second on each of the SCSI buses 540, arbitration, and disconnect/reconnect services. Each SCSI bus 540 is connected to a respective SCSI adaptor 542, which in the present embodiment is an AIC 6250 controller IC manufactured by Adaptec Inc., Milpitas, Califomia, operating in the non-multiplexed address bus mode. The AIC 6250 is described in detail in "AIC-6250 Functional Specification," by Adaptec Inc. The SCSI adaptors 542 each provide the necessary hardware interface and low-level electrical protocol to implement its respective SCSI channel.

The 8-bit data port of each of the SCSI adaptors 542 is connected to port A of a respective one of a set of ten parity FIFOs 544a-544j. The FIFOs 544 are the same as FIFOs 240, 260 and 270 on NC 110a, and are connected and configured to provide parity covered data transfers between the 8-bit data port of the respective SCSI adaptors 542 and a 36-bit (32-bit plus 4 bits of parity) common data bus 550. The FIFOs 544 provide handshake, status, word assembly/disassembly and speed matching FIFO buffering for this purpose. The FIFOs 544 also generate and check parity for the 32-bit bus, and for RAID 5 implementations they accumulate and check redundant data and accumulate recovered data.

All of the SCSI adaptors 542 reside at a single location of the address space of the microprocessor 510, as do all of the parity FIFOs 544. The microprocessor 510 selects individual controllers and FIFOs for access in pairs, by first programming a pair select register (not shown) to point to the desired pair and then reading from or writing to the control register address of the desired chip in the pair. The microprocessor 510 communicates with the control registers on the SCSI adaptors 542 via the control bus 516 and an additional bidirectional buffer 546, and communicates with the control registers on FIFOs 544 via the control bus 516 and a bidirectional buffer 552. Both the SCSI adaptors 542 and FIFOs 544 employ 8-bit control registers, and register addressing of the FIFOs 544 is arranged such that such registers alias in consecutive byte locations. This allows the microprocessor 510 to write to the registers as a single 32-bit register, thereby reducing instruction overhead.

The parity FIFOs 544 are each configured in their Adaptec 6250 mode. Referring to the Appendix, the FIFOs 544 are programmed with the following bit settings in the Data Transfer Configuration Register:

Bit	Definition	Setting
0	WD Mode	(0)
1	Parity Chip	(1)
2	Parity Correct Mode	(0)
3	8/16 bits CPU & PortA interface	(0)
4	Invert Port A address 0	(1)
5	Invert Port A address 1	(1)
6	Checksum Carry Wrap	(0)
7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

Bit	Definition	Setting
0	Enable PortA Req/Ack	(1)
1	Enable PortB Req/Ack	(1)
2	Data Transfer Direction	as desired
3	CPU parity enable	(0)
4	PortA parity enable	(1)
5	PortB parity enable	(1)
6	Checksum Enable	(0)
7	PortA Master	(0)

In addition, bit 4 of the RAM Access Control Register (Long Burst) is programmed for 8-byte bursts.

SCSI adaptors 542 each generate a respective interrupt signal, the status of which are provided to microprocessor 510 as 10 bits of a 16-bit SCSI interrupt register 556. The SCSI interrupt register 556 is connected to the control bus 516. Additionally, a composite SCSI interrupt is provided through the MFP 524 whenever any one of the SCSI adaptors 542 needs servicing.

An additional parity FIFO 554 is also provided in the SP 114a, for message passing. Again referring to the Appendix, the parity FIFO 554 is programmed with the following bit settings in the Data Transfer Configuration Register:

Bit	Definition	Setting
0	WD Mode	(0)
1	Parity Chip	(1)
2	Parity Correct Mode	(0)
3	8/16 bits CPU & PortA interface	(1)
4	Invert Port A address 0	(1)
5	Invert Port A address 1	(1)
6	Checksum Carry Wrap	(0)
7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

Bit	Definition	Setting
0	Enable PortA Req/Ack	(0)
1	Enable PortB Req/Ack	(1)
2	Data Transfer Direction	as desired
3	CPU parity enable	(0)
4	PortA parity enable	(0)
5	PortB parity enable	(1)
6	Checksum Enable	(0)
7	PortA Master	(0)

In addition, bit 4 of the RAM Access Control Register (Long Burst) is programmed for 8-byte bursts.

Port A of FIFO 554 is connected to the 16-bit control bus 516, and port B is connected to the common data bus 550. FIFO 554 provides one means by which the microprocessor 510 can communicate directly with the VME bus 120, as is described in more detail below.

The microprocessor 510 manages data movement using a set of 15 channels, each of which has an unique status which indicates its current state. Channels are implemented using a channel enable register 560 and a channel status register 562, both connected to the control bus 516. The channel enable register 560 is a 16-bit write-only register, whereas the channel status register 562 is a 16-bit read-only register. The two registers reside at the same address to microprocessor 510. The microprocessor 510 enables a particular channel by setting its respective bit in channel enable register 560, and recognizes completion of the specified operation by testing for a "done" bit in the channel status register 562. The microprocessor 510 then resets the enable bit, which causes the respective "done" bit in the channel status register 562 to be cleared.

The channels are defined as follows:

CHANNEL	FUNCTION
0:9	These channels control data movement to and from the respective FIFOs 544 via the common data bus 550. When a FIFO is enabled and a request is received from it, the channel becomes ready. Once the channel has been serviced a status of done is generated.
11:10	These channels control data movement between a local data buffer 564, described below, and the VME bus 120. When enabled the channel becomes ready. Once the channel has been serviced a status of done is generated.
12	When enabled, this channel causes the DRAM in local data buffer 564 to be refreshed based on a clock which is generated by the MFP 524. The refresh consists of a burst of 16 rows. This channel does not generate a status of done.
13	The microprocessor's communication FIFO 554 is serviced by this channel. When enable is set and the FIFO 554 asserts a request then the channel becomes ready. This channel generates a status of done.
14	Low latency writes from microprocessor 510 onto the VME bus 120 are controlled by this channel. When this channel is enabled data is moved from a special 32 bit register, described below, onto the VME bus 120. This channel generates a done status.
15	This is a null channel for which neither a ready status nor done status is generated.

Channels are prioritized to allow servicing of the more critical requests first. Channel priority is assigned in a descending order starting at channel 14. That is, in the event that all channels are requesting service, channel 14 will be the first one served.

The common data bus 550 is coupled via a bidirectional register 570 to a 36-bit junction bus 572. A second bidirectional register 574 connects the junction bus 572 with the local data bus 532. Local data buffer 564, which comprises 1MB of DRAM, with parity, is coupled bidirectionally to the junction bus 572. It is organized to provide 256K 32-bit words with byte parity. The SP 114a operates the DRAMs in page mode to support a very high data rate, which requires bursting of data instead of random single-word accesses. It will be seen that the local data buffer 564 is used to implement a RAID (redundant array of inexpensive disks) 'algorithm, and is not used for direct reading and writing between the VME bus 120 and a peripheral on one of the SCSI buses 540.

A read-only register 576, containing all zeros, is also connected to the junction bus 572. This register is used mostly for diagnostics, initialization, and clearing of large blocks of data in system memory 116.

The movement of data between the FIFOs 544 and 554, the local data buffer 564, and a remote entity such as the system memory 116 on the VME bus 120, is all controlled by a VME/FIFO DMA controller 580. The VME/FIFO DMA controller 580 is similar to the VME/FIFO DMA controller 272 on network controller 110a (Fig. 3), and is described in the Appendix. Briefly, it includes a bit slice engine 582 and a dual-port static RAM 584. One port of the dual-port static RAM 584 communicates over the 32-bit microprocessor data bus 512 with microprocessor 510, and the other port communicates over a separate 16-bit bus with the bit slice engine 582. The microprocessor 510 places command parameters in the dual-port RAM 584, and uses the channel enables 560 to signal the VME/FIFO DMA controller 580 to proceed with the command. The VME/FIFO DMA controller is responsible for scanning the channel status and servicing requests, and returning ending status in the dual-port RAM 584. The dual-port RAM 584 is organized as 1K x 32 bits at the 32-bit port and as 2K x 16 bits at the 16-bit port. A example showing the method by which the microprocessor 510 controls the VME/FIFO DMA controller 580 is as follows. First, the microprocessor 510 writes into the dual-port RAM 584 the desired command and associated parameters for the desired channel. For example, the command might be, "copy a block of data from FIFO 544h out into a block of system memory 116 beginning at a specified VME address." Second, the microprocessor sets the channel enable bit in channel enable register 560 for the desired channel.

At the time the channel enable bit is set, the appropriate FIFO may not yet be ready to send data. Only when the VME/FIFO DMA controller 580 does receive a "ready" status from the channel, will the controller 580 execute the command. In the meantime, the DMA controller 580 is free to execute commands and move data to or from other channels.

When the DMA controller 580 does receive a status of "ready" from the specified channel, the controller fetches the channel command and parameters from the dual-ported RAM 584 and executes. When the command is complete, for example all the requested data has been copied, the DMA controller writes status back into the dual-port RAM 584 and asserts "done" for the channel in channel status register 562. The microprocessor 510 is then interrupted, at which time it reads channel status register 562 to determine which channel interrupted. The microprocessor 510 then clears the channel enable for the appropriate channel and checks the ending channel status in the dual-port RAM 584.

In this way a high-speed data transfer can take place under the control of DMA controller 580, fully in parallel with other activities being performed by microprocessor 510. The data transfer takes place over busses different from microprocessor data bus 512, thereby avoiding any interference with microprocessor instruction fetches.

The SP 114a also includes a high-speed register 590, which is coupled between the microprocessor data bus 512 and the local data bus 532. The high-speed register 590 is used to write a single 32-bit word to an VME bus target with a minimum of overhead. The register is write only as viewed from the microprocessor 510. In order to write a word onto the VME bus 120, the microprocessor 510 first writes the word into the register 590, and the desired VME target address into dual-port RAM 584. When the microprocessor 510 enables the appropriate channel in channel enable register 560, the DMA controller 580 transfers the data from the register 590 into the VME bus address specified in the dual-port RAM 584. The DMA controller 580 then writes the ending status to the dual-port RAM and sets the channel "done" bit in channel status register 562.

This procedure is very efficient for transfer of a single word of data, but becomes inefficient for large blocks of data. Transfers of greater than one word of data, typically for message passing, are usually performed using the FIFO 554.

The SP 114a also includes a series of registers 592, similar to the registers 282 on NC 110a (Fig. 3) and the registers 382 on FC 112a (Fig. 4). The details of these registers are not important for an understanding of the present invention.

STORAGE PROCESSOR OPERATION

The 30 SCSI disk drives supported by each of the SPs 114 are visible to a client processor, for example one of the file controllers 112, either as three large, logical disks or as 30 independent SCSI drives, depending on configuration. When the drives are visible as three logical disks, the SP uses RAID 5 design algorithms to distribute data for each logical drive on nine physical drives to minimize disk arm contention. The tenth drive is left as a spare. The RAID 5 algorithm (redundant array of inexpensive drives, revision 5) is described in "A Case For a Redundant Arrays of Inexpensive Disks (RAID)", by Patterson et al., published at ACM SIGMOD Conference, Chicago, Ill., June 1-3, 1988.

In the RAID 5 design, disk data are divided into stripes. Data stripes are recorded sequentially on eight different disk drives. A ninth parity stripe, the exclusive-or of eight data stripes, is recorded on a ninth drive. If a stripe size is set to 8K bytes, a read of 8K of data involves only one drive. A write of 8K of data involves two drives: a data drive and a parity drive. Since a write requires the reading back of old data to generate a new parity stripe, writes are also referred to as modify writes. The SP 114a supports nine small reads to nine SCSI drives concurrently. When stripe size is set to 8K, a read of 64K of data starts all eight SCSI drives, with each drive reading one 8K stripe worth of data. The parallel operation is transparent to the caller client.

The parity stripes are rotated among the nine drives in order to avoid drive contention during write operations. The parity stripe is used to improve availability of data. When one drive is down, the SP 114a can reconstruct the missing data from a parity stripe. In such case, the SP 114a is running in error recovery mode. When a bad drive is repaired, the SP 114a can be instructed to restore data on the repaired drive while the system is on-line.

When the SP 114a is used to attach thirty independent SCSI drives, no parity stripe is created and the client addresses each drive directly.

The SP 114a processes multiple messages (transactions, commands) at one time, up to 200 messages per second. The SP 114a does not initiate any messages after initial system configuration. The following SP 114a operations are defined:

01	No Op
02	Send Configuration Data
03	Receive Configuration Data
05	Read and Write Sectors
06	Read and Write Cache Pages
07	IOCTL Operation
08	Dump SP 114a Local Data Buffer
09	Start/Stop A SCSI Drive

(continued)

0C	Inquiry
0E	Read Message Log Buffer
0F	Set SP 114a Interrupt

The above transactions are described in detail in the above-identified application entitled MULTIPLE FACILITY OPERATING SYSTEM ARCHITECTURE. For the understanding of the invention, it will be useful to describe the function and operation of only two of these commands: read and write sectors, and read and write cache pages.

Read and Write Sectors

This command, issued usually by an FC 112, causes the SP 114a to transfer data between a specified block of system memory and a specified series of contiguous sectors on the SCSI disks. As previously described in connection with the file controller 112, the particular sectors are identified in physical terms. In particular, the particular disk sectors are identified by SCSI channel number (0-9), SCSI ID on that channel number (0-2), starting sector address on the specified drive, and a count of the number of sectors to read or write. The SCSI channel number is zero if the SP 114a is operating under RAID 5.

The SP 114a can execute up to 30 messages on the 30 SCSI drives simultaneously. Unlike most of the commands to an SP 114, which are processed by microprocessor 510 as soon as they appear on the command FIFO 534, read and write sectors commands (as well as read and write cache memory commands) are first sorted and queued. Hence, they are not served in the order of arrival.

When a disk access command arrives, the microprocessor 510 determines which disk drive is targeted and inserts the message in a queue for that disk drive sorted by the target sector address. The microprocessor 510 executes commands on all the queues simultaneously, in the order present in the queue for each disk drive. In order to minimize disk arm movements, the microprocessor 510 moves back and forth among queue entries in an elevator fashion.

If no error conditions are detected from the SCSI disk drives, the command is completed normally. When a data check error condition occurs and the SP 114a is configured for RAID 5, recovery actions using redundant data begin automatically. When a drive is down while the SP 114a is configured for RAID 5, recovery actions similar to data check recovery take place.

Read/Write Cache Pages

This command is similar to read and write sectors, except that multiple VME addresses are provided for transferring disk data to and from system memory 116. Each VME address points to a cache page in system memory 116, the size of which is also specified in the command. When transferring data from a disk to system memory 116, data are scattered to different cache pages; when writing data to a disk, data are gathered from different cache pages in system memory 116. Hence, this operation is referred to as a scatter-gather function.

The target sectors on the SCSI disks are specified in the command in physical terms, in the same manner that they are specified for the read and write sectors command. Termination of the command with or without error conditions is the same as for the read and write sectors command.

The dual-port RAM 584 in the DMA controller 580 maintains a separate set of commands for each channel controlled by the bit slice engine 582. As each channel completes its previous operation, the microprocessor 510 writes a new DMA operation into the dual-port RAM 584 for that channel in order to satisfy the next operation on a disk elevator queue.

The commands written to the DMA controller 580 include an operation code and a code indicating whether the operation is to be performed in non-block mode, in standard VME block mode, or in enhanced block mode. The operation codes supported by DMA controller 580 are as follows:

OP CODE	OPERATION	
0	NO-OP	
1	ZEROES → BUFFER	Move zeros from zeros register 576 to local data buffer 564.
2	ZEROES → FIFO	Move zeros from zeros register 576 to the currently selected FIFO on common data bus 550.

(continued)

OP CODE	OPERATION	
3	ZEROES → VMEbus	Move zeros from zeros register 576 out onto the VME bus 120. Used for initializing cache buffers in system memory 116.
4	VMEbus → BUFFER	Move data from the VME bus 120 to the local data buffer 564. This operation is used during a write, to move target data intended for a down drive into the buffer for participation in redundancy generation. Used only for RAID 5 application.
5	VMEbus → FIFO	New data to be written from VME bus onto a drive. Since RAID 5 requires redundancy data to be generated from data that is buffered in local data buffer 564, this operation will be used only if the SP 114a is not configured for RAID 5.
6	VMEbus → BUFFER & FIFO	Target data is moved from VME bus 120 to a SCSI device and is also captured in the local data buffer 564 for participation in redundancy generation. Used only if SP 114a is configured for RAID 5 operation.
7	BUFFER → VMEbus	This operation is not used.
8	BUFFER → FIFO	Participating data is transferred to create redundant data or recovered data on a disk drive. Used only in RAID 5 applications.
9	FIFO → VMEbus	This operation is used to move target data directly from a disk drive onto the VME bus 120.
A	FIFO → BUFFER	Used to move participating data for recovery and modify operations. Used only in RAID 5 applications.
B	FIFO → VMEbus & BUFFER	This operation is used to save target data for participation in data recovery. Used only in RAID 5 applications.

SYSTEM MEMORY

Fig. 6 provides a simplified block diagram of the preferred architecture of one of the system memory cards 116a. Each of the other system memory cards are the same. Each memory card 116 operates as a slave on the enhanced VME bus 120 and therefore requires no on-board CPU. Rather, a timing control block 610 is sufficient to provide the necessary slave control operations. In particular, the timing control block 610, in response to control signals from the control portion of the enhanced VME bus 120, enables a 32-bit wide buffer 612 for an appropriate direction transfer of 32-bit data between the enhanced VME bus 120 and a multiplexer unit 614. The multiplexer 614 provides a multiplexing and demultiplexing function, depending on data transfer direction, for a six megabit by seventy-two bit word memory array 620. An error correction code (ECC) generation and testing unit 622 is also connected to the multiplexer 614 to generate or verify, again depending on transfer direction, eight bits of ECC data. The status of ECC verification is provided back to the timing control block 610.

ENHANCED VME BUS PROTOCOL

VME bus 120 is physically the same as an ordinary VME bus, but each of the NCs and SPs include additional circuitry and firmware for transmitting data using an enhanced VME block transfer protocol. The enhanced protocol is described in detail in the above-identified application entitled ENHANCED VMEBUS PROTOCOL UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA TRANSFER, and summarized in the Appendix hereto. Typically transfers of LNFS file data between NCs and system memory, or between SPs and system memory, and transfers of packets being routed from one NC to another through system memory, are the only types of transfers that use the enhanced protocol in server 100. All other data transfers on VME bus 120 use either conventional VME block transfer protocols or ordinary non-block transfer protocols.

MESSAGE PASSING

As is evident from the above description, the different processors in the server 100 communicate with each other via certain types of messages. In software, these messages are all handled by the messaging kernel, described in detail in the MULTIPLE FACILITY OPERATING SYSTEM ARCHITECTURE application cited above. In hardware, they are implemented as follows.

Each of the NCs 110, each of the FCs 112, and each of the SPs 114 includes a command or communication FIFO such as 290 on NC 110a. The host 118 also includes a command FIFO, but since the host is an unmodified purchased processor board, the FIFO is emulated in software. The write port of the command FIFO in each of the processors is directly addressable from any of the other processors over VME bus 120.

Similarly, each of the processors except SPs 114 also includes shared memory such as CPU memory 214 on NC 110a. This shared memory is also directly addressable by any of the other processors in the server 100.

If one processor, for example network controller 110a, is to send a message or command to a second processor, for example file controller 112a, then it does so as follows. First, it forms the message in its own shared memory (e.g., in CPU memory 214 on NC 110a). Second, the microprocessor in the sending processor directly writes a message descriptor into the command FIFO in the receiving processor. For a command being sent from network controller 110a to file controller 112a, the microprocessor 210 would perform the write via buffer 284 on NC 110a, VME bus 120, and buffer 384 on file controller 112a.

The command descriptor is a single 32-bit word containing in its high order 30 bits a VME address indicating the start of a quad-aligned message in the sender's shared memory. The low order two bits indicate the message type as follows:

Type	Description
0	Pointer to a new message being sent
1	Pointer to a reply message
2	Pointer to message to be forwarded
3	Pointer to message to be freed; also message acknowledgment

All messages are 128-bytes long.

When the receiving processor reaches the command descriptor on its command FIFO, it directly accesses the sender's shared memory and copies it into the receiver's own local memory. For a command issued from network controller 110a to file controller 112a, this would be an ordinary VME block or non-block mode transfer from NC CPU memory 214, via buffer 284, VME bus 120 and buffer 384, into FC CPU memory 314. The FC microprocessor 310 directly accesses NC CPU memory 214 for this purpose over the VME bus 120.

When the receiving processor has received the command and has completed its work, it sends a reply message back to the sending processor. The reply message may be no more than the original command message unaltered, or it may be a modified version of that message or a completely new message. If the reply message is not identical to the original command message, then the receiving processor directly accesses the original sender's shared memory to modify the original command message or overwrite it completely. For replies from the FC 112a to the NC 110a, this involves an ordinary VME block or non-block mode transfer from the FC 112a, via buffer 384, VME bus 120, buffer 284 and into NC CPU memory 214. Again, the FC microprocessor 310 directly accesses NC CPU memory 214 for this purpose over the VME bus 120.

Whether or not the original command message has been changed, the receiving processor then writes a reply message descriptor directly into the original sender's command FIFO. The reply message descriptor contains the same VME address as the original command message descriptor, and the low order two bits of the word are modified to indicate that this is a reply message. For replies from the FC 112a to the NC 110a, the message descriptor write is accomplished by microprocessor 310 directly accessing command FIFO 290 via buffer 384, VME bus 120 and buffer 280 on the NC. Once this is done, the receiving processor can free the buffer in its local memory containing the copy of the command message.

When the original sending processor reaches the reply message descriptor on its command FIFO, it wakes up the process that originally sent the message and permits it to continue. After examining the reply message, the original sending processor can free the original command message buffer in its own local shared memory.

As mentioned above, network controller 110a uses the buffer 284 data path in order to write message descriptors onto the VME bus 120, and uses VME/FIFO DMA controller 272 together with parity FIFO 270 in order to copy messages from the VME bus 120 into CPU memory 214. Other processors read from CPU memory 214 using the buffer 284 data path.

File controller 112a writes message descriptors onto the VME bus 120 using the buffer 384 data path, and copies messages from other processors' shared memory via the same data path. Both take place under the control of microprocessor 310. Other processors copy messages from CPU memory 314 also via the buffer 384 data path.

Storage processor 114a writes message descriptors onto the VME bus using high-speed register 590 in the manner described above, and copies messages from other processors using DMA controller 580 and FIFO 554. The SP 114a has no shared memory, however, so it uses a buffer in system memory 116 to emulate that function. That is, before it writes a message descriptor into another processor's command FIFO, the SP 114a first copies the message into its own previously allocated buffer in system memory 116 using DMA controller 580 and FIFO 554. The VME address included in the message descriptor then reflects the VME address of the message in system memory 116.

In the host 118, the command FIFO and shared memory are both emulated in software.

APPENDIX A

VME/FIFO DMA Controller

In storage processor 114a, DMA controller 580 manages the data path under the direction of the microprocessor 510. The DMA controller 580 is a microcoded 16-bit bit-slice implementation executing pipelined instructions at a rate of one each 62.5ns. It is responsible for scanning the channel status 562 and servicing request with parameters stored in the dual-ported ram 584 by the microprocessor 510. Ending status is returned in the ram 584 and interrupts are generated for the microprocessor 510.

Control Store. The control store contains the microcoded instructions which control the DMA controller 580. The control store consists of 6 1K x 8 proms configured to yield a 1K x 48 bit microword. Locations within the control store are addressed by the sequencer and data is presented at the input of the pipeline registers.

Sequencer. The sequencer controls program flow by generating control store addresses based upon pipeline data and various status bits. The control store address consists of 10 bits. Bits 8:0 of the control store address derive from a multiplexer having as its inputs either an ALU output or the output of an incrementer. The incrementer can be preloaded with pipeline register bits 8:0, or it can be incremented as a result of a test condition. The 1K address range is divided into two pages by a latched flag such that the microprogram can execute from either page. Branches, however remain within the selected page. Conditional sequencing is performed by having the test condition increment the pipeline provided address. A false condition allows execution from the pipeline address while a true condition causes execution from the address + 1. The alu output is selected as an address source in order to directly vector to a routine or in order to return to a calling routine. Note that when calling a subroutine the calling routine must reside within the same page as the subroutine or the wrong page will be selected on the return.

ALU. The alu comprises a single IDT49C402A integrated circuit. It is 16 bits in width and most closely resembles four 2901s with 64 registers. The alu is used primarily for incrementing, decrementing, addition and bit manipulation. All necessary control signals originate in the control store. The IDT HIGH PERFORMANCE CMOS 1988 DATA BOOK, incorporated by reference herein, contains additional information about the alu.

Microword. The 48 bit microword comprises several fields which control various functions of the DMA controller 580. The format of the microword is defined below along with mnemonics and a description of each function.

AI<8:0> 47:39 (Alu Instruction bits 8:0) The AI bits provide the instruction for the 49C402A alu. Refer to the IDT data book for a complete definition of the alu instructions. Note that the I9 signal input of the 49C402A is always low.

CIN 38 (Carry INput) This bit forces the carry input to the alu.

RA<5:0> 37:32 (Register A address bits 5:0) These bits select one of 64 registers as the "A" operand for the alu. These bits also provide literal bits 15:10 for the alu bus.

RB<5:0> 31:26 (Register B address bits 5:0) These bits select one of 64 registers as the "B" operand for the alu. These bits also provide literal bits 9:4 for the alu bus.

LFD 25 (Latched Flag Data) When set this bit causes the selected latched flag to be set. When reset this bit causes the selected latched flag to be cleared. This bits also functions as literal bit 3 for the alu bus.

LFS<2:0> 24:22 (Latched Flag Select bits 2:0) The meaning of these bits is dependent upon the selected source for the alu bus. In the event that the literal field is selected as the bus source then LFS<2:0> function as literal bits <2:0> otherwise the bits are used to select one of the latched flags.

LFS<2:0>	SELECTED FLAG
0	This value selects a null flag.
1	When set this bit enables the buffer clock. When reset this bit disables the buffer clock.
2	When this bit is cleared VME bus transfers, buffer operations and RAS are all disabled.
3	NOT USED
4	When set this bit enables VME bus transfers.
5	When set this bit enables buffer operations.
6	When set this bit asserts the row address strobe to the dram buffer.
7	When set this bit selects page 0 of the control store.

SRC<1,0> 20,21 (alu bus SouRCe select bits 1,0) These bits select the data source to be enabled onto the alu bus.

SRC<1,0>	Selected Source
0	alu
1	dual ported ram
2	literal
3	reserved-not defined

PF<2:0> 19:17 (Pulsed Flag select bits 2:0) These bits select a flag/signal to be pulsed.

PF<2:0>	Flag
0	null
1	SGL_CLK generates a single transition of buffer clock.
2	SET_VB forces vme and buffer enable to be set.
3	CL_PERR clears buffer parity error status.
4	SET_DN set channel done status for the currently selected channel.
5	INC_ADR increment dual ported ram address.

(continued)

PF<2:0>	Flag
6:7	RESERVED - NOT DEFINED

DEST<3:0> 16:13 (DESTination select bits 3:0) These bits select one of 10 destinations to be loaded from the alu bus.

DEST<3:0>	Destination
0	null
1	WR_RAM causes the data on the alu bus to be written to the dual ported ram. D<15:0> → ram<15:0>
2	WR_BADD loads the data from the alu bus into the dram address counters. D<14:7> → mux addr<8:0>
3	WR_VADL loads the data from the alu bus into the least significant 2 bytes of the VME address register. D<15:2> → VME addr<15:2> D1 → ENB_tional registers D<15:2> → VME addr<15:2> D1 → ENB_ENH D0 → ENB_BLK
4	WR_VADH loads the most significant 2 bytes of the VME address register. D<15:0> → VME addr<31:16>
5	WR_RADD loads the dual ported ram address counters. D<10:0> → ram addr <10:0>
6	WR_WCNT loads the word counters. D15 → count enable* D<14:8> → count <6:0>
7	WR_CO loads the co-channel select register. D<7:4> → CO<3:0>
8	WR_NXT loads the next-channel select register. D<3:0> → NEXT<3:0>
9	WR_CUR loads the current-channel select register. D<3:0> → CURR <3:0>
10:14	RESERVED - NOT DEFINED

(continued)

DEST<3:0>	Destination
15	JUMP causes the control store sequencer to select the alu data bus. D<8:0> → CS_A<8:0>

TEST<3:0> 12:9 (TEST condition select bits 3:0) Select one of 16 inputs to the test multiplexor to be used as the carry input to the incrementer.

TEST<3:0>	Condition	
0	FALSE	-always false
1	TRUE	-always true
2	ALU_COUT	-carry output of alu
3	ALU_EQ	-equals output of alu
4	ALU_OVR	-alu overflow
5	ALU_NEG	-alu negative
6	XFR_DONE	-transfer complete
7	PAR_ERR	-buffer parity error
8	TIMOUT	-bus operation timeout
9	ANY_ERR	-any error status
14:10	RESERVED	-NOT DEFINED
15	CH_RDY	-next channel ready

NEXT_A<8:0> 8:0 (NEXT Address bits 8:0) Selects an instructions from the current page of the control store for execution.

Dual Ported Ram. The dual ported ram is the medium by which command, parameters and status are communicated between the DMA controller 580 and the microprocessor 510. The ram is organized as 1K x 32 at the master port and as 2K x 16 at the DMA port. The ram may be both written and read at either port.

The ram is addressed by the DMA controller 580 by loading an 11 bit address into the address counters. Data is then read into bidirectional registers and the address counter is incremented to allow read of the next location.

Writing the ram is accomplished by loading data from the processor into the registers after loading the ram address. Successive writes may be performed on every other processor cycle.

The ram contains current block pointers, ending status, high speed bus address and parameter blocks. The following is the format of the ram:

```
OFFSET      31                                  0
          --------------------------------
  0      |CURR POINTER 0 |    STATUS 0    |
          --------------------------------
  4      |        INITIAL POINTER 0       |
          --------------------------------
                         .
                         .
          --------------------------------
 58      |CURR POINTER B |    STATUS B    |
          --------------------------------
 5C      |        INITIAL POINTER B       |
          --------------------------------
 60      |   not used    |   not used     |
          --------------------------------
 64      |   not used    |   not used     |
          --------------------------------
 68      |CURR POINTER D |    STATUS D    |
          --------------------------------
 6C      |        INITIAL POINTER D       |
          --------------------------------
 70      |   not used    |    STATUS E    |
          --------------------------------
 74      |HIGH SPEED BUS ADDRESS 31:2|0|0|
          --------------------------------
 78      |       PARAMETER BLOCK 0        |
          --------------------------------
                         .
                         .
          --------------------------------
 ??      |       PARAMETER BLOCK n        |
          --------------------------------
```

The Initial Pointer is a 32 bit value which points the first command block of a chain. The current pointer is a sixteen bit value used by the DMA controller 580 to point to the current command block. The current command block pointer should be initialized to 0x0000 by the microprocessor 510 before enabling the channel. Upon detecting a value of 0x0000 in the current block pointer the DMA controller 580 will copy the lower 16 bits from the initial pointer to the current pointer. Once the DMA controller 580 has completed the specified operations for the parameter block the current pointer will be updated to point to the next block. In the event that no further parameter blocks are available the pointer will be set to 0x0000.

The status byte indicates the ending status for the last channel operation performed. The following status bytes are defined:

STATUS	MEANING
0	NO ERRORS
1	ILLEGAL OP CODE
2	BUS OPERATION TIMEOUT
3	BUS OPERATION ERROR
4	DATA PATH PARITY ERROR

The format of the parameter block is:

```
OFFSET    31                                        0
   0     |              FORWARD LINK                 |
   4     |    NOT USED        |      WORD COUNT      |
   8     |       VME ADDRESS 31:2, ENH, BLK          |
   C     | TERM 0  |  OP 0   |      BUF ADDR 0      |
                         .
                         .
                         .
                         .
C+(4Xn)     | TERM n  |  OP n   |       BUF ADDR n|
```

FORWARD LINK - The forward link points to the first word of the next parameter block for execution. It allows several parameter blocks to be initialized and chained to create a sequence of operations for execution. The forward pointer has the following format:

A31:A2,0,0

The format dictates that the parameter block must start on a quad byte boundary. A pointer of 0x00000000 is a special case which indicates no forward link exists.

WORD COUNT - The word count specifies the number of quad byte words that are to be transferred to or from each buffer address or to/from the VME address. A word count of 64K words may be specified by initializing the word count with the value of 0. The word count has the following format:

|D15|D14|D13|D12|D11|D10|D9|D8|D7|D6|D5|D4|D3|D2|D1|D0|

The word count is updated by the DMA controller 580 at the completion of a transfer to/from the last specified buffer address. Word count is not updated after transferring to/from each buffer address and is therefore not an accurate indicator of the total data moved to/from the buffer. Word count represents the amount of data transferred to the VME bus or one of the FIFOs 544 or 554.

VME ADDRESS - The VME address specifies the starting address for data transfers. Thirty bits allows the address to start at any quad byte boundary.

ENH - This bit when set selects the enhanced block transfer protocol described in the above-cited ENHANCED VMEBUS PROTOCOL UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA TRANSFER application, to be used during the VME bus transfer. Enhanced protocol will be disabled automatically when performing any transfer to or from 24 bit or 16 bit address space, when the starting address is not 8 byte aligned or when the word count is not even.

BLK - This bit when set selects the conventional VME block mode protocol to be used during the VME bus transfer. Block mode will be disabled automatically when performing any transfer to or from 16 bit address space.

BUF ADDR - The buffer address specifies the starting buffer address for the adjacent operation. Only 16 bits are available for a 1M byte buffer and as a result the starting address always falls on a 16 byte boundary. The programmer must ensure that the starting address is on a modulo 128 byte boundary. The buffer address is updated by the DMA controller 580 after completion of each data burst.

|A19|A18|A17|A16|A15|A14|A13|A12|A11|A10|A9|A8|A7|A6|A5|A4|

TERM - The last buffer address and operation within a parameter block is identified by the terminal bit. The DMA controller 580 continues to fetch buffer address s and operations to perform until this bit is encountered. Once the last operation within the parameter block is executed the word counter is updated and if not equal to zero the series of operations is repeated. Once the word counter reaches zero the forward link pointer is used to access the next parameter block.

|0|0|0|0|0|0|0|T|

OP - Operations are specified by the op code. The op code byte has the following format:

|0|0|0|0|OP3|OP2|OP1|OP0|

The op codes are listed below ("FIFO" refers to any of the FIFOs 544 or 554):

OP CODE	OPERATION
0	NO-OP
1	ZEROES → BUFFER
2	ZEROES → FIFO
3	ZEROES → VMEbus
4	VMEbus → BUFFER
5	VMEbus → FIFO
6	VMEbus → BUFFER & FIFO
7	BUFFER → VMEbus
8	BUFFER → FIFO
9	FIFO → VMEbus
A	FIFO → BUFFER
B	FIFO → VMEbus & BUFFER
C	RESERVED
D	RESERVED
E	RESERVED
F	RESERVED

APPENDIX B

Enhanced VME Block Transfer Protocol

The enhanced VME block transfer protocol is a VMEbus compatible pseudo-synchronous fast transfer handshake protocol for use on a VME backplane bus having a master functional module and a slave functional module logically interconnected by a data transfer bus. The data transfer bus includes a data strobe signal line and a data transfer acknowledge signal line. To accomplish the handshake, the master transmits a data strobe signal of a given duration on the data strobe line. The master then awaits the reception of a data transfer acknowledge signal from the slave module on the data transfer acknowledge signal line. The slave then responds by transmitting data transfer acknowledge signal of a given duration on the data transfer acknowledge signal line.

Consistent with the pseudo-synchronous nature of the handshake protocol, the data to be transferred is referenced to only one signal depending upon whether the transfer operation is a READ or WRITE operation.

In transferring data from the master functional unit to the slave, the master broadcasts the data to be transferred. The master asserts a data strobe signal and the slave, in response to the data strobe signal, captures the data broadcast by the master. Similarly, in transferring data from the slave to the master, the slave broadcasts the data to be transferred to the master unit. The slave then asserts a data transfer acknowledge signal and the master, in response to the data transfer acknowledge signal, captures the data broadcast by the slave.

The fast transfer protocol, while not essential to the present invention, facilitates the rapid transfer of large amounts of data across a VME backplane bus by substantially increasing the data transfer rate. These data rates are achieved by using a handshake wherein the data strobe and data transfer acknowledge signals are functionally decoupled and by specifying high current drivers for all data and control lines.

The enhanced pseudo-synchronous method of data transfer (hereinafter referred to as "fast transfer mode") is implemented so as to comply and be compatible with the IEEE VME backplane bus standard. The protocol utilizes user-defined address modifiers, defined in the VMEbus standard, to indicate use of the fast transfer mode. Conventional VMEbus functional units, capable only of implementing standard VMEbus protocols, will ignore transfers made using the fast transfer mode and, as a result, are fully compatible with functional units capable of implementing the fast transfer mode.

The fast transfer mode reduces the number of bus propagations required to accomplish a handshake from four propagations, as required under conventional VMEbus protocols, to only two bus propagations. Likewise, the number of bus propagations required to effect a BLOCK READ or BLOCK WRITE data transfer is reduced. Consequently, by reducing the propagations across the VMEbus to accomplish handshaking and data transfer functions, the transfer rate is materially increased.

The enhanced protocol is described in detail in the above-cited ENHANCED VMEBUS PROTOCOL application, and will only be summarized here. Familiarity with the conventional VME bus standards is assumed.

In the fast transfer mode handshake protocol, only two bus propagations are used to accomplish a handshake, rather than four as required by the conventional protocol. At the initiation of a data transfer cycle, the master will assert and deassert DS0* in the form of a pulse of a given duration. The deassertion of DS0* is accomplished without regard as to whether a response has been received from the slave. The master then waits for an acknowledgement from the slave. Subsequent pulsing of DS0* cannot occur until a responsive DTACK* signal is received from the slave. Upon receiving the slave's assertion of DTACK*, the master can then immediately reassert data strobe, if so desired. The fast transfer mode protocol does not require the master to wait for the deassertion of DTACK* by the slave as a condition precedent to subsequent assertions of DS0*. In the fast transfer mode, only the leading edge (i.e., the assertion) of a signal is significant. Thus, the deassertion of either DS0* or DTACK* is completely irrelevant for completion of a handshake. The fast transfer protocol does not employ the DS1* line for data strobe purposes at all.

The fast transfer mode protocol may be characterized as pseudo-synchronous as it includes both synchronous and asynchronous aspects. The fast transfer mode protocol is synchronous in character due to the fact that DS0* is asserted and deasserted without regard to a response from the slave. The asynchronous aspect of the fast transfer mode protocol is attributable to the fact that the master may not subsequently assert DS0* until a response to the prior strobe is received from the slave. Consequently, because the protocol includes both synchronous and asynchronous components, it is most accurately classified as "pseudo-synchronous."

The transfer of data during a BLOCK WRITE cycle in the fast transfer protocol is referenced only to DS0*. The master first broadcasts valid data to the slave, and then asserts DS0 to the slave. The slave is given a predetermined period of time after the assertion of DS0* in which to capture the data. Hence, slave modules must be prepared to capture data at any time, as DTACK* is not referenced during the transfer cycle.

Similarly, the transfer of data during a BLOCK READ cycle in the fast transfer protocol is referenced only to DTACK*. The master first asserts DS0*. The slave then broadcasts data to the master and then asserts DTACK*. The master is given a predetermined period of time after the assertion of DTACK in which to capture the data. Hence, master modules must be prepared to capture data at any time as DS0 is not referenced during the transfer cycle.

Fig. 7, parts A through C, is a flowchart illustrating the operations involved in accomplishing the fast transfer protocol BLOCK WRITE cycle. To initiate a BLOCK WRITE cycle, the master broadcasts the memory address of the data to be transferred and the address modifier across the DTB bus. The master also drives interrupt acknowledge signal (IACK*) high and the LWORD* signal low 701. A special address modifier, for example "1F,"broadcast by the master indicates to the slave module that the fast transfer protocol will be used to accomplish the BLOCK WRITE.

The starting memory address of the data to be transferred should reside on a 64-bit boundary and the size of block of data to be transferred should be a multiple of 64 bits. In order to remain in compliance with the VMEbus standard, the block must not cross a 256 byte boundary without performing a new address cycle.

The slave modules connected to the DTB receive the address and the address modifier broadcast by the master across the bus and receive LWORD* low and IACK* high 703. Shortly after broadcasting the address and address modifier 701, the master drives the AS* signal low 705. The slave modules receive the AS* low signal 707. Each slave individually determines whether it will participate in the data transfer by determining whether the broadcasted address is valid for the slave in question 709. If the address is not valid, the data transfer does not involve that particular slave and it ignores the remainder of the data transfer cycle.

The master drives WRITE* low to indicate that the transfer cycle about to occur is a WRITE operation 711. The slave receives the WRITE* low signal 713 and, knowing that the data transfer operation is a WRITE operation, awaits receipt of a high to low transition on the DS0* signal line 715. The master will wait until both DTACK* and BERR* are high 718, which indicates that the previous slave is no longer driving the DTB.

The master proceeds to place the first segment of the data to be transferred on data lines D00 through D31, 719. After placing data on D00 through D31, the master drives DS0* low 721 and, after a predetermined interval, drives DS0* high 723.

In response to the transition of DS0* from high to low, respectively 721 and 723, the slave latches the data being transmitted by the master over data lines D00 through D31, 725. The master places the next segment of the data to be transferred on data lines D00 through D31, 727, and awaits receipt of a DTACK* signal in the form of a high to low transition signal, 729 in Fig. 7B.

Referring to Fig. 7B, the slave then drives DTACK* low, 731, and, after a predetermined period of time, drives DTACK high, 733. The data latched by the slave, 725, is written to a device, which has been selected to store the data 735. The slave also increments the device address 735. The slave then waits for another transition of DS0* from high to low 737.

To commence the transfer of the next segment of the block of data to be transferred, the master drives DS0* low 739 and, after a predetermined period of time, drives DS0* high 741. In response to the transition of DS0* from high to low, respectively 739 and 741, the slave latches the data being broadcast by the master over data lines D00 through D31, 743. The master places the next segment of the data to be transferred on data lines D00 through D31, 745, and awaits receipt of a DTACK* signal in the form of a high to low transition, 747.

The slave then drives DTACK* low, 749, and, after a predetermined period of time, drives DTACK* high, 751. The data latched by the slave, 743, is written to the device selected to store the data and the device address is incremented 753. The slave waits for another transition of DS0* from high to low 737.

The transfer of data will continue in the above-described manner until all of the data has been transferred from the master to the slave. After all of the data has been transferred, the master will release the address lines, address modifier lines, data lines, IACK* line, LWORD* line and DS0* line, 755. The master will then wait for receipt of a DTACK* high to low transition 757. The slave will drive DTACK* low, 759 and, after a predetermined period of time, drive DTACK* high 761. In response to the receipt of the DTACK* high to low transition, the master will drive AS* high 763 and then release the AS* line 765.

Fig. 8, parts A through C, is a flowchart illustrating the operations involved in accomplishing the fast transfer protocol BLOCK READ cycle. To initiate a BLOCK READ cycle, the master broadcasts the memory address of the data to be transferred and the address modifier across the DTB bus 801. The master drives the LWORD* signal low and the IACK* signal high 801. As noted previously, a special address modifier indicates to the slave module that the fast transfer protocol will be used to accomplish the BLOCK READ.

The slave modules connected to the DTB receive the address and the address modifier broadcast by the master across the bus and receive LWORD* low and IACK* high 803. Shortly after broadcasting the address and address modifier 801, the master drives the AS* signal low 805. The slave modules receive the AS* low signal 807. Each slave individually determines whether it will participate in the data transfer by determining whether the broadcasted address is valid for the slave in question 809. If the address is not valid, the data transfer does not involve that particular slave and it ignores the remainder of the data transfer cycle.

The master drives WRITE* high to indicate that the transfer cycle about to occur is a READ operation 811. The slave receives the WRITE* high signal 813 and, knowing that the data transfer operation is a READ operation, places the first segment of the data to be transferred on data lines D00 through D31 819. The master will wait until both DTACK* and BERR* are high 818, which indicates that the previous slave is no longer driving the DTB.

The master then drives DS0* low 821 and, after a predetermined interval, drives DS0* high 823. The master then awaits a high to low transition on the DTACK* signal line 824. As shown in Fig. 8B, the slave then drives the DTACK* signal low 825 and, after a predetermined period of time, drives the DTACK* signal high 827.

In response to the transition of DTACK* from high to low, respectively 825 and 827, the master latches the data being transmitted by the slave over data lines D00 through D31, 831. The data latched by the master, 831, is written to a device, which has been selected to store the data the device address is incremented 833.

The slave places the next segment of the data to be transferred on data lines D00 through D31, 829, and then waits for another transition of DS0* from high to low 837.

To commence the transfer of the next segment of the block of data to be transferred, the master drives DS0* low 839 and, after a predetermined period of time, drives DS0* high 841. The master then waits for the DTACK* line to transition from high to low, 843.

The slave drives DTACK* low, 845, and, after a predetermined period of time, drives DTACK* high, 847. In response to the transition of DTACK* from high to low, respectively 839 and 841, the master latches the data being transmitted by the slave over data lines D00 through D31, 845. The data latched by the master, 845, is written to the device selected to store the data, 851 in Fig. 8C, and the device address is incremented. The slave places the next segment of the data to be transferred on data lines D00 through D31, 849.

The transfer of data will continue in the above-described manner until all of the data to be transferred from the slave to the master has been written into the device selected to store the data. After all of the data to be transferred has been written into the storage device, the master will release the address lines, address modifier lines, data lines, the IACK* line, the LWORD line and DS0* line 852. The master will then wait for receipt of a DTACK* high to low transition 853. The slave will drive DTACK* low 855 and, after a predetermined period of time, drive DTACK* high 857. In response to the receipt of the DTACK* high to low transition, the master will drive AS* high 859 and release the AS* line 861.

To implement the fast transfer protocol, a conventional 64 mA tri-state driver is substituted for the 48 mA open collector driver conventionally used in VME slave modules to drive DTACK*. Similarly, the conventional VMEbus data drivers are replaced with 64 mA tri-state drivers in SO-type packages. The latter modification reduces the ground lead inductance of the actual driver package itself and, thus, reduces "ground bounce" effects which contribute to skew between data, DS0* and DTACK*. In addition, signal return inductance along the bus backplane is reduced by using a connector system having a greater number of ground pins so as to minimize signal return and mated-pair pin inductance. One such connector system is the "High Density Plus" connector, Model No. 420-8015-000, manufactured by Teradyne Corporation.

APPENDIX C

Parity FIFO

The parity FIFOs 240, 260 and 270 (on the network controllers 110), and 544 and 554 (on storage processors 114) are each implemented as an ASIC. All the parity FIFOs are identical, and are configured on power-up or during normal operation for the particular function desired. The parity FIFO is designed to allow speed matching between buses of different speed, and to perform the parity generation and correction for the parallel SCSI drives.

The FIFO comprises two bidirectional data ports, Port A and Port B, with 36 x 64 bits of RAM buffer between them. Port A is 8 bits wide and Port B is 32 bits wide. The RAM buffer is divided into two parts, each 36 x 32 bits, designated RAM X and RAM Y. The two ports access different halves of the buffer alternating to the other half when available. When the chip is configured as a parallel parity chip (e.g. one of the FIFOs 544 on SP 114a), all accesses on Port B are monitored and parity is accumulated in RAM X and RAM Y alternately.

The chip also has a CPU interface, which may be 8 or 16 bits wide. In 16 bit mode the Port A pins are used as the most significant data bits of the CPU interface and are only actually used when reading or writing to the Fifo Data Register inside the chip.

A REQ, ACK handshake is used for data transfer on both Ports A and B. The chip may be configured as either a master or a slave on Port A in the sense that, in master mode the Port A ACK / RDY output signifies that the chip is ready to transfer data on Port A, and the Port A REQ input specifies that the slave is responding. In slave mode, however, the Port A REQ input specifies that the master requires a data transfer, and the chip responds with Port A ACK / RDY when data is available. The chip is a master on Port B since it raises Port B REQ and waits for Port B ACK to indicate completion of the data transfer.

SIGNAL DESCRIPTIONS

Port A 0-7, P

Port A is the 8 bit data port. Port A P, if used, is the odd parity bit for this port.

A Req, A Ack/Rdy

These two signals are used in the data transfer mode to control the handshake of data on Port A.

uP Data 0-7, uP Data P, uPAdd 0-2, CS

These signals are used by a microprocessor to address the programmable registers within the chip. The odd parity signal uP Data P is only checked when data is written to the Fifo Data or Checksum Registers and microprocessor parity is enabled.

Clk

The clock input is used to generate some of the chip timing. It is expected to be in the 10-20 Mhz range.

Read En, Write En

During microprocessor accesses, while CS is true, these signals determine the direction of the microprocessor accesses. During data transfers in the WD mode these signals are data strobes used in conjunction with Port A Ack.

Port B 00-07, 10-17, 20-27, 30-37, 0P-3P

Port B is a 32 bit data port. There is one odd parity bit for each byte. Port B 0P is the parity of bits 00-07, PortB 1P is the parity of bits 10-17, Port B 2P is the parity of bits 20-27, and Port B 3P is the parity of bits 30-37.

B Select, B Req, B Ack, Parity Sync, B Output Enable

These signals are used in the data transfer mode to control the handshake of data on Port B. Port B Req and Port B Ack are both gated with Port B Select. The Port B Ack signal is used to strobe the data on the Port B data lines. The parity sync signal is used to indicate to a chip configured as the parity chip to indicate that the last words of data involved in the parity accumulation are on Port B. The Port B data lines will only be driven by the Fifo chip if all of the following conditions are met:

a. the data transfer is from Port A to Port B;
b. the Port B select signal is true;
c. the Port B output enable signal is true; and
d. the chip is not configured as the parity chip or it is in parity correct mode and the Parity Sync signal is true.

Reset

This signal resets all the registers within the chip and causes all bidirectional pins to be in a high impedance state.

DESCRIPTION OF OPERATION

Normal Operation. Normally the chip acts as a simple FIFO chip. A FIFO is simulated by using two RAM buffers in a simple ping-pong mode. It is intended, but not mandatory, that data is burst into or out of the FIFO on Port B. This is done by holding Port B Sel signal low and pulsing the Port B Ack signal. When transferring data from Port B to Port A, data is first written into RAM X and when this is full, the data paths will be switched such that Port B may start writing to RAM Y. Meanwhile the chip will begin emptying RAM X to Port A. When RAM Y is full and RAM X empty the data paths will be switched again such that Port B may reload RAM X and Port A may empty RAM Y.

Port A Slave Mode. This is the default mode and the chip is reset to this condition. In this mode the chip waits for a master such as one of the SCSI adapter chips 542 to raise Port A Request for data transfer. If data is available the Fifo chip will respond with Port A Ack/Rdy.

Port A WD Mode. The chip may be configured to run in the WD or Western Digital mode. In this mode the chip must be configured as a slave on Port A. It differs from the default slave mode in that the chip responds with Read Enable or Write Enable as appropriate together with Port A Ack/Rdy. This mode is intended to allow the chip to be interfaced to the Western Digital 33C93A SCSI chip or the NCR 53C90 SCSI chip.

Port A Master Mode. When the chip is configured as a master, it will raise Port A Ack/Rdy when it is ready for data transfer. This signal is expected to be tied to the Request input of a DMA controller which will respond with Port A Req when data is available. In order to allow the DMA controller to burst, the Port A Ack/Rdy signal will only be negated after every 8 or 16 bytes transferred.

Port B Parallel Write Mode. In parallel write mode, the chip is configured to be the parity chip for a parallel transfer from Port B to Port A. In this mode, when Port B Select and Port B Request are asserted, data is written into RAM X or RAM Y each time the Port B Ack signal is received. For the first block of 128 bytes data is simply copied into the selected RAM. The next 128 bytes driven on Port B will be exclusive-ORed with the first 128 bytes. This procedure will be repeated for all drives such that the parity is accumulated in this chip. The Parity Sync signal should be asserted to the parallel chip together with the last block of 128 bytes. This enables the chip to switch access to the other RAM and start accumulating a new 128 bytes of parity.

Port B Parallel Read Mode - Check Data. This mode is set if all drives are being read and parity is to be checked. In this case the Parity Correct bit in the Data Transfer Configuration Register is not set. The parity chip will first read 128 bytes on Port A as in a normal read mode and then raise Port B Request. While it has this signal asserted the chip will monitor the Port B Ack signals and exclusive-or the data on Port B with the data in its selected RAM. The Parity Sync should again be asserted with the last block of 128 bytes. In this mode the chip will not drive the Port B data lines but will check the output of its exclusive-or logic for zero. If any bits are set at this time a parallel parity error will be flagged.

Port B Parallel Read Mode - Correct Data. This mode is set by setting the Parity Correct bit in the Data Transfer Configuration Register. In this case the chip will work exactly as in the check mode except that when Port B Output Enable, Port B Select and Parity Sync are true the data is driven onto the Port B data lines and a parallel parity check for zero is not performed.

Byte Swap. In the normal mode it is expected that Port B bits 00-07 are the first byte, bits 10-17 the second byte, bits 20-27 the third byte, and bits 30-37 the last byte of each word. The order of these bytes may be changed by writing to the byte swap bits in the configuration register such that the byte address bits are inverted. The way the bytes are written and read also depend on whether the CPU interface is configured as 16 or 8 bits. The following table shows the byte alignments for the different possibilities for data transfer using the Port A Request / Acknowledge handshake:

CPU I/F	Invert Addr 1	Invert Addr 0	Port B 00-07	Port B 10-17	Port B 20-27	Port B 30-37
8	False	False	Port A	Port A	Port A	Port A

(continued)

CPU I/F	Invert Addr 1	Invert Addr 0	Port B 00-07	Port B 10-17	Port B 20-27	Port B 30-37
			byte 0	byte 1	byte 2	byte 1
8	False	True	Port A byte 1	Port A byte 0	Port A byte 3	Port A byte 2
8	True	False	Port A byte 2	Port A byte 3	Port A byte 0	Port A byte 1
8	True	True	Port A byte 3	Port A byte 2	Port A byte 1	Port A byte 0
16	False	False	Port A byte 0	uProc byte 0	Port A byte 1	uProc byte 1
16	False	True	uProc byte 0	Port A byte 0	uProc byte 1	Port A byte 1
16	True	False	Port A byte 1	uProc byte 1	Port A byte 0	uProc byte 0
16	True	True	uProc byte 1	Port A byte 1	uProc byte 0	Port A byte 0

When the Fifo is accessed by reading or writing the Fifo Data Register through the microprocessor port in 8 bit mode, the bytes are in the same order as the table above but the uProc data port is used instead of Port A. In 16 bit mode the table above applies.

Odd Length Transfers. If the data transfer is not a multiple of 32 words, or 128 bytes, the microprocessor must manipulate the internal registers of the chip to ensure all data is transferred. Port A Ack and Port B Req are normally not asserted until all 32 words of the selected RAM are available. These signals may be forced by writing to the appropriate RAM status bits of the Data Transfer Status Register.

When an odd length transfer has taken place the microprocessor must wait until both ports are quiescent before manipulating any registers. It should then reset both of the Enable Data Transfer bits for Port A and Port B in the Data Transfer Control Register. It must then determine by reading their Address Registers and the RAM Access Control Register whether RAM X or RAM Y holds the odd length data. It should then set the corresponding Address Register to a value of 20 hexadecimal, forcing the RAM full bit and setting the address to the first word. Finally the microprocessor should set the Enable Data Transfer bits to allow the chip to complete the transfer.

At this point the Fifo chip will think that there are now a full 128 bytes of data in the RAM and will transfer 128 bytes if allowed to do so. The fact that some of these 128 bytes are not valid must be recognized externally to the FIFO chip.

PROGRAMMABLE REGISTERS

Data Transfer Configuration Register (Read/Write)

Register Address 0. This register is cleared by the reset signal.

Bit 0	WD Mode. Set if data transfers are to use the Western Digital WD33C93A protocol, otherwise the Adaptec 6250 protocol will be used.
Bit 1	Parity Chip. Set if this chip is to accumulate Port B parities.
Bit 2	Parity Correct Mode. Set if the parity chip is to correct parallel parity on Port B.

(continued)

Bit 3	CPU Interface 16 bits wide. If set, the microprocessor data bits are combined with the Port A data bits to effectively produce a 16 bit Port. All accesses by the microprocessor as well as all data transferred using the Port A Request and Acknowledge handshake will transfer 16 bits.
Bit 4	Invert Port A byte address 0. Set to invert the least significant bit of Port A byte address.
Bit 5	Invert Port A byte address 1. Set to invert the most significant bit of Port A byte address.
Bit 6	Checksum Carry Wrap. Set to enable the carry out of the 16 bit checksum adder to carry back into the least significant bit of the adder.
Bit 7	Reset. Writing a 1 to this bit will reset the other registers. This bit resets itself after a maximum of 2 clock cycles and will therefore normally be read as a 0. No other register should be written for a minimum of 4 clock cycles after writing to this bit.

Data Transfer Control Register (Read/Write)

Register Address 1. This register is cleared by the reset signal or by writing to the reset bit.

Bit 0	Enable Data Transfer on Port A. Set to enable the Port A Req/Ack handshake.
Bit 1	Enable Data Transfer on Port B. Set to enable the Port B Req/Ack handshake.
Bit 2	Port A to Port B. If set, data transfer is from Port A to Port B. If reset, data transfer is from Port B to Port A. In order to avoid any glitches on the request lines, the state of this bit should not be altered at the same time as the enable data transfer bits 0 or 1 above.
Bit 3	uProcessor Parity Enable. Set if parity is to be checked on the microprocessor interface. It will only be checked when writing to the Fifo Data Register or reading from the Fifo Data or Checksum Registers, or during a Port A Request/Acknowledge transfer in 16 bit mode. The chip will, however, always re-generate parity ensuring that correct parity is written to the RAM or read on the microprocessor interface.
Bit 4	Port A Parity Enable. Set if parity is to be checked on Port A. It is checked when accessing the Fifo Data Register in 16 bit mode, or during a Port A Request/Acknowledge transfer. The chip will, however, always re-generate parity ensuring that correct parity is written to the RAM or read on the Port A interface.
Bit 5	Port B Parity Enable. Set if Port B data has valid byte parities. If it is not set, byte parity is generated internally to the chip when writing to the RAMs. Byte parity is not checked when writing from Port B, but always checked when reading to Port B.
Bit 6	Checksum Enable. Set to enable writing to the 16 bit checksum register. This register accumulates a 16 bit checksum for all RAM accesses, including accesses to the Fifo Data Register, as well as all writes to the checksum register. This bit must be reset before reading from the Checksum Register.
Bit 7	Port A Master. Set if Port A is to operate in the master mode on Port A during the data transfer.

Data Transfer Status Register (Read Only)

Register Address 2. This register is cleared by the reset signal or by writing to the reset bit.

Bit 0	Data in RAM X or RAM Y. Set if any bits are true in the RAM X, RAM Y, or Port A byte address registers.

(continued)

Bit 1	uProc Port Parity Error. Set if the uProc Parity Enable bit is set and a parity error is detected on the microprocessor interface during any RAM access or write to the Checksum Register in 16 bit mode.
Bit 2	Port A Parity Error. Set if the Port A Parity Enable bit is set and a parity error is detected on the Port A interface during any RAM access or write to the Checksum Register.
Bit 3	Port B Parallel Parity Error . Set if the chip is configured as the parity chip, is not in parity correct mode, and a non zero result is detected when the Parity Sync signal is true. It is also set whenever data is read out onto Port B and the data being read back through the bidirectional buffer does not compare.
Bits 4-7	Port B Bytes 0-3 Parity Error. Set whenever the data being read out of the RAMs on the Port B side has bad parity.

Ram Access Control Register (Read/Write)

Register Address 3. This register is cleared by the reset signal or by writing to the reset bit. The Enable Data Transfer bits in the Data Transfer Control Register must be reset before attempting to write to this register, else the write will be ignored.

Bit 0	Port A byte address 0. This bit is the least significant byte address bit. It is read directly bypassing any inversion done by the invert bit in the Data Transfer Configuration Register.
Bit 1	Port A byte address 1. This bit is the most significant byte address bit. It is read directly bypassing any inversion done by the invert bit in the Data Transfer Configuration Register.
Bit 2	Port A to RAM Y. Set if Port A is accessing RAM Y, and reset if it is accessing RAM X .
Bit 3	Port B to RAM Y. Set if Port B is accessing RAM Y, and reset if it is accessing RAM X .
Bit 4	Long Burst. If the chip is configured to transfer data on Port A as a master, and this bit is reset, the chip will only negate Port A Ack/Rdy after every 8 bytes, or 4 words in 16 bit mode, have been transferred. If this bit is set, Port A Ack/Rdy will be negated every 16 bytes, or 8 words in 16 bit mode.
Bits 5-7	Not Used.

RAM X Address Register (Read/Write)

Register Address 4. This register is cleared by the reset signal or by writing to the reset bit. The Enable Data Transfer bits in the Data Transfer Control Register must be reset before attempting to write to this register, else the write will be ignored.

Bits 0-4	RAM X word address
Bit 5	RAM X full
Bits 6-7	Not Used

RAM Y Address Register (Read/Write)

Register Address 5. This register is cleared by the reset signal or by writing to the reset bit. The Enable Data Transfer bits in the Data Transfer Control Register must be reset before attempting to write to this register, else the write will be ignored.

Bits 0-4	RAM Y word address
Bit 5	RAM Y full

(continued)

Bits 6-7	Not Used

Fifo Data Register (Read/Write)

Register Address 6. The Enable Data Transfer bits in the Data Transfer Control Register must be reset before attempting to write to this register, else the write will be ignored. The Port A to Port B bit in the Data Transfer Control register must also be set before writing this register. If it is not, the RAM controls will be incremented but no data will be written to the RAM. For consistency, the Port A to PortB should be reset prior to reading this register.

Bits 0-7 are Fifo Data. The microprocessor may access the FIFO by reading or writing this register. The RAM control registers are updated as if the access was using Port A. If the chip is configured with a 16 bit CPU Interface the most significant byte will use the Port A 0-7 data lines, and each Port A access will increment the Port A byte address by 2.

Port A Checksum Register (Read/Write)

Register Address 7. This register is cleared by the reset signal or by writing to the reset bit.

Bits 0-7 are Checksum Data. The chip will accumulate a 16 bit checksum for all Port A accesses. If the chip is configured with a 16 bit CPU interface, the most significant byte is read on the Port A 0-7 data lines. If data is written directly to this register it is added to the current contents rather than overwriting them. It is important to note that the Checksum Enable bit in the Data Transfer Control Register must be set to write this register and reset to read it.

PROGRAMMING THE FIFO CHIP

In general the fifo chip is programmed by writing to the data transfer configuration and control registers to enable a data transfer, and by reading the data transfer status register at the end of the transfer to check the completion status. Usually the data transfer itself will take place with both the Port A and the Port B handshakes enabled, and in this case the data transfer itself should be done without any other microprocessor interaction. In some applications, however, the Port A handshake may not be enabled, and it will be necessary for the microprocessor to fill or empty the fifo by repeatedly writing or reading the Fifo Data Register.

Since the fifo chip has no knowledge of any byte counts, there is no way of telling when any data transfer is complete by reading any register within this chip itself. Determination of whether the data transfer has been completed must therefore be done by some other circuitry outside this chip.

The following C language routines illustrate how the parity FIFO chip may be programmed. The routines assume that both Port A and the microprocessor port are connected to the system microprocessor, and return a size code of 16 bits, but that the hardware addresses the Fifo chip as long 32 bit registers.

```
struct FIFO_regs {
     unsigned char config,a1,a2,a3 ;
     unsigned char control,b1,b2,b3;
     unsigned char status,c1,c2,c3;
     unsigned char ram_access_control,d1,d2,d3;
     unsigned char ram_X_addr,e1,e2,e3;
     unsigned char ram_Y_addr,f1,f2,f3;
     unsigned long data;
     unsigned int checksum,h1;
     };

#define FIFO1 ((struct  FIFO_regs*) FIFO_BASE_ADDRESS)

#define FIFO_RESET 0x80
#define FIFO_16_BITS 0x08
#define FIFO_CARRY_WRAP 0x40
#define FIFO_PORT_A_ENABLE 0x01
#define FIFO_PORT_B_ENABLE 0x02
#define FIFO_PORT_ENABLES 0x03
#define FIFO_PORT_A_TO_B 0x04
#define FIFO_CHECKSUM_ENABLE 0x40
#define FIFO_DATA_IN_RAM 0x01
#define FIFO_FORCE_RAM_FULL 0x20

#define PORT_A_TO_PORT_B(fifo) ((fifo-> control ) & 0x04)
#define PORT_A_BYTE_ADDRESS(fifo) ((fifo->ram_access_control) &
0x03)
#define PORT_A_TO_RAM_Y(fifo)  ((fifo->ram_access_control ) &    0x04)
#define PORT_B_TO_RAM_Y(fifo)  ((fifo-> ram_access_control ) &    0x08)
```

```
/*************************************************************
          The following routine initiates a Fifo data transfer using two
values passed to it.

    config_data   This is the data to be written to the configuration register.

    control_data  This is the data to be written to the Data Transfer Control
                  Register. If the data transfer is to take place
                  automatically using both the Port Aand Port B
                  handshakes, both data transfer enables bits should be
                  set in this parameter.
*************************************************************/

FIFO_initiate_data_transfer(config_data, control_data)
unsigned char config_data, control_data;
{
    FIFO1->config = config_data | FIFO_RESET;       /* Set
Configuration value & Reset */
    FIFO1->control = control_data & (~FIFO_PORT_ENABLES);  /* Set
          everything but enables */
    FIFO1->control = control_data ;               /* Set data transfer
enables */
}

/*************************************************************
          The following routine forces the transfer of any odd bytes that
have been left in the Fifo at the end of a data transfer.
It first disables both ports, then forces the Ram Full bits, and then
re-enables the appropriate Port.
*************************************************************/

FIFO_force_odd_length_transfer()
{
    FIFO1->control &= ~FIFO_PORT_ENABLES; /* Disable Ports A & B
*/
    if (PORT_A_TO_PORT_B(FIFO1)) {
          if (PORT_A_TO_RAM_Y(FIFO1)) {
                FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL; /*
Set RAM Y full */
          }
          else FIFO1->ram_X_addr = FIFO_FORCE_RAM_FULL ;  /* Set
RAM X full */
          FIFO1->control |= FIFO_PORT_B_ENABLE ;          /*
Re-Enable Port B */
    }
    else {
          if (PORT_B_TO_RAM_Y(FIFO1)) {
                FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL ;  /*
Set RAM Y full */
          }
          else FIFO1->ram_X_addr = FIFO_FORCE_RAM_FULL ;  /* Set
RAM X full */
```

```
            FIFO1->control |= FIFO_PORT_A_ENABLE ;          /*
Re-Enable Port A */
    }
}

/**************************************************************
            The following routine returns how many odd bytes have been
left in the Fifo at the end of a data transfer.
**************************************************************/

int FIFO_count_odd_bytes()
{
    int number_odd_bytes;
    number_odd_bytes=0;
    if (FIFO1->status & FIFO_DATA_IN_RAM)  {
            if (PORT_A_TO_PORT_B(FIFO1))  {
                    number_odd_bytes =
(PORT_A_BYTE_ADDRESS(FIFO1)) ;
                    if (PORT_A_TO_RAM_Y(FIFO1))
                            number_odd_bytes += (FIFO1->ram_Y_addr) *
4 ;
                    else number_odd_bytes += (FIFO1->ram_X_addr) * 4 ;
            }
            else {
                    if (PORT_B_TO_RAM_Y(FIFO1))
                            number_odd_bytes = (FIFO1->ram_Y_addr) * 4 ;
                    else number_odd_bytes = (FIFO1->ram_X_addr) * 4 ;
            }
    }
    return (number_odd_bytes);
}

/**************************************************************
            The following routine tests the microprocessor interface of the
chip.  It first writes and reads the first 6 registers.  It then writes 1s, 0s, and
an address pattern to the RAM, reading the data back and checking it.

            The test returns a bit significant error code where each bit
represents the address of the registers that failed.

    Bit 0 = config register failed
    Bit 1 = control register failed
    Bit 2 = status register failed
    Bit 3 = ram access control register failed
    Bit 4 = ram X address register failed
    Bit 5 = ram Y address register failed
    Bit 6 = data register failed
    Bit 7 = checksum register failed
**************************************************************/

#define RAM_DEPTH 64            /* number of long words in Fifo Ram */

reg_expected_data[6] = { 0x7F, 0xFF, 0x00, 0x1F, 0x3F, 0x3F };
```

```
char FIFO_uprocessor_interface_test()
{
    unsigned long test_data;
    char *register_addr;
    int i;
    char j,error;
    FIFO1->config = FIFO_RESET;              /* reset the chip */
    error=0;
    register_addr =(char *) FIFO1;
    j=1;

    /* first test registers 0 thru 5 */

    for  (i=0; i<6; i++)  {
           *register_addr = 0xFF;                /* write test data */
           if (*register_addr != reg_expected_data[i]) error |= j;
           *register_addr = 0;             /* write 0s to register */
           if (*register_addr) error |= j;
           *register_addr = 0xFF;                /* write test data again */
           if (*register_addr != reg_expected_data[i]) error |= j;
           FIFO1->config = FIFO_RESET;              /* reset the chip */
           if (*register_addr) error |= j; /* register should be 0 */
           register_addr++;                /* go to next register */
           j <<=1;
    }

           /* now test Ram data & checksum registers
           test 1s throughout Ram & then test 0s */

     for (test_data = -1; test_data != 1; test_data++)   {       /* test for 1s
& 0s */
           FIFO1->config = FIFO_RESET | FIFO_16_BITS ;
           FIFO1->control = FIFO_PORT_A_TO_B;
           for (i=0;i<RAM_DEPTH;i++)                /* write data to RAM
*/
                  FIFO1->data = test_data;
           FIFO1->control = 0;
           for (i=0;i<RAM_DEPTH;i++)
                  if (FIFO1->data != test_data) error |= j;     /* read &
check data */
           if (FIFO1->checksum) error |= 0x80;              /* checksum
should = 0 */
     }

           /* now test Ram data with address pattern
            uses a different pattern for every byte */

     test_data=0x00010203;            /* address pattern start */
     FIFO1->config = FIFO_RESET | FIFO_16_BITS |
FIFO_CARRY_WRAP;
     FIFO1->control = FIFO_PORT_A_TO_B |
FIFO_CHECKSUM_ENABLE;
     for (i=0;i<RAM_DEPTH;i++) {
           FIFO1->data = test_data;            /* write address pattern */
```

```
          test_data += 0x04040404;
      }
      test_data=0x00010203;              /* address pattern start */
      FIFO1->control = FIFO_CHECKSUM_ENABLE;
      for (i=0;i<RAM_DEPTH;i++) {
          if (FIFO1->status != FIFO_DATA_IN_RAM)
                error | = 0x04;              /* should be data in ram */
          if (FIFO1->data != test_data) error | = j;    /* read & check
address pattern */
          test_data += 0x04040404;
      }
      if (FIFO1->checksum != 0x0102) error | = 0x80;   /* test checksum of
address pattern */
      FIFO1->config = FIFO_RESET | FIFO_16_BITS ; /* inhibit carry wrap
*/
      FIFO1->checksum = 0xFEFE;        /* writing adds to checksum */
      if (FIFO1->checksum) error |=0x80;          /* checksum should be 0
*/
      if (FIFO1->status) error | = 0x04;     /* status should be 0 */
      return (error);
}
```

Claims

1. Network server apparatus (100) for use with a first data network (122a) and a mass storage device, including a host processor unit (118) capable of running remote procedures defined by a client node on said network, characterised in that said network server apparatus further comprises:

   an interface processor unit coupleable to said network and to said mass storage device; and
   means (110a, 112a, 114a, 116a, 120) in said interface processor unit for satisfying network storage requests from said network to store data from said network in said mass storage device, for satisfying network retrieval requests from said network to retrieve data from said mass storage device to said network, and for transmitting predefined categories of messages from said network to said host processor unit for processing in said host processor unit, said transmitted messages including all requests by a network client to run client-defined procedures on said network server apparatus.

2. Apparatus according to claim 1, wherein said interface processor unit comprises:

   a network control unit (110a) coupleable to said network (122a);
   a data control unit (112a, 114a) coupleable to said mass storage device;
   a buffer memory (116a); and
   means (210,212,214,220,222,224,232,234,236,252,254,256) in said network control unit:
   for transmitting to said data control unit said network storage requests from said network to store specified storage data from said network in said mass storage device,
   for transmitting said specified storage data from said network to said buffer memory and from said buffer memory to said data control unit,
   for transmitting to said data control unit said network retrieval requests from said network to retrieve specified retrieval data from said mass storage device to said network,
   for transmitting said specified retrieval data from said data control unit to said buffer memory and from said buffer memory to said network, and
   for transmitting said predefined categories of messages from said network to said host processor unit for processing by said host processor unit.

3. Apparatus according to claim 2, wherein said data control unit comprises:

a storage processor unit (114a) coupleable to said mass storage device;
a file processor unit (112a) and
means (310,312,314,320,324,390,396) in said file processor unit;
for translating said network storage requests from said network into requests to store data at specified physical storage locations in said mass storage device,
for instructing said storage processor unit to write data from said buffer memory into said specified physical storage locations in said mass storage device,
for translating said network retrieval requests from said network into requests to retrieve data from specified physical retrieval locations in said mass storage device, and
for instructing said storage processor unit to retrieve data from said specified physical retrieval locations in said mass storage device to said buffer memory if said data from said specified physical locations is not already in said buffer memory; and
means (510,512,514,524,532,534) in said storage processor unit for transmitting data between said buffer memory and said mass storage device.

4. Apparatus according to claim 1, wherein said interface processor unit comprises:

a network control module (110a), including a network interface (234; 214) coupled to receive said network retrieval requests from said network;
a file system control module (112a, 114a), including a mass storage device interface (540, 514) coupled to said mass storage device; and
a communication path (120) coupled directly between said network control module and said file system control module, said communication path carrying local retrieval requests prepared by said network control module in response to said network retrieval requests, to retrieve specified retrieval data from said mass storage device,
said file system control module retrieving said specified retrieval data from said mass storage device in response to said local retrieval requests and returning said specified retrieval data to said network control module, and said network control module preparing reply messages containing said specified retrieval data from said file system control module for return transmission on said network.

5. Apparatus according to claim 4, wherein said file system control module returns said specified retrieval data directly to said network control module.

6. Apparatus according to claim 4 or 5, wherein said network interface is coupled further to receive said network storage requests from said network, wherein said network control module further prepares local storage requests in response to said network storage requests, to store specified storage data in said mass storage device, said network control module communicating said local storage requests to said file system control module,
and wherein said file system control module further stores said specified storage data in said mass storage device in response to said local storage requests.

7. Apparatus according to claim 6, wherein said local storage requests are communicated to said file system control module via said communication path (120).

8. Apparatus according to any preceding claim, wherein said host processor unit runs a general purpose operating system, and wherein said interface processor unit runs no general purpose operating system.

9. Apparatus according to claim 8, wherein said general purpose operating system run on said host processor unit is a UNIX operating system.

10. Apparatus according to any preceding claim, wherein said interface processor unit includes means (214, 314, 514) for decoding all network file system NFS requests addressed to said interface processing unit from said network, for performing all procedures for satisfying said NFS requests, and for encoding any NFS reply messages for return transmission on said network.

11. Apparatus according to any preceding claim, wherein said transmitted messages include all calls issued by a network client to said network server apparatus which are within a predefined set of remote procedure calls.

12. Apparatus according to any preceding claim, further comprising means in said interface processor unit for detecting

requests from said network for an address of said network server apparatus, for preparing a response packet to such an address request, and for transmitting said response packet over said network.

13. Apparatus according to any preceding claim, for use further with a second data network (122b), said interface processor unit being coupleable further to said second network, further comprising means (210, 214, 230, 250, 234, 254) in said interface processor unit for detecting a request from said first network to route a message contained in certain packets to a destination reachable over said second network, and means for transmitting said message over said second network.

Patentansprüche

1. Netzwerkservervorrichtung (100) zur Verwendung mit einem ersten Datennetz (122a) und einer Massenspeichereinrichtung, umfassend eine Zentralrechnereinheit (118), die zum Ausführen von durch einen Clientknoten im Netzwerk definierten Fernprozeduren geeignet ist, dadurch gekennzeichnet, daß die Netzwerkservervorrichtung ferner umfaßt:

    eine Schnittstellenprozessoreinheit, die an das Netzwerk und an die Massenspeichereinrichtung koppelbar ist; und
    Mittel (110a, 112a, 114a, 116a, 120) in der Schnittstellenprozessoreinheit zum Befriedigen von Netzwerkspeicheranforderungen aus dem Netzwerk, Daten aus dem Netzwerk in der Massenspeichereinrichtung zu speichern, zum Befriedigen von Netzwerkausleseanforderungen vom Netzwerk, Daten aus der Massenspeichereinrichtung zum Netzwerk auszulesen und zum Übertragen vorbestimmter Kategorien von Mitteilungen aus dem Netzwerk zur Zentralrechnereinheit zum Verarbeiten in der Zentralrechnereinheit, wobei die übertragenen Mitteilungen alle Anforderungen durch einen Netzwerkclient enthalten, clientdefinierte Prozeduren auf der Netzwerkservervorrichtung auszuführen.

2. Vorrichtung nach Anspruch 1, wobei die Schnittstellenprozessoreinheit umfaßt:

    eine an das Netzwerk (122a) koppelbare Netzwerksteuereinheit (110a);
    eine an die Massenspeichereinrichtung koppelbare Datensteuereinheit (112a, 114a);
    einen Pufferspeicher (116a); und
    Mittel (210, 212, 214, 220, 222, 224, 232, 234, 236, 252, 254, 256) in der Netzwerksteuereinheit:
    zum Übertragen der Netzwerkspeicheranforderungen aus dem Netzwerk zu der Datensteuereinheit, um bestimmte Speicherdaten aus dem Netzwerk in der Massenspeichereinrichtung zu speichern,
    zum Übertragen der bestimmten Speicherdaten aus dem Netzwerk zum Pufferspeicher und aus dem Pufferspeicher zur Datensteuereinheit,
    zum Übertragen der Netzwerkausleseanforderungen aus dem Netzwerk zur Datensteuereinheit, um bestimmte Auslesedaten aus der Massenspeichereinrichtung zum Netzwerk auszulesen,
    zum Übertragen der bestimmten Auslesedaten aus der Datensteuereinheit zum Pufferspeicher und aus dem Pufferspeicher zum Netzwerk, und
    zum Übertragen der vorbestimmten Kategorien von Mitteilungen aus dem Netzwerk zur Zentralrechnereinheit zum Verarbeiten durch die Zentralrechnereinheit.

3. Vorrichtung nach Anspruch 2, wobei die Datensteuereinheit umfaßt:

    eine an die Massenspeichereinrichtung koppelbare Speicherprozessoreinheit (114a);
    eine Dateiprozessoreinheit (112a) und
    Mittel (310, 312, 314, 320, 324, 390, 396) in der Dateiprozessoreinheit: zum Übersetzen der Netzwerkspeicheranforderungen aus dem Netzwerk in Anforderungen, Daten an bestimmten physikalischen Speicherstellen in der Massenspeichereinrichtung zu speichern,
    zum Anweisen der Speicherprozessoreinheit, Daten aus dem Pufferspeicher in die bestimmten physikalischen Speicherstellen in der Massenspeichereinrichtung zu schreiben,
    zum Übersetzen der Netzwerkausleseanforderungen aus dem Netzwerk in Anforderungen, Daten von bestimmten physikalischen Auslesestellen in der Massenspeichereinrichtung auszulesen, und
    zum Anweisen der Speicherprozessoreinheit, Daten von den bestimmten physikalischen Auslesestellen in der Massenspeichereinrichtung zum Pufferspeicher auszulesen, wenn die Daten von den bestimmten physikalischen Stellen nicht bereits im Pufferspeicher sind; und

Mittel (510, 512, 514, 524, 532, 534) in der Speicherprozessoreinheit zum Übertragen von Daten zwischen dem Pufferspeicher und der Massenspeichereinrichtung.

4. Vorrichtung nach Anspruch 1, wobei die Schnittstellenprozessoreinheit umfaßt:

   ein Netzwerksteuermodul (110a), umfassend eine Netzwerkschnittstelle (234; 214), die angekoppelt ist, um die Netzwerkausleseanforderungen aus dem Netzwerk zu empfangen;
   ein Dateisystemsteuermodul (112a, 114a), umfassend eine Massenspeichereinrichtungsschnittstelle (540; 514), die an die Massenspeichereinrichtung gekoppelt ist; und
   einen Kommunikationsweg (120), der direkt zwischen dem Netzwerksteuermodul und dem Dateisystemsteuermodul angekoppelt ist, wobei der Kommunikationsweg lokale Ausleseanforderungen führt, die durch das Netzwerksteuermodul in Antwort auf die Netzwerkausleseanforderungen vorbereitet sind, um bestimmte Auslesedaten aus der Massenspeichereinrichtung auszulesen,
   wobei das Dateisystemsteuermodul die bestimmten Auslesedaten aus der Massenspeichereinrichtung in Antwort auf die lokalen Ausleseanforderungen ausliest und die bestimmten Auslesedaten zu dem Netzwerksteuermodul zurückgibt,
   und wobei das Netzwerksteuermodul Antwortmitteilungen vorbereitet, die die bestimmten Auslesedaten vom Dateisystemsteuermodul enthalten, zur Rückübertragung auf das Netzwerk.

5. Vorrichtung nach Anspruch 4, wobei das Dateisystemsteuermodul die bestimmten Auslesedaten direkt zum Netzwerksteuermodul zurückgibt.

6. Vorrichtung nach Anspruch 4 oder 5, wobei die Netzwerkschnittstelle ferner angekoppelt ist, um die Netzwerkspeicheranforderungen aus dem Netzwerk zu empfangen, wobei das Netzwerksteuermodul ferner lokale Speicheranforderungen in Antwort auf die Netzwerkspeicheranforderungen vorbereitet, bestimmte Speicherdaten in der Massenspeichereinrichtung zu speichern, wobei das Netzwerksteuermodul die lokalen Speicheranforderungen dem Dateisystemsteuermodul mitteilt,
   und wobei das Dateisystemsteuermodul femer die bestimmten Speicherdaten in der Massenspeichereinrichtung in Antwort auf lokale Speicheranforderungen speichert.

7. Vorrichtung nach Anspruch 6, wobei die lokalen Speicheranforderungen dem Dateisystemsteuermodul über den Kommunikationsweg (120) mitgeteilt werden.

8. Vorrichtung nach einem der vorangegangenen Ansprüche, wobei die Zentralrechnereinheit ein universelles Betriebssystem ausführt, und wobei die Schnittstellenprozessoreinheit kein universelles Betriebssystem ausführt.

9. Vorrichtung nach Anspruch 8, wobei das auf der Zentralprozessoreinheit ausgeführte universelle Betriebssystem ein UNIX-Betriebssystem ist.

10. Vorrichtung nach einem der vorangegangenen Ansprüche, wobei die Schnittstellenprozessoreinheit Mittel (214, 314, 514) zum Dekodieren aller Netzwerkdateisystem-NFS-Anforderungen, die vom Netzwerk an die Schnittstellenverarbeitungseinheit gerichtet sind, um alle Prozeduren zum Befriedigen der NFS-Anforderungen durchzuführen, und um NFS-Antwortmitteilungen zur Rückübertragung auf das Netzwerk zu kodieren.

11. Vorrichtung nach einem der vorangegangenen Ansprüche, wobei die übertragenen Mitteilungen alle durch einen Netzwerkclient an die Netzwerkservervorrichtung ausgegebenen Aufrufe umfassen, die in einem vorbestimmten Satz von Femprozeduraufrufen sind.

12. Vorrichtung nach einem der vorangegangenen Ansprüche, ferner umfassend Mittel in der Schnittstellenprozessoreinheit zum Erfassen von Anforderungen aus dem Netzwerk für eine Adresse der Netzwerkservervorrichtung, zum Vorbereiten eines Antwortpakets für eine derartige Adressenanforderung, und zum Übertragen des Antwortpakets über das Netzwerk.

13. Vorrichtung nach einem der vorangegangenen Ansprüche, zur Verwendung mit einem zweiten Datennetzwerk (122b), wobei die Schnittstellenprozessoreinheit ferner an das zweite Netzwerk koppelbar ist, ferner umfassend Mittel (210, 214, 230, 250, 234, 254) in der Schnittstellenprozessoreinheit zum Erfassen einer Anforderung vom ersten Netzwerk, um eine in gewissen Paketen enthaltene Mitteilung an ein über das zweite Netzwerk erreichbares Ziel zu leiten, und Mittel zum Übertragen der Mitteilung über das zweite Netzwerk.

Revendications

1. Dispositif serveur de réseau (100) destiné à être utilisé avec un premier réseau de données (122a) et un dispositif de stockage de masse, comprenant une unité à processeur hôte (118) capable d'exécuter des procédures à distance définies par un noeud client sur ledit réseau, caractérisé en ce que ledit dispositif serveur de réseau comprend en outre :

   une unité d'interface à processeur pouvant être couplée audit réseau et audit dispositif de stockage de masse ; et
   des moyens (110a, 112a, 114a, 116a, 120) dans ladite unité d'interface à processeur destinés à satisfaire des demandes de stockage réseau à partir dudit réseau pour stocker des données à partir dudit réseau dans ledit dispositif de stockage de masse, destinés à satisfaire des demandes de recherches réseau à partir dudit réseau pour rechercher des données à partir dudit dispositif de stockage de masse vers ledit réseau, et destinés à transmettre des catégories prédéfinies de messages à partir dudit réseau vers ladite unité à processeur hôte afin de traiter dans ladite unité à processeur hôte, lesdits messages transmis comprenant toutes les demandes par un client du réseau pour exécuter des procédures définies par le client sur ledit dispositif serveur de réseau.

2. Dispositif selon la revendication 1, dans lequel ladite unité d'interface à processeur comprend :

   une unité de régulation du réseau (110a) pouvant être couplée audit réseau (122a) ;
   une unité de régulation de données (112a, 114a) pouvant être couplée audit dispositif de stockage de masse ;
   une mémoire tampon (116a) ; et
   des moyens (210, 212, 214, 220, 222, 224, 232, 234, 236, 252, 254, 256) dans ladite unité de régulation du réseau :
   destinés à transmettre vers ladite unité de régulation de données lesdites demandes de stockage réseau à partir dudit réseau pour stocker des données de stockage spécifiées à partir dudit réseau dans ledit dispositif de stockage de masse,
   destinés à transmettre lesdites données de stockage spécifiées à partir dudit réseau vers ladite mémoire tampon et à partir de ladite mémoire tampon vers ladite unité de régulation de données,
   destinés à transmettre vers ladite unité de régulation de données lesdites demandes de recherches réseau à partir dudit réseau pour rechercher des données de recherches spécifiées à partir dudit dispositif de stockage de masse vers ledit réseau,
   destinés à transmettre lesdites données de recherches spécifiées à partir de ladite unité de régulation de données vers ladite mémoire tampon et à partir de ladite mémoire tampon vers ledit réseau, et
   destinés à transmettre lesdites catégories prédéfinies de messages à partir dudit réseau vers ladite unité à processeur hôte afin d'être traitées par ladite unité à processeur hôte.

3. Dispositif selon la revendication 2, dans lequel ladite unité de régulation de données comprend :

   une unité de stockage à processeur (114a) pouvant être couplée audit dispositif de stockage de masse ;
   une unité de fichier à processeur (112a) ; et
   des moyens (310, 312, 314, 320, 324, 390, 396) dans ladite unité de fichiers à processeur :
   destinés à traduire lesdites demandes de stockage réseau à partir dudit réseau en demandes pour stocker des données à des emplacements spécifiés de stockage physique dans ledit dispositif de stockage de masse,
   destinés à donner l'ordre à ladite unité de stockage à processeur d'écrire des données à partir de ladite mémoire tampon dans lesdits emplacements spécifiés de stockage physiques dans ledit dispositif de stockage de masse,
   destinés à traduire lesdites demandes de recherches réseau à partir dudit réseau en demandes pour rechercher des données à partir des emplacements spécifiés de recherches physiques dans ledit dispositif stockage de masse, et
   destinés à donner l'ordre à ladite unité de stockage à processeur de rechercher des données à partir desdits emplacements spécifiés de recherches physiques dans ledit dispositif de stockage de masse vers ladite mémoire tampon si lesdites données à partir desdits emplacements spécifiés physiques ne sont pas déjà dans ladite mémoire tampon ; et
   des moyens (510, 512, 514, 524, 532, 534) dans ladite unité de stockage à processeur destinés à transmettre des données entre ladite mémoire tampon et ledit dispositif de stockage de masse.

4. Dispositif selon la revendication 1, dans lequel ladite unité d'interface à processeur comprend :

   un module de régulation du réseau (110a), comprenant une interface réseau (234, 214) couplée pour recevoir lesdites demandes de recherches réseau à partir dudit réseau ;
   un module de régulation du système fichier (112a, 114a), comprenant une interface pour dispositif de stockage de masse (540, 514) couplée audit dispositif de stockage de masse ; et
   un chemin de communication (120) directement couplé entre ledit module de régulation du réseau et ledit module de régulation du système fichier, ledit chemin de communication transportant des demandes de recherches locales préparées par ledit module de régulation du réseau en réponse auxdites demandes de recherches réseau, afin de rechercher des données de recherches spécifiées à partir dudit dispositif de stockage de masse,
   ledit module de régulation du système fichier recherchant lesdites données de recherches spécifiées à partir du dispositif de stockage de masse en réponse auxdites demandes de recherches locales et retournant lesdites données de recherches spécifiées vers ledit module de régulation du réseau,
   et ledit module de régulation du réseau préparant des messages de réponse contenant lesdites données de recherches spécifiées à partir dudit module de régulation du système fichier afin de les retourner par transmission sur ledit réseau.

5. Dispositif selon la revendication 4, dans lequel ledit module de régulation du système fichier retourne directement lesdites données de recherches spécifiées vers ledit module de régulation du réseau.

6. Dispositif selon la revendication 4 ou 5, dans lequel ladite interface réseau est en outre couplée pour recevoir lesdites demandes de stockage réseau à partir dudit réseau,

   dans lequel ledit module de régulation du réseau prépare en outre des demandes locales de stockage en réponse auxdites demandes de stockage réseau, afin de stocker des données spécifiées de stockage dans ledit dispositif de stockage de masse, ledit module de régulation du réseau communiquant lesdites demandes locales de stockage audit module de régulation du système fichier,
   et dans lequel ledit module de régulation du système fichier stocke en outre lesdites données spécifiées de stockage dans ledit dispositif de stockage de masse en réponse auxdites demandes locales de stockage.

7. Dispositif selon la revendication 6, dans lequel lesdites demandes locales de stockage sont communiquées audit module de régulation du système fichier via ledit chemin de communication (120).

8. Dispositif selon l'une des revendications précédentes, dans lequel ladite unité à processeur hôte exécute un système d'exploitation général, et dans lequel ladite unité d'interface à processeur n'exécute aucun système d'exploitation général.

9. Dispositif selon la revendication 8, dans lequel ledit système d'exploitation général est exécuté sur ladite unité à processeur hôte est un système d'exploitation UNIX.

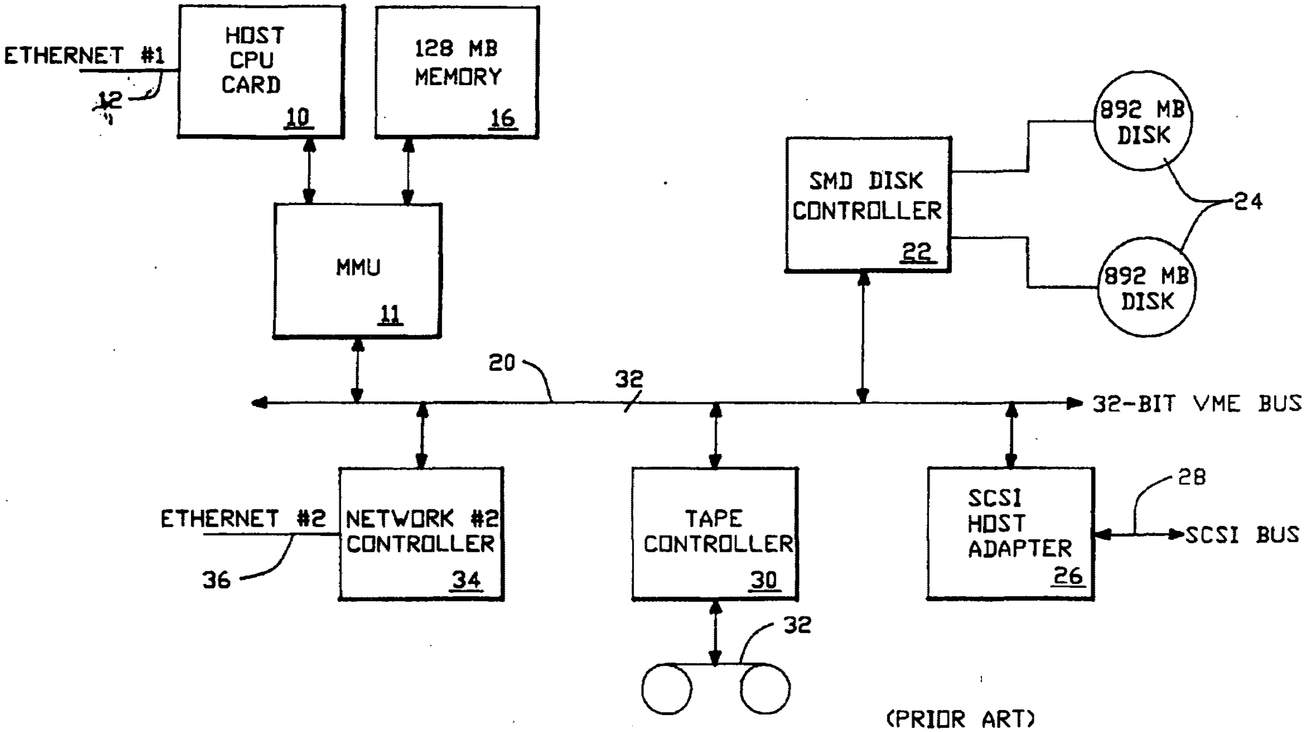
10. Dispositif selon l'une des revendications précédentes, dans lequel ladite unité d'interface à processeur comprend des moyens (214, 314 ,514) destinés à décoder toutes les demandes pour systèmes de fichier réseau (NFS) adressées à ladite unité d'interface à processeur à partir dudit réseau, destinés à mener toutes les procédures pour satisfaire lesdites demandes pour NFS, et destinés à coder tous les messages de réponse pour NFS afin de les retourner par transmission sur ledit réseau.

11. Dispositif selon l'une des revendications précédentes, dans lequel lesdits messages transmis comprennent tous les appels provenant d'un client du réseau audit dispositif serveur de réseau qui sont compris dans un jeu prédéfini d'appels en procédure à distance.

12. Dispositif selon l'une des revendications précédentes, comprenant en outre des moyens dans ladite unité d'interface à processeur destinés à détecter des demandes à partir dudit réseau pour une adresse dudit dispositif serveur de réseau, destinés à préparer un paquet de réponse à une telle demande d'adresse, et destinés à transmettre ledit paquet de réponse à travers ledit réseau.

13. Dispositif selon l'une des revendication précédentes, destiné à être en outre utilisé avec un deuxième réseau de données (122b), ladite unité d'interface à processeur pouvant être en outre couplée audit deuxième réseau, com-

prenant en outre des moyens (210, 214, 230, 250, 234, 254) dans ladite unité d'interface à processeur destinés à détecter une demande à partir dudit premier réseau pour acheminer un message contenu dans certains paquets vers une destination pouvant être atteinte à travers ledit deuxième réseau, et des moyens destinés à transmettre ledit message à travers ledit deuxième réseau.



(PRIOR ART)

FIG.-1

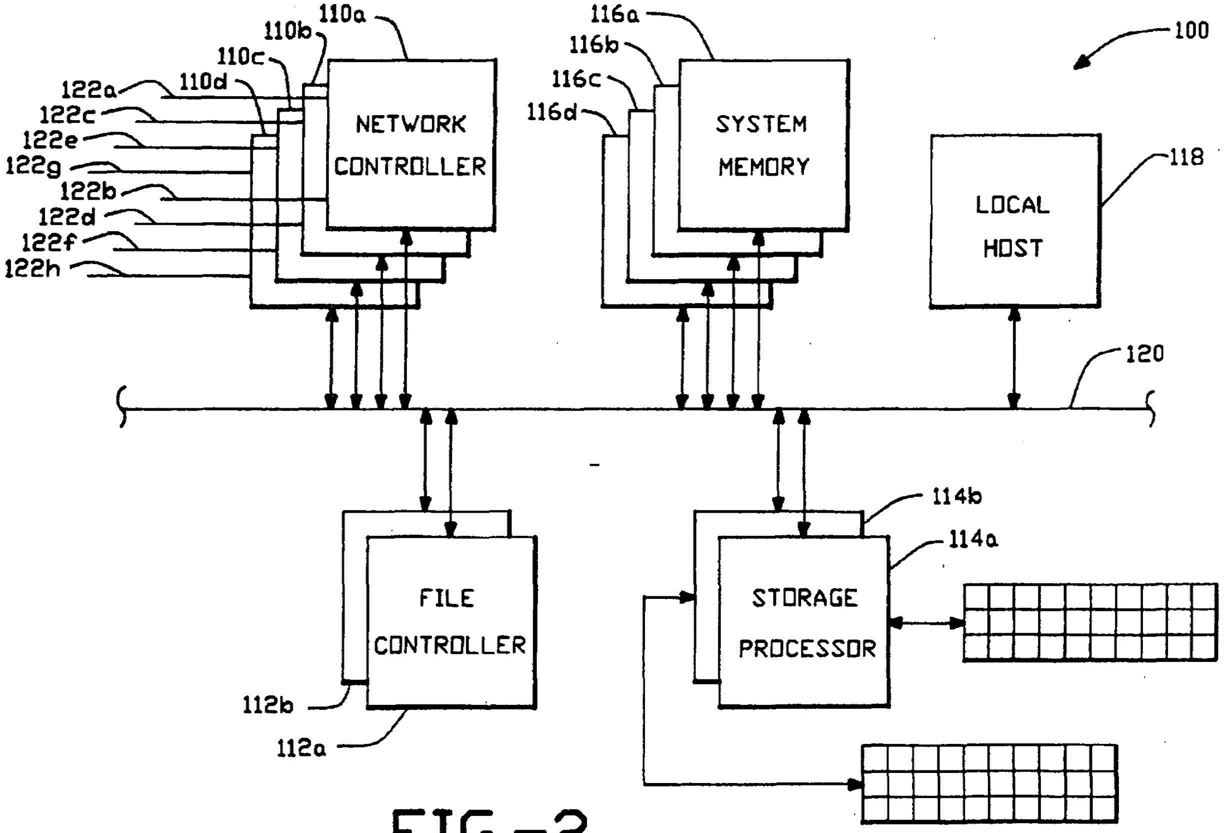

100
110a
110b
110c
110d
122a
122c
122e
122g
122b
122d
122f
122h
NETWORK CONTROLLER
116a
116b
116c
116d
SYSTEM MEMORY
LOCAL HOST
118
120
FILE CONTROLLER
112b
112a
STORAGE PROCESSOR
114b
114a

FIG.-2

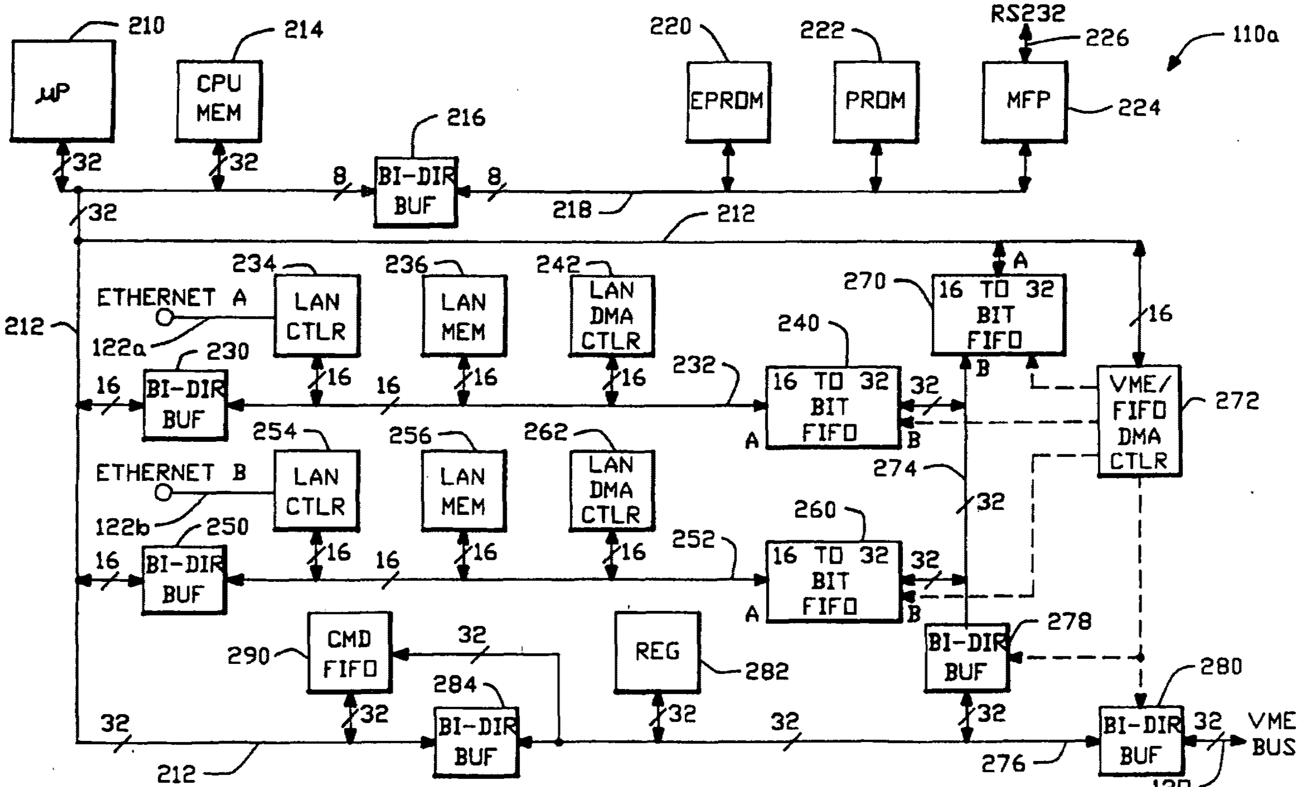


FIG.-3 (NETWORK CONTROLLER)

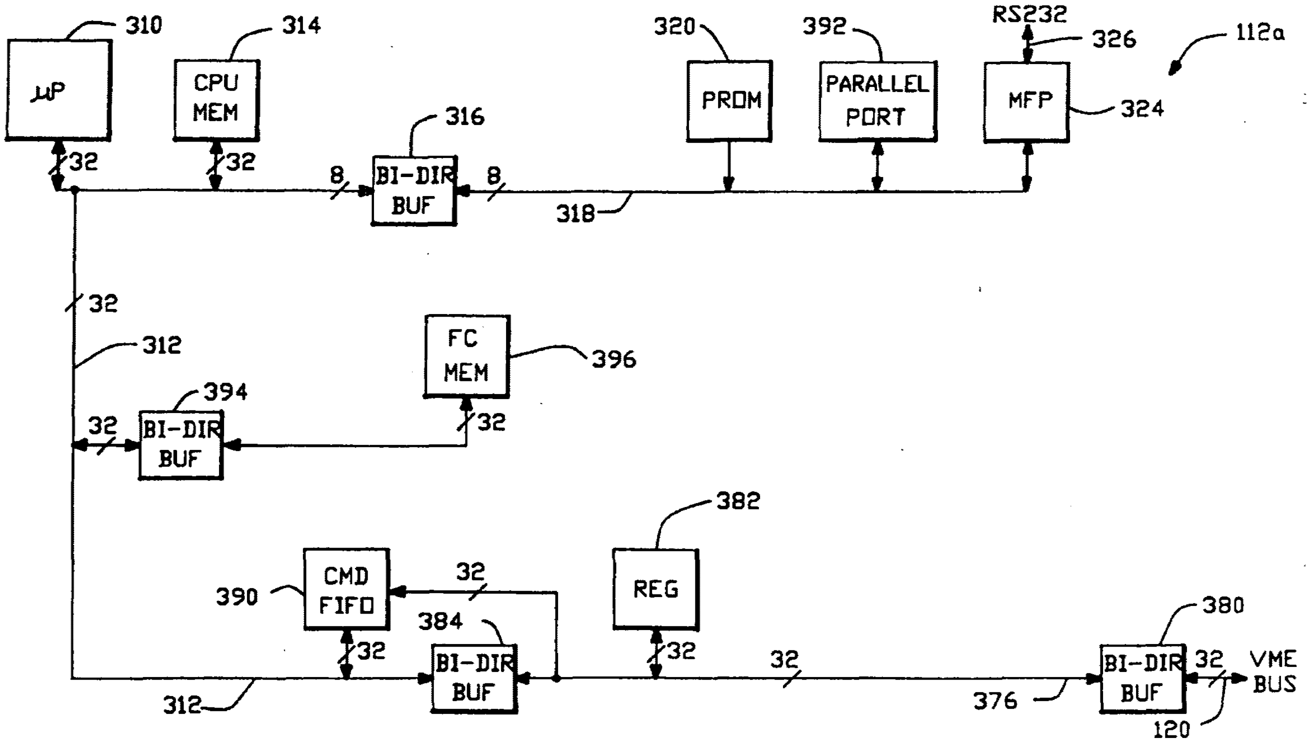


FIG.-4 (FILE CONTROLLER)

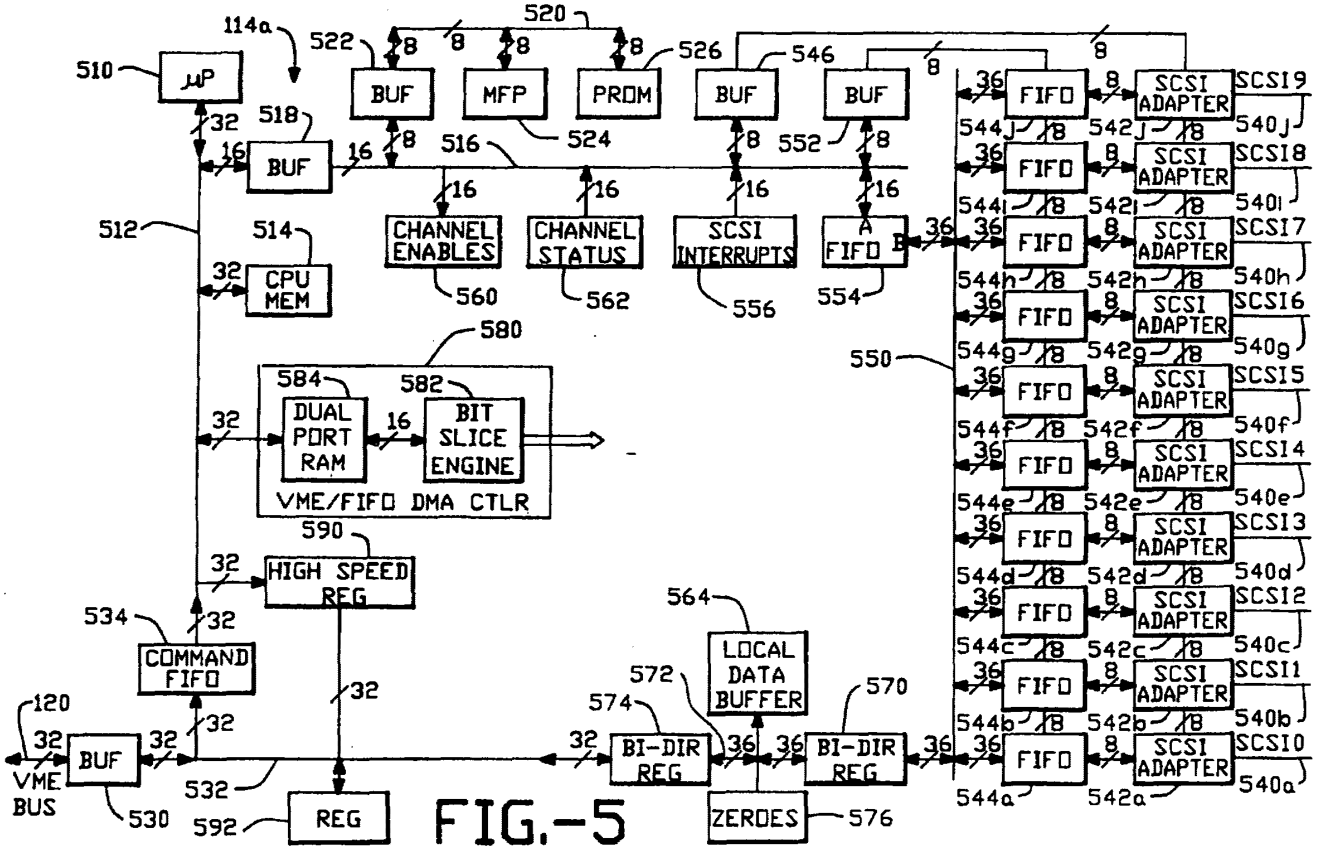

114a
510
μP
32
518
BUF
16
16
512
514
CPU MEM
32
522
BUF
8
520
8
MFP
8
PROM
8
526
524
516
CHANNEL ENABLES
16
560
CHANNEL STATUS
16
562
SCSI INTERRUPTS
16
556
546
BUF
8
552
BUF
8
16
A FIFO B
554
36
550
580
584
DUAL PORT RAM
16
582
BIT SLICE ENGINE
VME/FIFO DMA CTLR
590
HIGH SPEED REG
534
COMMAND FIFO
32
120
32
VME BUS
BUF
530
532
32
REG
592
564
LOCAL DATA BUFFER
572
574
BI-DIR REG
36
36
570
BI-DIR REG
ZEROES
576
FIFO
SCSI ADAPTER
SCSI9
544j
542j
540j
SCSI8
544i
542i
540i
SCSI7
544h
542h
540h
SCSI6
544g
542g
540g
SCSI5
544f
542f
540f
SCSI4
544e
542e
540e
SCSI3
544d
542d
540d
SCSI2
544c
542c
540c
SCSI1
544b
542b
540b
SCSI0
544a
542a
540a

FIG.-5

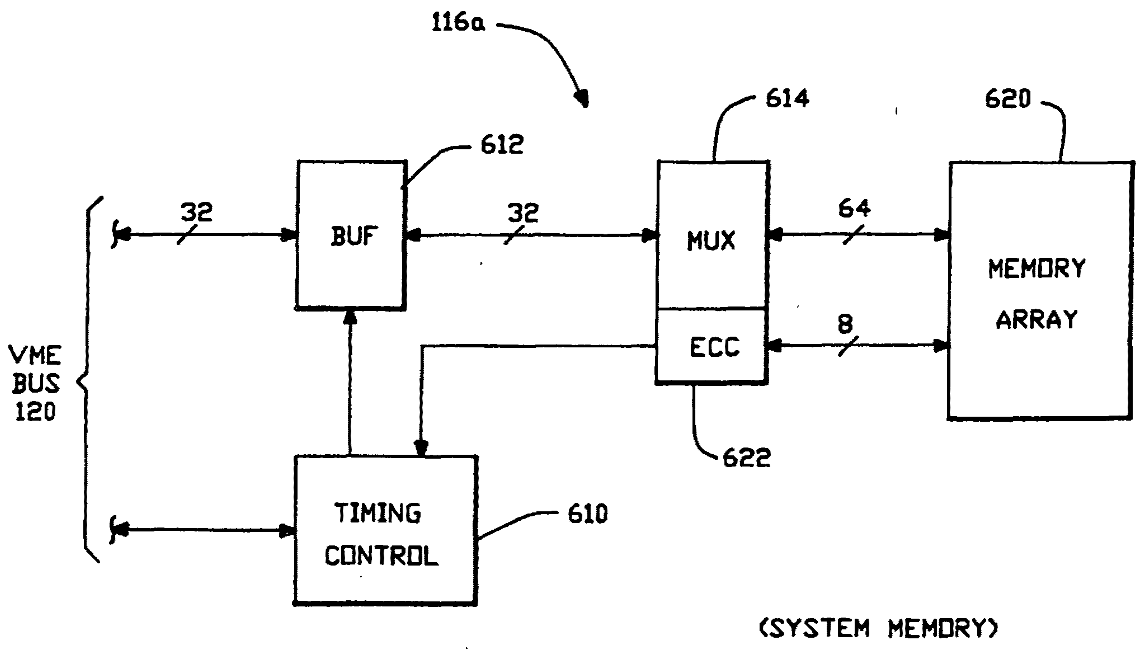

116a
614
620
612
32
32
64
BUF
MUX
MEMORY
ARRAY
8
ECC
VME
BUS
120
622
TIMING
CONTROL
610
(SYSTEM MEMORY)

FIG.-6

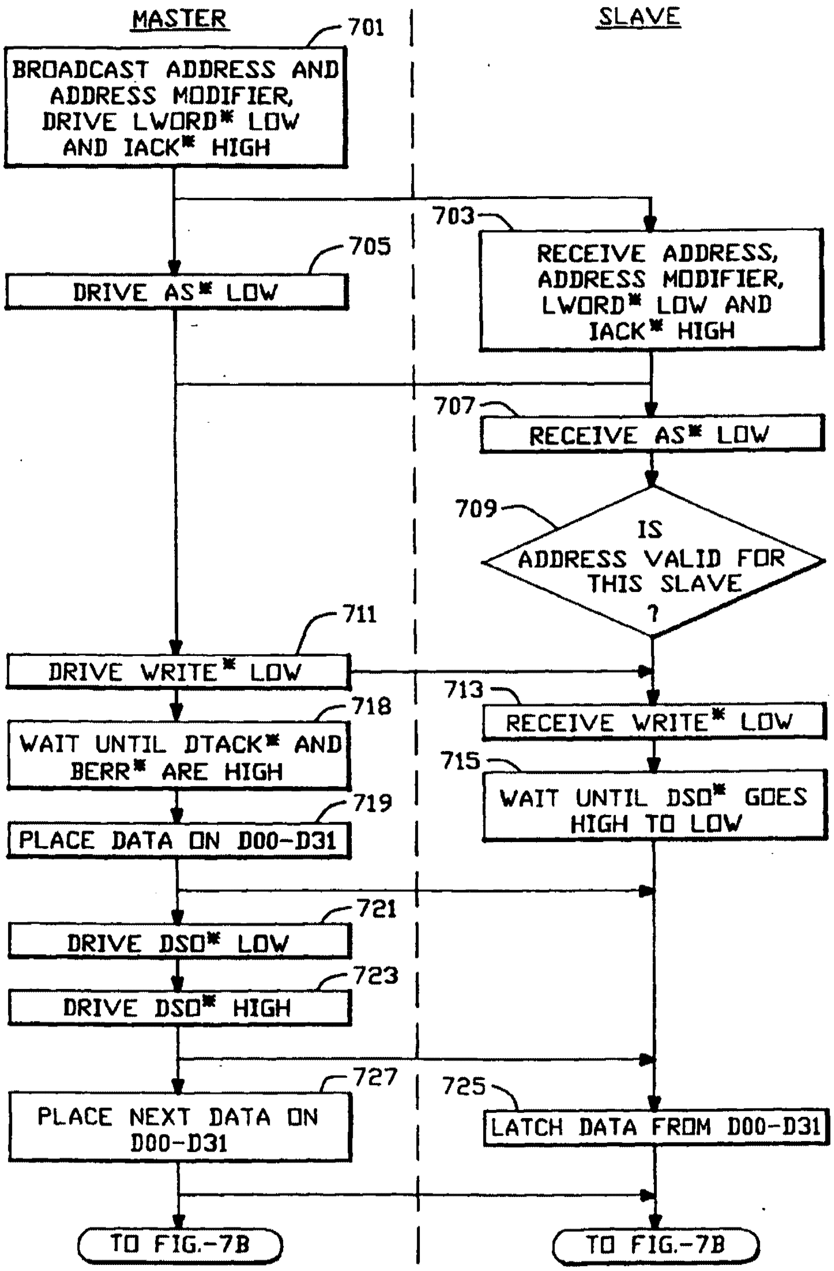

MASTER
SLAVE
701
BROADCAST ADDRESS AND ADDRESS MODIFIER, DRIVE LWORD* LOW AND IACK* HIGH
703
RECEIVE ADDRESS, ADDRESS MODIFIER, LWORD* LOW AND IACK* HIGH
705
DRIVE AS* LOW
707
RECEIVE AS* LOW
709
IS ADDRESS VALID FOR THIS SLAVE ?
711
DRIVE WRITE* LOW
713
RECEIVE WRITE* LOW
718
WAIT UNTIL DTACK* AND BERR* ARE HIGH
715
WAIT UNTIL DSO* GOES HIGH TO LOW
719
PLACE DATA ON D00-D31
721
DRIVE DSO* LOW
723
DRIVE DSO* HIGH
727
PLACE NEXT DATA ON D00-D31
725
LATCH DATA FROM D00-D31
TO FIG.-7B
TO FIG.-7B

FIG.-7A

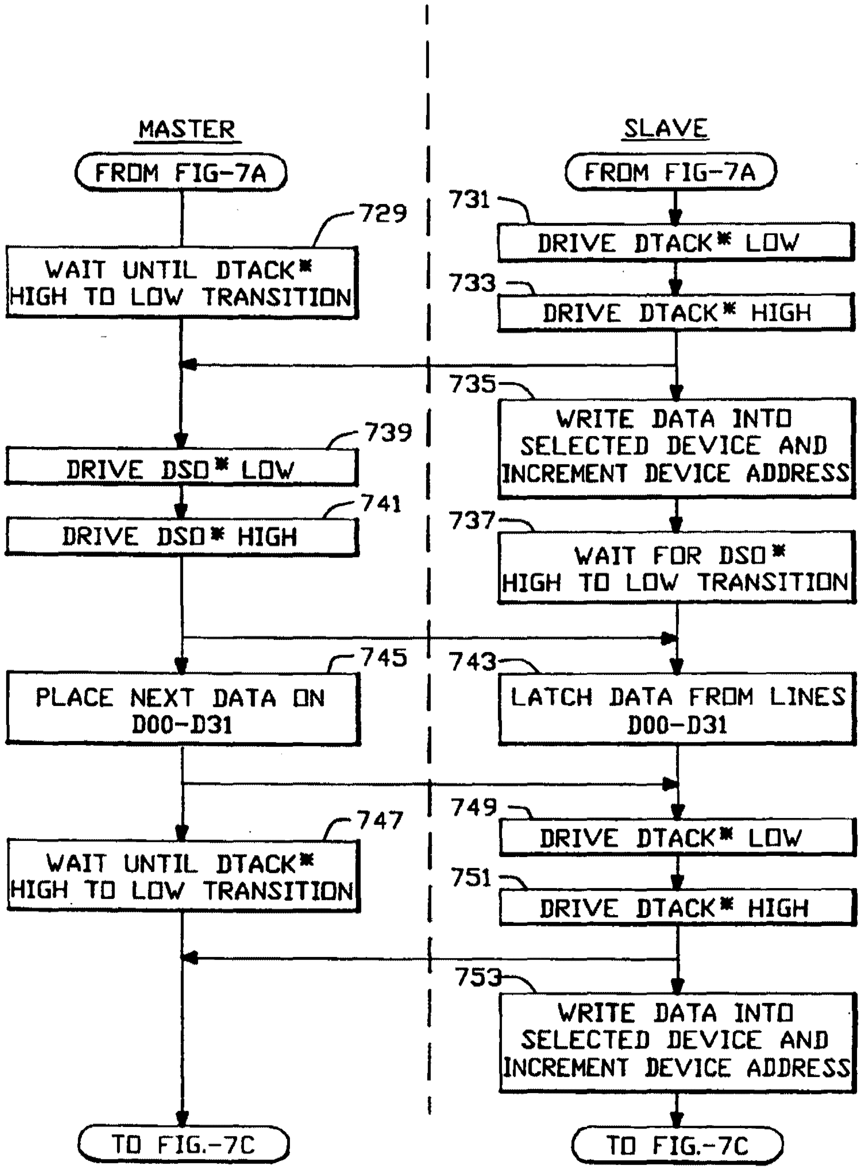

MASTER
FROM FIG-7A
729
WAIT UNTIL DTACK* HIGH TO LOW TRANSITION
739
DRIVE DSO* LOW
741
DRIVE DSO* HIGH
745
PLACE NEXT DATA ON D00-D31
747
WAIT UNTIL DTACK* HIGH TO LOW TRANSITION
TO FIG.-7C
SLAVE
FROM FIG-7A
731
DRIVE DTACK* LOW
733
DRIVE DTACK* HIGH
735
WRITE DATA INTO SELECTED DEVICE AND INCREMENT DEVICE ADDRESS
737
WAIT FOR DSO* HIGH TO LOW TRANSITION
743
LATCH DATA FROM LINES D00-D31
749
DRIVE DTACK* LOW
751
DRIVE DTACK* HIGH
753
WRITE DATA INTO SELECTED DEVICE AND INCREMENT DEVICE ADDRESS
TO FIG.-7C

FIG.-7B

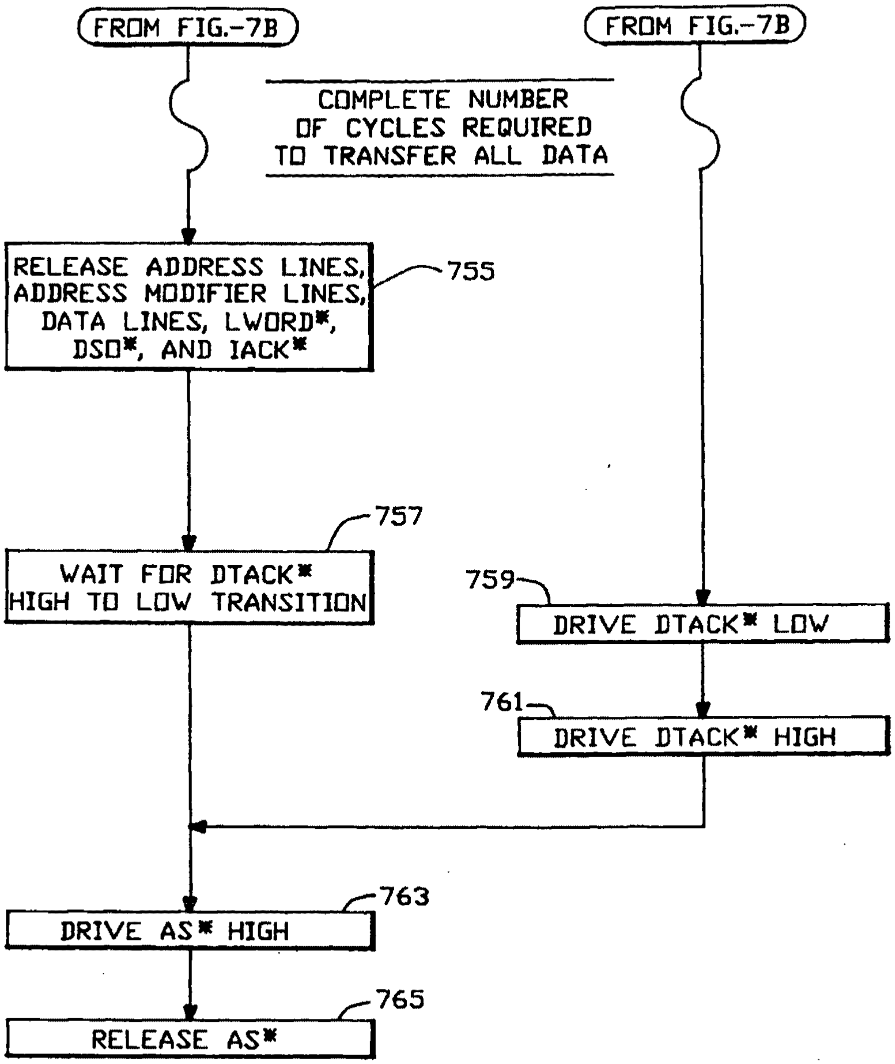

FROM FIG.-7B
FROM FIG.-7B
COMPLETE NUMBER OF CYCLES REQUIRED TO TRANSFER ALL DATA
RELEASE ADDRESS LINES, ADDRESS MODIFIER LINES, DATA LINES, LWORD*, DSO*, AND IACK*
755
757
WAIT FOR DTACK* HIGH TO LOW TRANSITION
759
DRIVE DTACK* LOW
761
DRIVE DTACK* HIGH
763
DRIVE AS* HIGH
765
RELEASE AS*

FIG.-7C

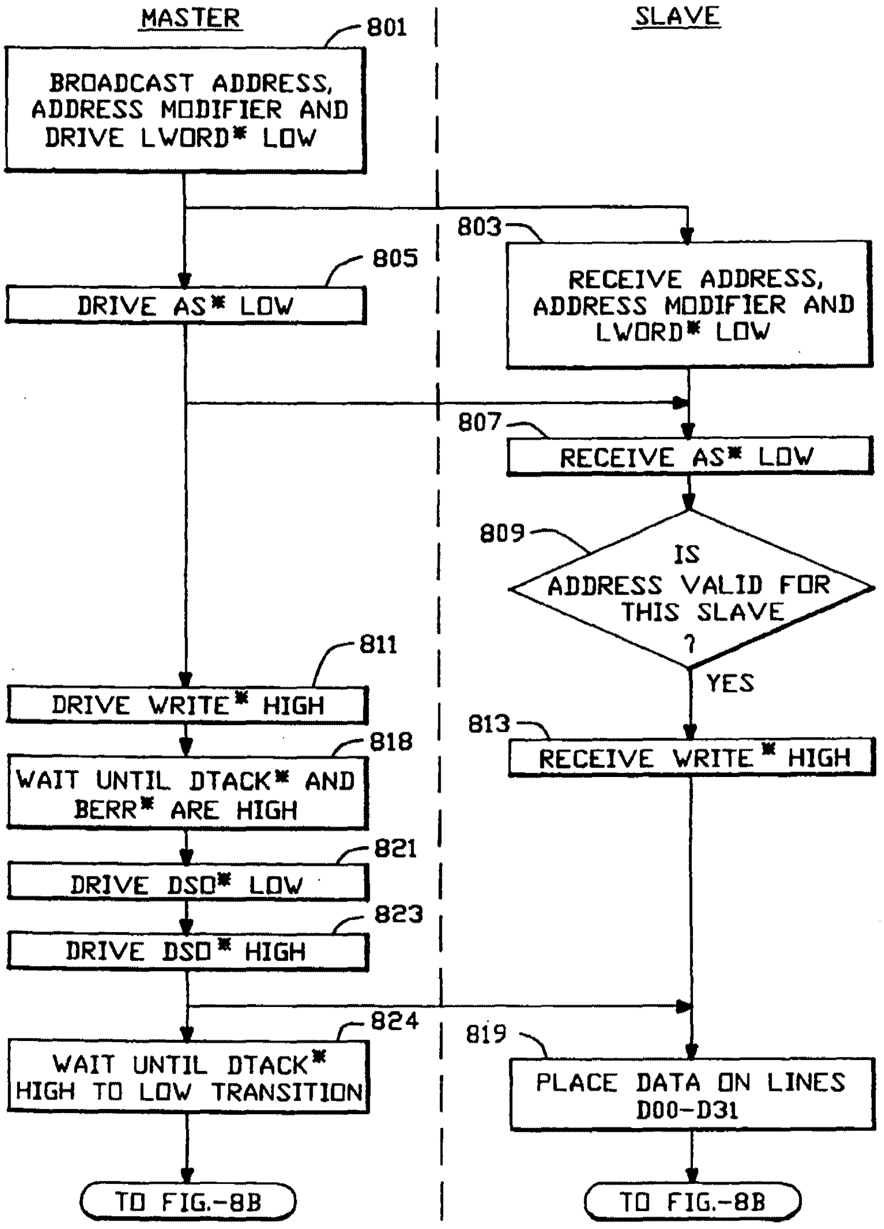

MASTER
SLAVE
801
BROADCAST ADDRESS, ADDRESS MODIFIER AND DRIVE LWORD* LOW
803
RECEIVE ADDRESS, ADDRESS MODIFIER AND LWORD* LOW
805
DRIVE AS* LOW
807
RECEIVE AS* LOW
809
IS ADDRESS VALID FOR THIS SLAVE ?
YES
811
DRIVE WRITE* HIGH
813
RECEIVE WRITE* HIGH
818
WAIT UNTIL DTACK* AND BERR* ARE HIGH
821
DRIVE DS0* LOW
823
DRIVE DS0* HIGH
824
WAIT UNTIL DTACK* HIGH TO LOW TRANSITION
819
PLACE DATA ON LINES D00-D31
TO FIG.-8B
TO FIG.-8B

FIG.-8A

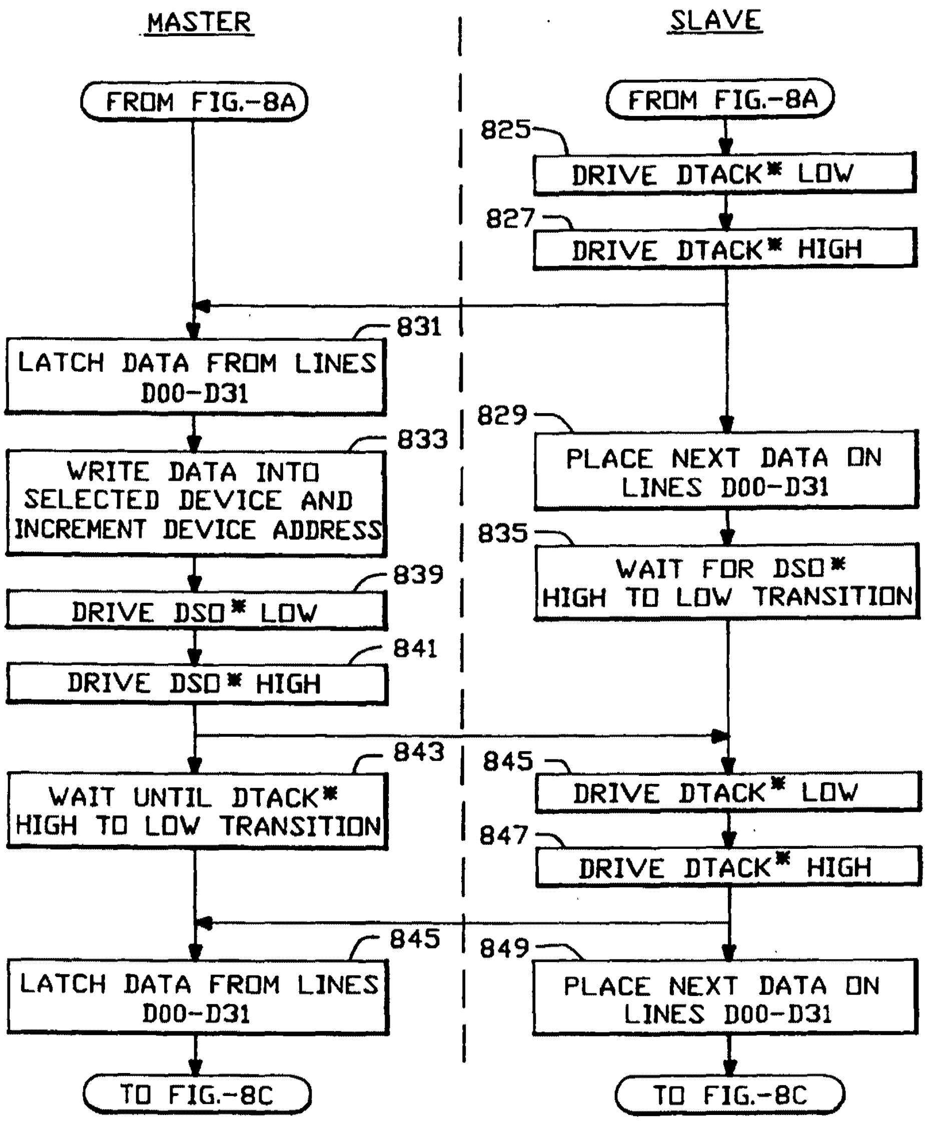

MASTER
SLAVE
FROM FIG.-8A
FROM FIG.-8A
825
DRIVE DTACK* LOW
827
DRIVE DTACK* HIGH
831
LATCH DATA FROM LINES D00-D31
829
PLACE NEXT DATA ON LINES D00-D31
833
WRITE DATA INTO SELECTED DEVICE AND INCREMENT DEVICE ADDRESS
835
WAIT FOR DSO* HIGH TO LOW TRANSITION
839
DRIVE DSO* LOW
841
DRIVE DSO* HIGH
843
WAIT UNTIL DTACK* HIGH TO LOW TRANSITION
845
DRIVE DTACK* LOW
847
DRIVE DTACK* HIGH
845
LATCH DATA FROM LINES D00-D31
849
PLACE NEXT DATA ON LINES D00-D31
TO FIG.-8C
TO FIG.-8C

FIG.-8B

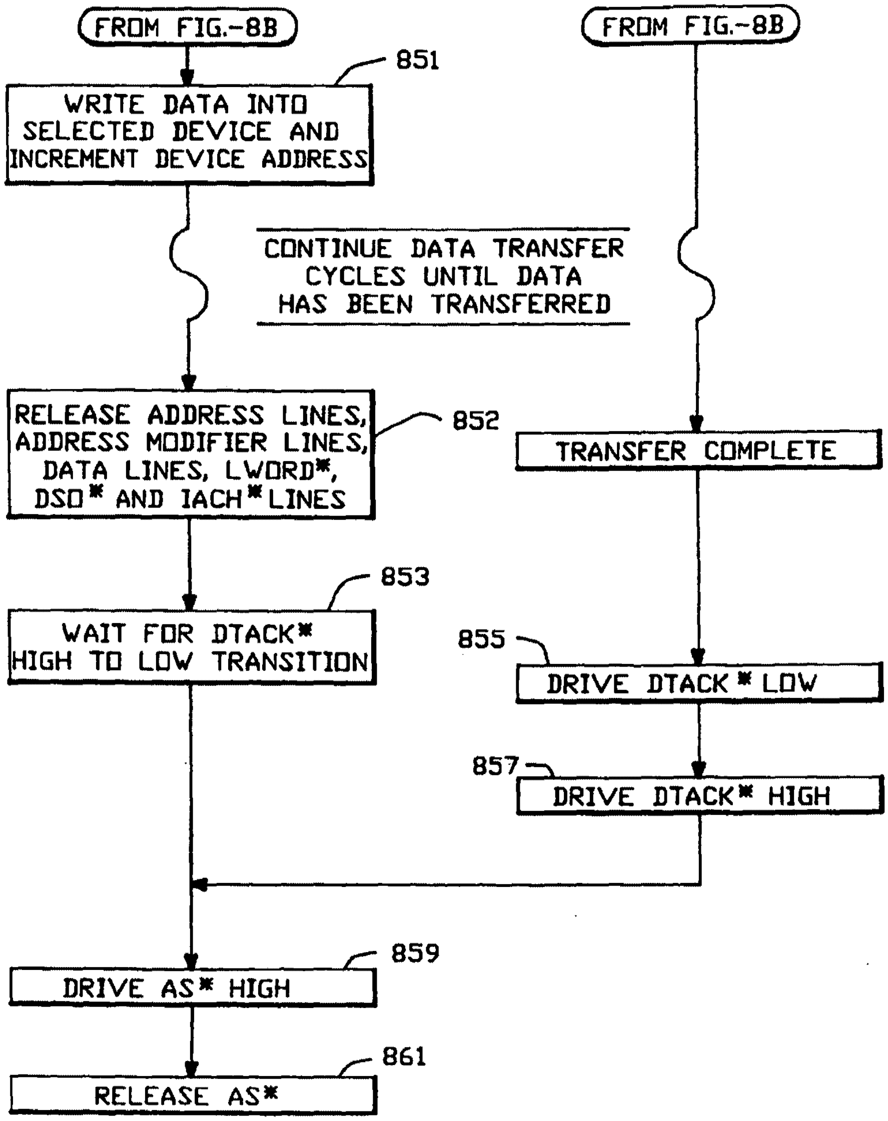

FROM FIG.-8B
FROM FIG.-8B
851
WRITE DATA INTO SELECTED DEVICE AND INCREMENT DEVICE ADDRESS
CONTINUE DATA TRANSFER CYCLES UNTIL DATA HAS BEEN TRANSFERRED
RELEASE ADDRESS LINES, ADDRESS MODIFIER LINES, DATA LINES, LWORD*, DSO* AND IACH* LINES
852
TRANSFER COMPLETE
853
WAIT FOR DTACK* HIGH TO LOW TRANSITION
855
DRIVE DTACK* LOW
857
DRIVE DTACK* HIGH
859
DRIVE AS* HIGH
861
RELEASE AS*

FIG.-8C

B30
14

לשימוש הלשכה
For Office Use

095447	מספר: Number
21-08-1990	תאריך: Date
	הוקדם/נדחה Ante/Post-dated

C: 11071

חוק הפטנטים, תשכ"ז — 1967
PATENT LAW, 5727 - 1967

בקשה לפטנט
Application for Patent

אני, (שם המבקש, מענו ולגבי גוף מאוגד - מקום התאגדותו)
I (Name and address of applicant, and in case of body corporate-place of incorporation)

AUSPEX SYSTEMS, INC.,
2952 Bunker Hill Lane,
Santa Clara,
Ca. 95054, U.S.A.

(Incorporated in the State of California, USA)

00095447 01029
21

בעל אמצאה מכח .... ששמה הוא
Owner, by virtue of .... of an invention the title of which is

(בעברית) (Hebrew) אריכטקטורה מקבילה למערכת שירות תיקים

(באנגלית) (English) PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE

מבקש בזאת כי ינתן לי עליה פטנט
hereby apply for a patent to be granted to me in respect thereof.

• בקשת חלוקה — Application of Division	• בקשת פטנט מוסף — Application for Patent Addition	• דרישה דין קדימה Priority Claim		
מבקשת פטנט from Application	• לבקשה/לפטנט to Patent/Appl.	מספר/סימן Number/Mark	תאריך Date	מדינת האגוד Convention Country
No. .... מס' dated .... מיום	No. .... מס' dated .... מיום	07/ 404,959	8.9.89	U.S.A.

• יפוי כח: כללי / מיוחד — רצוף בזה / עוד יוגש
P.O.A.: general/individual-attached/to be filed later-
filed in case .... הוגש בענין

המען למסירת מסמכים בישראל
Address for Service in Israel
Sanford T. Colb & Co. סנפורד ט. קולב
P.O.B. 2273 ת.ד. 2273
Rehovot 76 122 רחובות 76122

חתימת המבקש
Signature of Applicant
For the Applicant,

Sanford T. Colb & Co.
C: 11071

היום 20 This, בחודש VIII of, שנת 1990 of the year

לשימוש הלשכה
For Office Use

תאריך הפרסום: Publication date: 30-05-1994
תאריך החיתום: Sealing date:

טופס זה, כשהוא מוטבע בחותם לשכת הפטנטים ומושלם במספר ובתאריך ההגשה, הנו אישור להגשת הבקשה שפרטיה רשומים לעיל.
This from, impressed with the Seal of the Patent Office and indicating the number and date of filing, certifies the filing of the application the particulars of which are set our above.

Delete whatever is inapplicable מחק את המיותר *

אריכטקטורה מקבילה למערכת שירות תיקים

PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE

AUSPEX SYSTEMS, INC.
C: 11071

BACKGROUND OF THE INVENTION

Field of the Invention

The invention relates to computer data networks, and more particularly, to network file server architectures for computer networks.

Description of the Related Art

Over the past ten years, remarkable increases in hardware price/performance ratios have caused a startling shift in both technical and office computing environments. Distributed workstation-server networks are displacing the once pervasive dumb terminal attached to mainframe or minicomputer. To date, however, network I/O limitations have constrained the potential performance available to workstation users. This situation has developed in part because dramatic jumps in microprocessor performance have exceeded increases in network I/O performance.

In a computer network, individual user workstations are referred to as clients, and shared resources for filing, printing, data storage and wide-area communications are referred to as servers. Clients and servers are all considered nodes of a network. Client nodes use standard communications protocols to exchange service requests and responses with server nodes.

Present-day network clients and servers usually run the DOS, MacIntosh OS, OS/2, or Unix operating systems. Local networks are usually Ethernet or Token Ring at the high end, Arcnet in the midrange, or LocalTalk or StarLAN at the low end. The client-server communication protocols are fairly strictly dictated by the operating system environment -- usually one of several proprietary schemes for PCs (NetWare, 3Plus, Vines, LANManager, LANServer); AppleTalk for MacIntoshes; and TCP/IP with NFS or RFS for Unix. These protocols are all well-known in the industry.

Unix client nodes typically feature a 16- or 32-bit microprocessor with 1-8 MB of primary memory, a 640 x 1024 pixel display, and a built-in network interface. A 40-100 MB local disk is often optional. Low-end examples are 80286-based PCs or 68000-based MacIntosh I's; mid-range machines include 80386 PCs, MacIntosh II's, and 680X0-based Unix workstations; high-end machines include RISC-based DEC, HP, and Sun Unix workstations. Servers are typically nothing more than repackaged client nodes, configured in 19-inch racks rather than desk sideboxes. The extra space of a 19-inch rack is used for additional backplane slots, disk or tape drives, and power supplies.

Driven by RISC and CISC microprocessor developments, client workstation performance has increased by more than a factor of ten in the last few years. Concurrently, these extremely fast clients have also gained an appetite for data that remote servers are unable to satisfy. Because the I/O shortfall is most dramatic in the Unix environment, the description of the preferred embodiment of the present invention will focus on Unix file servers. The architectural principles that solve the Unix server I/O problem, however, extend easily to server performance bottlenecks in other operating system environments as well. Similarly, the description of the preferred embodiment will focus on Ethernet implementations, though the principles extend easily to other types of networks.

In most Unix environments, clients and servers exchange file data using the Network File System ("NFS"), a standard promulgated by Sun Microsystems and now widely adopted by the Unix community. NFS is defined in a document entitled, "NFS: Network File System Protocol Specification," Request For Comments (RFC) 1094, by Sun Microsystems, Inc. (March 1989). This document is incorporated herein by reference in its entirety.

While simple and reliable, NFS is not optimal. Clients using NFS place considerable demands upon both networks and NFS servers supplying clients with NFS data. This demand is particularly acute for so-called diskless clients that have no local disks and therefore depend on a file server for application binaries and virtual memory paging as well as data. For these Unix client-server configurations, the ten-to-one increase in client power has not been matched by a ten-to-one increase in Ethernet capacity, in disk speed, or server disk-to-network I/O throughput.

The result is that the number of diskless clients that a single modern high-end server can adequately support has dropped to between 5-10, depending on client power and application workload. For clients containing small local disks for applications and paging, referred to as dataless clients, the client-to-server ratio is about twice this, or between 10-20.

Such low client/server ratios cause piecewise network configurations in which each local Ethernet contains isolated traffic for its own 5-10 (diskless) clients and dedicated server. For overall connectivity, these local networks are usually joined together with an Ethernet backbone or, in the future, with an FDDI backbone. These backbones are typically connected to the local networks either by IP routers or MAC-level bridges, coupling the local networks together directly, or by a second server functioning as a network interface, coupling servers for all the local networks together.

In addition to performance considerations, the low client-to-server ratio creates computing problems in several additional ways:

1. Sharing. Development groups of more than 5-10 people cannot share the same server, and thus cannot easily share files without file replication and manual, multi-server updates. Bridges or routers are a partial solution but inflict a performance penalty due to more network hops.

2. Administration. System administrators must maintain many limited-capacity servers rather than a few more substantial servers. This burden includes network administration, hardware maintenance, and user account administration.

3. File System Backup. System administrators or operators must conduct multiple file system backups, which can be onerously time consuming tasks. It is also expensive to duplicate backup peripherals on each server (or every few servers if slower network backup is used).

4. Price Per Seat. With only 5-10 clients per server, the cost of the server must be shared by only a small number of users. The real cost of an entry-level Unix workstation is therefore significantly greater, often as much as 140% greater, than the cost of the workstation alone.

The widening I/O gap, as well as administrative and economic considerations, demonstrates a need for higher-performance, larger-capacity Unix file servers.

Conversion of a display-less workstation into a server may address disk capacity issues, but does nothing to address fundamental I/O limitations. As an NFS server, the one-time workstation must sustain 5-10 or more times the network, disk, backplane, and file system throughput than it was designed to support as a client. Adding larger disks, more network adaptors, extra primary memory, or even a faster processor do not resolve basic architectural I/O constraints; I/O throughput does not increase sufficiently.

Other prior art computer architectures, while not specifically designed as file servers, may potentially be used as such. In one such well-known architecture, a CPU, a memory unit, and two I/O processors are connected to a single bus. One of the I/O processors operates a set of disk drives, and if the architecture is to be used as a server, the other I/O processor would be connected to a network. This architecture is not optimal as a file server, however, at least because the two I/O processors cannot handle network file requests without involving the CPU. All network file requests that are received by the network I/O processor are first transmitted to the CPU, which makes appropriate requests to the disk-I/O processor for satisfaction of the network request.

In another such computer architecture, a disk controller CPU manages access to disk drives, and several other CPUs, three for example, may be clustered around the disk controller CPU. Each of the other CPUs can be connected to its own network. The network CPUs are each connected to the disk controller CPU as well as to each other for interprocessor communication. One of the disadvantages of this computer architecture is that each CPU in the system runs its own complete operating system. Thus, network file server requests must be handled by an operating system which is also heavily loaded with facilities and processes for performing a large number of other, non file-server tasks. Additionally, the interprocessor communication is not optimized for file server type requests.

In yet another computer architecture, a plurality of CPUs, each having its own cache memory for data and instruction storage, are connected to a common bus with a system memory and a disk controller. The disk controller and each of the CPUs have direct memory access to the system memory, and one or more of the CPUs can be connected to a network. This architecture is disadvantageous as a file server because, among other things, both file data and the instructions for the CPUs reside in the same system memory. There will be instances, therefore, in which the CPUs must stop running while they wait for large blocks of file data to be transferred between system memory and the network CPU. Additionally, as with both of the previously described computer architectures, the entire operating system runs on each of the CPUs, including the network CPU.

In yet another type of computer architecture, a large number of CPUs are connected together in a hypercube topology. One of more of these CPUs can be connected to networks, while another can be connected to disk drives. This architecture is also disadvantageous as a file server because, among other things, each processor runs the entire operating system. Interprocessor communication is also not optimal for file server applications.

SUMMARY OF THE INVENTION

The present invention involves a new, server-specific I/O architecture that is optimized for a Unix file server's most common actions -- file operations. Roughly stated, the invention involves a file server architecture comprising one or more network controllers, one or more file controllers, one or more storage processors, and a system or buffer memory, all connected over a message passing bus and operating in parallel with the Unix host processor. The network controllers each connect to one or more network, and provide all protocol processing between the network layer data format and an internal file server format for communicating client requests to other processors in the server. Only those data packets which cannot be interpreted by the network controllers, for example client requests to run a client-defined program on the server, are transmitted to the Unix host for processing. Thus the network controllers, file controllers and storage processors contain only small parts of an overall operating system, and each is optimized for the particular type of work to which it is dedicated.

Client requests for file operations are transmitted to one of the file controllers which, independently of the Unix host, manages the virtual file system of a mass storage device which is coupled to the storage processors. The file controllers may also control data buffering between the storage processors and the network controllers, through the system memory. The file controllers preferably each include a local buffer memory for caching file control information, separate from the system memory for caching file data. Additionally, the network controllers, file processors and storage processors are all designed to avoid any instruction fetches from the system memory, instead keeping all instruction memory separate and local. This arrangement eliminates contention on the backplane between microprocessor instruction fetches and transmissions of message and file data.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described with respect to particular embodiments thereof, and reference will be made to the drawings, in which:

Fig. 1. is a block diagram of a prior art file server architecture;

Fig. 2 is a block diagram of a file server architecture according to the invention;

Fig. 3 is a block diagram of one of the network controllers shown in Fig. 2;

Fig. 4 is a block diagram of one of the file controllers shown in Fig. 2;

Fig. 5 is a block diagram of one of the storage processors shown in Fig. 2;

Fig. 6 is a block diagram of one of the system memory cards shown in Fig. 2;

Figs. 7A-C are a flowchart illustrating the operation of a fast transfer protocol BLOCK WRITE cycle; and

Figs. 8A-C are a flowchart illustrating the operation of a fast transfer protocol BLOCK READ cycle.

DETAILED DESCRIPTION

For comparison purposes and background, an illustrative prior-art file server architecture will first be described with respect to Fig. 1. Fig. 1 is an overall block diagram of a conventional prior-art Unix-based file server for Ethernet networks. It consists of a host CPU card 10 with a single microprocessor on board. The host CPU card 10 connects to an Ethernet #1 12, and it connects via a memory management unit (MMU) 11 to a large memory array 16. The host CPU card 10 also drives a keyboard, a video display, and two RS232 ports (not shown). It also connects via the MMU 11 and a standard 32-bit VME bus 20 to various peripheral devices, including an SMD disk controller 22 controlling one or two disk drives 24, a SCSI host adaptor 26 connected to a SCSI bus 28, a tape controller 30 connected to a quarter-inch tape drive 32, and possibly a network #2 controller 34 connected to a second Ethernet 36. The SMD disk controller 22 can communicate with memory array 16 by direct memory access via bus 20 and MMU 11, with either the disk controller or the MMU acting as a bus master.

This configuration is illustrative; many variations are available.

The system communicates over the Ethernets using industry standard TCP/IP and NFS protocol stacks. A description of protocol stacks in general can be found in Tanenbaum, "Computer Networks" (Second Edition, Prentice Hall: 1988). File server protocol stacks are described at pages 535-546. The Tanenbaum reference is incorporated herein by reference.

Basically, the following protocol layers are implemented in the apparatus of Fig. 1:

Network Layer. The network layer converts data packets between a formal specific to Ethernets and a format which is independent of the particular type of network used. the Ethernet-specific format which is used in the apparatus of Fig. 1 is described in Hornig, "A Standard For The Transmission of IP Datagrams Over Ethernet Networks," RFC 894 (April 1984), which is incorporated herein by reference.

The Internet Protocol (IP) Layer. This layer provides the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. For messages to be sent from the file server to a client, a higher level in the server calls the IP module, providing the internet address of the destination client and the message to transmit. The IP module performs any required fragmentation of the message to accommodate packet size limitations of any intervening gateway, adds internet headers to each fragment, and calls on the network layer to transmit the resulting internet datagrams. The internet header includes a local network destination address (translated from the internet address) as well as other parameters.

For messages received by the IP layer from the network layer, the IP module determines from the internet address whether the datagram is to be forwarded to another host on another network, for example on a second Ethernet such as 36 in Fig. 1, or whether it is intended for the server itself. If it is intended for another host on the second network, the IP module determines a local net address for the destination and calls on the local network layer for that network to send the datagram. If the datagram is intended for an application program within the server, the IP layer strips off the header and passes the remaining portion of the message to the appropriate next higher layer. The internet protocol standard used in the illustrative apparatus of Fig. 1 is specified in Information Sciences Institute, "Internet Protocol, DARPA Internet Program Protocol Specification," RFC 791

(September 1981), which is incorporated herein by reference.

TCP/UDP Layer. This layer is a datagram service with more elaborate packaging and addressing options than the IP layer. For example, whereas an IP datagram can hold about 1,500 bytes and be addressed to hosts, UDP datagrams can hold about 64KB and be addressed to a particular port within a host. TCP and UDP are alternative protocols at this layer; applications requiring ordered reliable delivery of streams of data may use TCP, whereas applications (such as NFS) which do not require ordered and reliable delivery may use UDP.

The prior art file server of Fig. 1 uses both TCP and UDP. It uses UDP for file server-related services, and uses TCP for certain other services which the server provides to network clients. The UDP is specified in Postel, "User Datagram Protocol," RFC 768 (August 28, 1980), which is incorporated herein by reference. TCP is specified in Postel, "Transmission Control Protocol," RFC 761 (January 1980) and RFC 793 (September 1981), which is also incorporated herein by reference.

XDR/RPC Layer. This layer provides functions callable from higher level programs to run a designated procedure on a remote machine. It also provides the decoding necessary to permit a client machine to execute a procedure on the server. For example, a caller process in a client node may send a call message to the server of Fig. 1. The call message includes a specification of the desired procedure, and its parameters. The message is passed up the stack to the RPC layer, which calls the appropriate procedure within the server. When the procedure is complete, a reply message is generated and RPC passes it back down the stack and over the network to the caller client. RPC is described in Sun Microsystems, Inc., "RPC: Remote Procedure Call Protocol Specification, Version 2," RFC 1057 (June 1988), which is incorporated herein by reference.

RPC uses the XDR external data representation standard to represent information passed to and from the underlying UDP layer. XDR is merely a data encoding standard, useful for transferring data between different computer architectures. Thus, on the network side of the XDR/RPC layer, information is machine-independent; on the host application side, it may not be. XDR is described in Sun Microsystems, Inc., "XDR: External Data Representation Standard," RFC 1014 (June 1987), which is incorporated herein by reference.

NFS Layer. The NFS ("network file system") layer is one of the programs available on the server which an RPC request can call. The combination of host address, program number, and procedure number in an RPC request can specify one remote NFS procedure to be called.

Remote procedure calls to NFS on the file server of Fig. 1 provide transparent, stateless, remote access to shared files on the disks 24. NFS assumes a file system that is hierarchical, with directories as all but the bottom level of files. Client hosts can call any of about 20 NFS procedures including such procedures as reading a specified number of bytes from a specified file; writing a specified number of bytes to a specified file; creating, renaming and removing specified files; parsing directory trees; creating and removing directories; and reading and setting file attributes. The location on disk to which and from which data is stored and retrieved is always specified in logical terms, such as by a file handle or Inode designation and a byte offset. The details of the actual data storage are hidden from the client. The NFS procedures, together with possible higher level modules such as Unix VFS and UFS, perform all conversion of logical data addresses to physical data addresses such as drive, head, track and sector identification. NFS is specified in Sun Microsystems, Inc., "NFS: Network File System Protocol

Specification," RFC 1094 (March 1989), incorporated herein by reference.

With the possible exception of the network layer, all the protocol processing described above is done in software, by a single processor in the host CPU card 10. That is, when an Ethernet packet arrives on Ethernet 12, the host CPU 10 performs all the protocol processing in the NFS stack, as well as the protocol processing for any other application which may be running on the host 10. NFS procedures are run on the host CPU 10, with access to memory 16 for both data and program code being provided via MMU 11. Logically specified data addresses are converted to a much more physically specified form and communicated to the SMD disk controller 22 or the SCSI bus 28, via the VME bus 20, and all disk caching is done by the host CPU 10 through the memory 16. The host CPU card 10 also runs procedures for performing various other functions of the file server, communicating with tape controller 30 via the VME bus 20. Among these are client-defined remote procedures requested by client workstations.

If the server serves a second Ethernet 36, packets from that Ethernet are transmitted to the host CPU 10 over the same VME bus 20 in the form of IP datagrams. Again, all protocol processing except for the network layer is performed by software processes running on the host CPU 10. In addition, the protocol processing for any message that is to be sent from the server out on either of the Ethernets 12 or 36 is also done by processes running on the host CPU 10.

It can be seen that the host CPU 10 performs an enormous amount of processing of data, especially if 5-10 clients on each of the two Ethernets are making file server requests and need to be sent responses on a frequent basis. The host CPU 10 runs a multitasking Unix operating system, so each incoming request need not wait for the previous request to be completely processed and returned before being processed. Multiple processes are activated on the host CPU 10 for performing different stages of the processing of different requests, so many requests may be in process at the same time. But there is only one CPU on the card 10, so the processing of these requests is not accomplished in a truly parallel manner. The processes are instead merely time sliced. The CPU 10 therefore represents a major bottleneck in the processing of file server requests.

Another bottleneck occurs in MMU 11, which must transmit both instructions and data between the CPU card 10 and the memory 16. All data flowing between the disk drives and the network passes through this interface at least twice.

Yet another bottleneck can occur on the VME bus 20, which must transmit data among the SMD disk controller 22, the SCSI host adaptor 26, the host CPU card 10, and possibly the network #2 controller 34.

PREFERRED EMBODIMENT-OVERALL HARDWARE ARCHITECTURE

In Fig. 2 there is shown a block diagram of a network file server 100 according to the invention. It can include multiple network controller (NC) boards, one or more file controller (FC) boards, one or more storage processor (SP) boards, multiple system memory boards, and one or more host processors. The particular embodiment shown in Fig. 2 includes four network controller boards 110a-110d, two file controller boards 112a-112b, two storage processors 114a-114b, four system memory cards 116a-116d for a total of 192MB of memory, and one local host processor 118. The boards 110, 112, 114, 116 and 118 are connected together over a VME bus 120 on which an enhanced block transfer mode as described in the ENHANCED VMEBUS PROTOCOL application identified above may be used. Each of the four network controllers 110 shown in Fig. 2 can be connected to up to two Ethernets 122, for a total capacity of 8 Ethernets 122a-122h. Each of the storage processors 114 operates ten parallel SCSI busses, nine of which can each support up to three SCSI disk drives each. The tenth SCSI channel on each of the storage processors 114 is used for tape drives and other SCSI peripherals.

The host 118 is essentially a standard SunOs Unix processor, providing all the standard Sun Open Network Computing (ONC) services except NFS and IP routing. Importantly, all network requests to run a user-defined procedure are passed to the host for execution. Each of the NC boards 110, the FC boards 112 and the SP boards 114 includes its own independent 32-bit microprocessor. These boards essentially off-load from the host processor 118 virtually all of the NFS and disk processing. Since the vast majority of messages to and from clients over the Ethernets 122 involve NFS requests and responses, the processing of these requests in parallel by the NC, FC and SP processors, with minimal involvement by the local host 118, vastly improves file server performance. Unix is explicitly eliminated from virtually all network, file, and storage processing.

OVERALL SOFTWARE ORGANIZATION AND DATA FLOW

Prior to a detailed discussion of the hardware subsystems shown in Fig. 2, an overview of the software structure will now be undertaken. The software organization is described in more detail in the above-identified application entitled MULTIPLE FACILITY OPERATING SYSTEM ARCHITECTURE.

Most of the elements of the software are well known in the field and are found in most networked Unix systems, but there are two components which are not: Local NFS ("LNFS") and the messaging kernel ("MK") operating system kernel. These two components will be explained first.

The Messaging Kernel. The various processors in file server 100 communicate with each other through the use of a messaging kernel running on each of the processors 110, 112, 114 and 118. These processors do not share any instruction memory, so task-level communication cannot occur via straightforward procedure calls as it does in conventional Unix. Instead, the messaging kernel passes messages over VME bus 120 to accomplish all necessary inter-processor communication. Message passing is preferred over remote procedure calls for reasons of simplicity and speed.

Messages passed by the messaging kernel have a fixed 128-byte length. Within a single processor, messages are sent by reference; between processors, they are copied by the messaging kernel and then delivered to the destination process by reference. The processors of Fig. 2 have special hardware, discussed below, that can expediently exchange and buffer interprocessor messaging kernel messages.

The LNFS Local NFS interface. The 22-function NFS standard was specifically designed for stateless operation using unreliable communication. This means that neither clients nor server can be sure if they hear each other when they talk (unreliability). In practice, in an Ethernet environment, this works well.

Within the server 100, however, NFS level datagrams are also used for communication between processors, in particular between the network controllers 110 and the file controller 112, and between the host processor 118 and the file controller 112. For this internal communication to be both efficient and convenient, it is undesirable and impractical to have complete statelessness or unreliable communications. Consequently, a modified form of NFS, namely LNFS, is used for internal communication of NFS requests and responses. LNFS is used only within the file server 100; the external network protocol supported by the server is precisely standard, licensed NFS. LNFS is described in more detail below.

The Network Controllers 110 each run an NFS server which, after all protocol processing is done up to the

NFS layer, converts between external NFS requests and responses and internal LNFS requests and responses. For example, NFS requests arrive as RPC requests with XDR and enclosed in a UDP datagram. After protocol processing, the NFS server translates the NFS request into LNFS form and uses the messaging kernel to send the request to the file controller 112.

The file controller runs an LNFS server which handles LNFS requests both from network controllers and from the host 118. The LNFS server translates LNFS requests to a form appropriate for a file system server, also running on the file controller, which manages the system memory file data cache through a block I/O layer.

An overview of the software in each of the processors will now be set forth.

Network Controller 110

The optimized dataflow of the server 100 begins with the intelligent network controller 110. This processor receives Ethernet packets from client workstations. It quickly identifies NFS-destined packets and then performs full protocol processing on them to the NFS level, passing the resulting LNFS requests directly to the file controller 112. This protocol processing includes IP routing and reassembly, UDP demultiplexing, XDR decoding, and NFS request dispatching. The reverse steps are used to send an NFS reply back to a client. Importantly, these time-consuming activities are performed directly in the Network Controller 110, not in the host 118.

The server 100 uses conventional NFS ported from Sun Microsystems, Inc., Mountain View, CA, and is NFS protocol compatible.

Non-NFS network traffic is passed directly to its destination host processor 118.

The NCs 110 also perform their own IP routing. Each network controller 110 supports two fully parallel Ethernets. There are four network controllers in the embodiment of the server 100 shown in Fig. 2, so that server can support up to eight Ethernets. For the two Ethernets on the same network controller 110, IP routing occurs completely within the network controller and generates no backplane traffic. Thus attaching two mutually active Ethernets to the same controller not only minimizes their inter-net transit time, but also significantly reduces backplane contention on the VME bus 120. Routing table updates are distributed to the network controllers from the host processor 118, which runs either the gated or routed Unix demon.

While the network controller described here is designed for Ethernet LANs, it will be understood that the invention can be used just as readily with other network types, including FDDI.

File Controller 112

In addition to dedicating a separate processor for NFS protocol processing and IP routing, the server 100 also dedicates a separate processor, the intelligent file controller 112, to be responsible for all file system processing. It uses conventional Berkeley Unix 4.3 file system code and uses a binary-compatible data representation on disk. These two choices allow all standard file system utilities (particularly block-level tools) to run unchanged.

The file controller 112 runs the shared file system used by all NCs 110 and the host processor 118. Both the NCs and the host processor communicate with the file controller 112 using the LNFS interface. The NCs 110 use LNFS as described above, while the host processor 118 uses LNFS as a plug-in module to SunOs's standard Virtual File System ("VFS") interface.

When an NC receives an NFS read request from a client workstation, the resulting LNFS request passes to the FC 112. The FC 112 first searches the system memory 116 buffer cache for the requested data. If found, a reference to the buffer is returned to the NC 110. If not found, the LRU (least recently used) cache buffer in system memory 116 is freed and reassigned for the requested block. The FC then directs the SP 114 to read the block into the cache buffer from a disk drive array. When complete, the SP so notifies the FC, which in turn notifies the NC 110. The NC 110 then sends an NFS reply, with the data from the buffer, back to the NFS client workstation out on the network. Note that the SP 114 transfers the data into system memory 116, if necessary, and the NC 110 transfers the data from system memory 116 to the networks. The process takes place without any involvement of the host 118.

<u>Storage Processor</u>

The intelligent storage processor 114 manages all disk and tape storage operations. While autonomous, storage processors are primarily directed by the file controller 112 to move file data between system memory 116 and the disk subsystem. The exclusion of both the host 118 and the FC 112 from the actual data path helps to supply the performance needed to service many remote clients.

Additionally, coordinated by a Server Manager in the host 118, storage processor 114 can execute server backup by moving data between the disk subsystem and tape or other archival peripherals on the SCSI channels. Further, if directly accessed by host processor 118, SP 114 can provide a much higher performance conventional disk interface for Unix, virtual memory, and databases. In Unix nomenclature, the host processor 118 can mount boot, storage swap, and raw partitions via the storage processors 114.

Each storage processor 114 operates ten parallel, fully synchronous SCSI channels (busses) simultaneously. Nine of these channels support three arrays of nine SCSI disk drives each, each drive in an array being assigned to a different SCSI channel. The tenth SCSI channel hosts up to seven tape and other SCSI peripherals. In addition to performing reads and writes, SP 114 performs device-level optimizations such as disk seek queue sorting, directs device error recovery, and controls DMA transfers between the devices and system memory 116.

Host Processor 118

The local host 118 has three main purposes: to run Unix, to provide standard ONC network services for clients, and to run a Server Manager. Since Unix and ONC are ported from the standard SunOs Release 4 and ONC Services Release 2, the server 100 can provide identically compatible high-level ONC services such as the Yellow Pages, Lock Manager, DES Key Authenticator, Auto Mounter, and Port Mapper. Sun/2 Network disk booting and more general IP internet services such as Telnet, FTP, SMTP, SNMP, and reverse ARP are also supported. Finally, print spoolers and similar Unix demons operate transparently.

The host processor 118 runs the following software modules:

TCP and socket layers. The Transport Control Protocol ("TCP"), which is used for certain server functions other than NFS, provides reliable bytestream communication between two processors. Sockets are used to establish TCP connections.

VFS interface. The Virtual File System ("VFS") interface is a standard SunOs file system interface. It paints a uniform file-system picture for both users and the non-file parts of the Unix operating system, hiding the details of the specific file system. Thus standard NFS, LNFS, and any local Unix file system can coexist harmoniously.

UFS interface. The Unix File System ("UFS") interface is the traditional and well-known Unix interface for communication with local-to-the-processor disk drives. In the server 100, it is used to occasionally mount storage processor volumes directly, without going through the file controller 112.

Normally, the host 118 uses LNFS and goes through the file controller.

Device layer. The device layer is a standard software interface between the Unix device model and different physical device implementations. In the server 100, disk devices are not attached to host processors directly, so the disk driver in the host's device layer uses the messaging kernel to communicate with the storage processor 114.

Route and Port Mapper Demons. The Route and Port Mapper demons are Unix user-level background processes that maintain the Route and Port databases for packet routing. They are mostly inactive and not in any performance path.

Yellow Pages and Authentication Demon. The Yellow Pages and Authentication services are Sun-ONC standard network services. Yellow Pages is a widely used multipurpose name-to-name directory lookup service. The Authentication service uses cryptographic keys to authenticate, or validate, requests to insure that requestors have the proper privileges for any actions or data they desire.

Server Manager. The Server Manager is an administrative application suite that controls configuration, logs error and performance reports, and provides a monitoring and tuning interface for the system administrator. These functions can be exercised from either system console connected to the host 118, or from a system administrator's workstation.

The host processor 118 is a conventional OEM Sun central processor card, Model 3E/120. It incorporates a Motorola 68020 microprocessor and 4MB of on-board memory. Other processors, such as a SPARC-based processor, are also possible.

The structure and operation of each of the hardware components of server 100 will now be described in detail.

NETWORK CONTROLLER HARDWARE ARCHITECTURE

Fig. 3 is a block diagram showing the data path and some control paths for an illustrative one of the network controllers 110a. It comprises a 20 MHz 68020 microprocessor 210 connected to a 32-bit microprocessor data bus 212. Also connected to the microprocessor data bus 212 is a 256K byte CPU memory 214. The low order 8 bits of the microprocessor data bus 212 are connected through a bidirectional buffer 216 to an 8-bit slow-speed data bus 218. On the slow-speed data bus 218 is a 128K byte EPROM 220, a 32 byte PROM 222, and a multi-function peripheral (MFP) 224. The EPROM 220 contains boot code for the network controller 110a, while the PROM 222 stores various operating parameters such as the Ethernet addresses assigned to each of the two Ethernet interfaces on the board. Ethernet address information is read into the corresponding interface control block in the CPU memory 214 during initialization. The MFP 224 is a Motorola 68901, and performs various local functions such as timing, interrupts, and general purpose I/O. The MFP 224 also includes a UART for interfacing to an RS232 port 226. These functions are not critical to the invention and will not be further described herein.

The low order 16 bits of the microprocessor data bus 212 are also coupled through a bidirectional buffer 230 to a 16-bit LAN data bus 232. A LAN controller chip 234, such as the Am7990 LANCE Ethernet controller manufactured by Advanced Micro Devices, Inc. Sunnyvale, CA., interfaces the LAN data bus 232 with the first Ethernet 122a shown in Fig. 2. Control and data for the LAN controller 234 are stored in a 512K byte LAN memory 236, which is also connected to the LAN data bus 232. A specialized 16 to 32 bit FIFO chip 240, referred to herein as a parity FIFO chip and described below, is also connected to the LAN data bus 232. Also connected to the LAN data bus 232 is a LAN DMA controller 242, which controls movements of packets of data between the LAN memory 236 and the FIFO chip 240.

The LAN DMA controller 242 may be a Motorola M68440 DMA controller using channel zero only.

The second Ethernet 122b shown in Fig. 2 connects to a second LAN data bus 252 on the network controller card 110a shown in Fig. 3. The LAN data bus 252 connects to the low order 16 bits of the microprocessor data bus 212 via a bidirectional buffer 250, and has similar components to those appearing on the LAN data bus 232. In particular, a LAN controller 254 interfaces the LAN data bus 252 with the Ethernet 122b, using LAN memory 256 for data and control, and a LAN DMA controller 262 controls DMA transfer of data between the LAN memory 256 and the 16-bit wide data port A of the parity FIFO 260.

The low order 16 bits of microprocessor data bus 212 are also connected directly to another parity FIFO 270, and also to a control port of a VME/FIFO DMA controller 272. The FIFO 270 is used for passing messages between the CPU memory 214 and one of the remote boards 110, 112, 114, 116 or 118 (Fig. 2) in a manner described below. The VME/FIFO DMA controller 272, which supports three round-robin non-prioritized channels for copying data, controls all data transfers between one of the remote boards and any of the FIFOs 240, 260 or 270, as well as between the FIFOs 240 and 260.

32-bit data bus 274, which is connected to the 32-bit port B of each of the FIFOs 240, 260 and 270, is the data bus over which these transfers take place. Data bus 274 communicates with a local 32-bit bus 276 via a bidirectional pipelining latch 278, which is also controlled by VME/FIFO DMA controller 272 which in turn communicates with the VME bus 120 via a bidirectional buffer 280.

The local data bus 276 is also connected to a set of control registers 282, which are directly addressable across the VME bus 120. The registers 282 are used mostly for system initialization and diagnostics.

The local data bus 276 is also coupled to the microprocessor data bus 212 via a bidirectional buffer 284. When the NC 110a operates in slave mode, the CPU memory 214 is directly addressable from VME bus 120. One of the remote boards can copy data directly from the CPU memory 214 via the bidirectional buffer 284. LAN memories 236 and 256 are not directly addressed over VME bus 120.

The parity FIFOs 240, 260 and 270 each consist of an ASIC, the functions and operation of which are described in the Appendix C. The FIFOs 240 and 260 are configured for packet data transfer and the FIFO 270 is configured for message passing. Referring to the

Appendix C, the FIFOs 240 and 260 are programmed with the following bit settings in the Data Transfer Configuration Register:

Bit	Definition	Setting
0	WD Mode	N/A
1	Parity Chip	N/A
2	Parity Correct Mode	N/A
3	8/16 bits CPU & PortA interface	16 bits (1)
4	Invert Port A address 0	no (0)
5	Invert Port A address 1	yes (1)
6	Checksum Carry Wrap	yes (1)
7	Reset	no (0)

The Data Transfer Control Register is programmed as follows:

Bit	Definition	Setting
0	Enable PortA Req/Ack	yes (1)
1	Enable PortB Req/Ack	yes (1)
2	Data Transfer Direction	(as desired)
3	CPU parity enable	no (0)
4	PortA parity enable	no (0)
5	PortB parity enable	no (0)
6	Checksum Enable	yes (1)
7	PortA Master	yes (1)

Unlike the configuration used on FIFOs 240 and 260, the microprocessor 210 is responsible for loading and unloading Port A directly. The microprocessor 210 reads an entire 32-bit word from port A with a single instruction using two port A access cycles. Port A data transfer is disabled by unsetting bits 0 (Enable PortA Req/Ack) and 7 (PortA Master) of the Data Transfer Control Register.

The remainder of the control settings in FIFO 270 are the same as those in FIFOs 240 and 260 described above.

The NC 110a also includes a command FIFO 290. The command FIFO 290 includes an input port coupled to the local data bus 276, and which is directly addressable across the VME bus 120, and includes an output port connected to the microprocessor data bus 212. As explained in more detail below, when one of the remote boards issues a command or response to the NC 110a, it does so by directly writing a 1-word (32-bit) message descriptor into NC 110a's command FIFO 290. Command FIFO 290 generates a "FIFO not empty" status to the microprocessor 210, which then reads the message descriptor off the top of FIFO 290 and processes it. If the message is a command, then it includes a VME address at which the message is located (presumably an address in a shared memory similar to 214 on one of the remote boards). The microprocessor 210 then programs the FIFO 270 and the VME/FIFO DMA controller 272 to copy the message from the remote location into the CPU memory 214.

Command FIFO 290 is a conventional two-port FIFO, except that additional circuitry is included for generating a Bus Error signal on VME bus 120 if an attempt is made to write to the data input port while the FIFO is full. Command FIFO 290 has space for 256 entries.

A noteworthy feature of the architecture of NC 110a is that the LAN buses 232 and 252 are independent of the microprocessor data bus 212. Data packets being routed to or from an Ethernet are stored in LAN memory 236 on the LAN data bus 232 (or 256 on the LAN data bus 252), and not in the CPU memory 214. Data transfer between the LAN memories 236 and 256 and the Ethernets 122a and 122b, are controlled by LAN controllers 234 and 254, respectively, while most data transfer between LAN memory 236 or 256 and a remote port on the VME bus 120 are controlled by LAN DMA controllers 242 and 262, FIFOs 240 and 260, and VME/FIFO DMA controller 272. An exception to this rule occurs when the size of the data transfer is small, e.g., less than 64 bytes, in which case microprocessor 210 copies it directly without using DMA. The microprocessor 210 is not involved in larger transfers except in initiating them and in receiving notification when they are complete.

The CPU memory 214 contains mostly instructions for microprocessor 210, messages being transmitted to or from a remote board via FIFO 270, and various data blocks for controlling the FIFOs, the DMA controllers and the LAN controllers. The microprocessor 210 accesses the data packets in the LAN memories 236 and 256 by directly addressing them through the bidirectional buffers 230 and 250, respectively, for protocol processing. The local high-speed static RAM in CPU memory 214 can therefore provide zero wait state memory access for microprocessor 210 independent of network traffic. This is in sharp contrast to the prior art architecture shown in Fig. 1, in which all data and data packets, as well as microprocessor instructions for host CPU card 10, reside in the memory 16 and must communicate with the host CPU card 10 via the MMU 11.

While the LAN data buses 232 and 252 are shown as separate buses in Fig. 3, it will be understood that they may instead be implemented as a single combined bus.

NETWORK CONTROLLER OPERATION

In operation, when one of the LAN controllers (such as 234) receives a packet of information over its Ethernet 122a, it reads in the entire packet and stores it in corresponding LAN memory 236. The LAN controller 234 then issues an interrupt to microprocessor 210 via MFP 224, and the microprocessor 210 examines the status register on LAN controller 234 (via bidirectional buffer 230) to determine that the event causing the interrupt was a "receive packet completed." In order to avoid a potential lockout of the second Ethernet 122b caused by the prioritized interrupt handling characteristic of MFP 224, the microprocessor 210 does not at this time immediately process the received packet; instead, such processing is scheduled for a polling function.

When the polling function reaches the processing of the received packet, control over the packet is passed to a software link level receive module. The link level receive module then decodes the packet according to either of two different frame formats: standard Ethernet format or SNAP (IEEE 802 LCC) format. An entry in the header in the packet specifies which frame format was used. The link level driver then determines which of three types of messages is contained in the received packet: (1) IP, (2) ARP packets which can be handled by a local ARP module, or (3) ARP packets and other packet types which must be forwarded to the local host 118 (Fig. 2) for processing. If the packet is an ARP packet which can be handled by the NC 110a, such as a request for the address of server 100, then the microprocessor 210 assembles a response packet in LAN memory 236 and, in a conventional manner, causes LAN controller 234 to transmit that packet back over Ethernet 122a. It is noteworthy that the data manipulation for accomplishing this task is performed almost completely in LAN memory 236, directly addressed by microprocessor 210 as controlled by instructions in CPU memory 214. The function is accomplished also without generating any traffic on the VME backplane 120 at all, and without disturbing the local host 118.

If the received packet is either an ARP packet which cannot be processed completely in the NC 110a, or is another type of packet which requires delivery to the local host 118 (such as a client request for the server 100 to execute a client-defined procedure), then the microprocessor 210 programs LAN DMA controller 242 to load the packet from LAN memory 236 into FIFO 240, programs FIFO 240 with the direction of data transfer, and programs DMA controller 272 to read the packet out of FIFO 240 and across the VME bus 120 into system memory 116. In particular, the microprocessor 210 first programs the LAN DMA controller 242 with the starting address and length of the packet in LAN memory 236, and programs the controller to begin transferring data from the LAN memory 236 to port A of parity FIFO 240 as soon as the FIFO is ready to receive data. Second, microprocessor 210 programs the VME/FIFO DMA controller 272 with the destination address in system memory 116 and the length of the data packet, and instructs the controller to begin transferring data from port B of the FIFO 260 onto VME bus 120. Finally, the microprocessor 210 programs FIFO 240 with the direction of the transfer to take place. The transfer then proceeds entirely under the control of DMA controllers 242 and 262, without any further involvement by microprocessor 210.

The microprocessor 210 then sends a message to host 118 that a packet is available at a specified system memory address. The microprocessor 210 sends such a message by writing a message descriptor to a software-emulated command FIFO on the host, which copies the message from CPU memory 214 on the NC via buffer 284 and into the host's local memory, in ordinary VME block transfer mode. The host then copies the packet from system memory 116 into the host's own local memory using ordinary VME transfers.

If the packet received by NC 110a from the network is an IP packet, then the microprocessor 210 determines whether it is (1) an IP packet for the server 100 which is not an NFS packet; (2) an IP packet to be routed to a different network; or (3) an NFS packet. If it is an IP packet for the server 100, but not an NFS packet, then the microprocessor 210 causes the packet to be transmitted from the LAN memory 236 to the host 118 in the same manner described above with respect to certain ARP packets.

If the IP packet is not intended for the server 100, but rather is to be routed to a client on a different network, then the packet is copied into the LAN memory associated with the Ethernet to which the destination client is connected. If the destination client is on the Ethernet 122b, which is on the same NC board as the source Ethernet 122a, then the microprocessor 210 causes the packet to be copied from LAN memory 236 into LAN 256 and then causes LAN controller 254 to transmit it over Ethernet 122b. (Of course, if the two LAN data buses 232 and 252 are combined, then copying would be unnecessary; the microprocessor 210 would simply cause the LAN controller 254 to read the packet out of the same locations in LAN memory to which the packet was written by LAN controller 234.)

The copying of a packet from LAN memory 236 to LAN memory 256 takes place similarly to the copying described above from LAN memory to system memory. For transfer sizes of 64 bytes or more, the microprocessor

210 first programs the LAN DMA controller 242 with the starting address and length of the packet in LAN memory 236, and programs the controller to begin transferring data from the LAN memory 236 into port A of parity FIFO 240 as soon as the FIFO is ready to receive data. Second, microprocessor 210 programs the LAN DMA controller 262 with a destination address in LAN memory 256 and the length of the data packet, and instructs that controller to transfer data from parity FIFO 260 into the LAN memory 256. Third, microprocessor 210 programs the VME/FIFO DMA controller 272 to clock words of data out of port B of the FIFO 240, over the data bus 274, and into port B of FIFO 260. Finally, the microprocessor 210 programs the two FIFOs 240 and 260 with the direction of the transfer to take place. The transfer then proceeds entirely under the control of DMA controllers 242, 262 and 272, without any further involvement by the microprocessor 210.. Like the copying from LAN memory to system memory, if the transfer size is smaller than 64 bytes, the microprocessor 210 performs the transfer directly, without DMA.

When each of the LAN DMA controllers 242 and 262 complete their work, they so notify microprocessor 210 by a respective interrupt provided through MFP 224. When the microprocessor 210 has received both interrupts, it programs LAN controller 254 to transmit the packet on the Ethernet 122b in a conventional manner.

Thus, IP routing between the two Ethernets in a single network controller 110 takes place over data bus 274, generating no traffic over VME bus 120. Nor is the host processor 118 disturbed for such routing, in contrast to the prior art architecture of Fig. 1. Moreover, all but the shortest copying work is performed by controllers outside microprocessor 210, requiring the involvement of the microprocessor 210, and bus traffic on microprocessor data bus 212, only for the supervisory functions of programming the DMA controllers and the parity FIFOs and instructing them to begin. The VME/FIFO DMA controller 272 is programmed by loading control registers via microprocessor data bus 212; the LAN DMA controllers 242 and 262 are programmed by loading control registers on the respective controllers via the microprocessor data bus 212, respective bidirectional buffers 230 and 250, and respective LAN data buses 232 and 252, and the parity FIFOs 240 and 260 are programmed as set forth in the Appendix C.

If the destination workstation of the IP packet to be routed is on an Ethernet connected to a different one of the network controllers 110, then the packet is copied into the appropriate LAN memory on the NC 110 to which that Ethernet is connected. Such copying is accomplished by first copying the packet into system memory 116, in the manner described above with respect to certain ARP packets, and then notifying the destination NC that a packet is available. When an NC is so notified, it programs its own parity FIFO and DMA controllers to copy the packet from system memory 116 into the appropriate LAN memory. It is noteworthy that though this type of IP routing does create VME bus traffic, it still does not involve the host CPU 118.

If the IP packet received over the Ethernet 122a and now stored in LAN memory 236 is an NFS packet intended for the server 100, then the microprocessor 210 performs all necessary protocol preprocessing to extract the NFS message and convert it to the local NFS (LNFS) format. This may well involve the logical concatenation of data extracted from a large number of individual IP packets stored in LAN memory 236, resulting in a linked list, in CPU memory 214, pointing to the different blocks of data in LAN memory 236 in the correct sequence.

The exact details of the LNFS format are not important for an understanding of the invention, except to note that it includes commands to maintain a directory of files which are stored on the disks attached to the storage processors 114, commands for reading and writing data to and from a file on the disks, and various configuration management and diagnostics control messages. The directory maintenance commands which are supported by LNFS include the following messages based on conventional NFS: get attributes of a file (GETATTR); set attributes of a file (SETATTR); look up a file (LOOKUP); created a file (CREATE); remove a file (REMOVE); rename a file (RENAME); created a new linked file (LINK); create a symlink (SYMLINK); remove a directory (RMDIR); and return file system statistics (STATFS). The data transfer commands supported by LNFS include read from a file (READ); write to a file (WRITE); read from a directory (READDIR); and read a link (READLINK). LNFS also supports a buffer release command (RELEASE), for notifying the file controller that an NC is finished using a specified buffer in system memory. It also supports a VOP-derived access command, for determining whether a given type access is legal for specified credential on a specified file.

If the LNFS request includes the writing of file data from the LAN memory 236 to disk, the NC 110a first requests a buffer in system memory 116 to be allocated by the appropriate FC 112. When a pointer to the buffer is returned, microprocessor 210 programs LAN DMA controller 242, parity FIFO 240 and VME/FIFO DMA controller 272 to transmit the entire block of file data to system memory 116. The only difference between this transfer and the transfer described above for transmitting IP packets and ARP packets to system memory 116 is that these data blocks will typically have portions scattered throughout LAN memory 236. The microprocessor 210 accommodates that situation by programming LAN DMA controller 242 successively for each portion of the data, in accordance with the linked list, after receiving notification that the previous portion is complete. The microprocessor 210 can program the parity FIFO 240 and the VME/FIFO DMA controller 272 once for the entire message, as long as the entire data block is to be placed contiguously in system memory 116. If it is not, then the microprocessor 210 can program the DMA controller 272 for successive blocks in the same manner LAN DMA controller 242.

If the network controller 110a receives a message from another processor in server 100, usually from file controller 112, that file data is available in system memory 116 for transmission on one of the Ethernets, for example Ethernet 122a, then the network controller 110a copies the file data into LAN memory 236 in a manner similar to the copying of file data in the opposite direction. In particular, the microprocessor 210 first programs VME/FIFO DMA controller 272 with the starting address and length of the data in system memory 116, and programs the controller to begin transferring data over the VME bus 120 into port B of parity FIFO 240 as soon as the FIFO is ready to receive data. The microprocessor 210 then programs the LAN DMA controller 242 with a destination address in LAN memory 236 and then length of the file data, and instructs that controller to transfer data from the parity FIFO 240 into the LAN memory 236. Third, microprocessor 210 programs the parity FIFO 240 with the direction of the transfer to take place. The transfer then proceeds entirely under the control of DMA controllers 242 and 272, without any further involvement by the microprocessor 210. Again, if the file data is scattered in multiple blocks in system memory 116, the microprocessor 210 programs the VME/FIFO DMA controller 272 with a linked list of the blocks to transfer in the proper order.

When each of the DMA controllers 242 and 262 complete their work, they so notify microprocessor 210 through MFP 224. The microprocessor 210 then performs all necessary protocol processing on the LNFS message in LAN memory 236 in order to prepare the message for transmission over the Ethernet 122a in the form of

Ethernet IP packets. As set forth above, this protocol processing is performed entirely in network controller 110a, without any involvement of the local host 118.

It should be noted that the parity FIFOs are designed to move multiples of 128-byte blocks most efficiently. The data transfer size through port B is always 32-bits wide, and the VME address corresponding to the 32-bit data must be quad-byte aligned. The data transfer size for port A can be either 8 or 16 bits. For bus utilization reasons, it is set to 16 bits when the corresponding local start address is double-byte aligned, and is set at 8 bits otherwise. The TCP/IP checksum is always computed in the 16 bit mode. Therefore, the checksum word requires byte swapping if the local start address is not double-byte aligned.

Accordingly, for transfer from port B to port A of any of the FIFOs 240, 260 or 270, the microprocessor 210 programs the VME/FIFO DMA controller to pad the transfer count to the next 128-byte boundary. The extra 32-bit word transfers do not involve the VME bus, and only the desired number of 32-bit words will be unloaded from port A.

For transfers from port A to port B of the parity FIFO 270, the microprocessor 210 loads port A word-by-word and forces a FIFO full indication when it is finished. The FIFO full indication enables unloading from port B. The same procedure also takes place for transfers from port A to port B of either of the parity FIFOs 240 or 260, since transfers of fewer than 128 bytes are performed under local microprocessor control rather than under the control of LAN DMA controller 242 or 262. For all of the FIFOs, the VME/FIFO DMA controller is programmed to unload only the desired number of 32-bit words.

FILE CONTROLLER HARDWARE ARCHITECTURE

The file controllers (FC) 112 may each be a standard off-the-shelf microprocessor board, such as one manufactured by Motorola Inc. Preferably, however, a more specialized board is used such as that shown in block diagram form in Fig. 4.

Fig. 4 shows one of the FCs 112a, and it will be understood that the other FC can be identical. In many aspects it is simply a scaled-down version of the NC 110a shown in Fig. 3, and in some respects it is scaled up. Like the NC 110a, FC 112a comprises a 20MHz 68020 microprocessor 310 connected to a 32-bit microprocessor data bus 312. Also connected to the microprocessor data bus 312 is a 256K byte shared CPU memory 314. The low order 8 bits of the microprocessor data bus 312 are connected through a bidirectional buffer 316 to an 8-bit slow-speed data bus 318. On slow-speed data bus

318 are a 128K byte PROM 320, and a multifunction peripheral (MFP) 324. The functions of the PROM 320 and MFP 324 are the same as those described above with respect to EPROM 220 and MFP 224 on NC 110a. FC 112a does not include PROM like the PROM 222 on NC 110a, but does include a parallel port 392. The parallel port 392 is mainly for testing and diagnostics.

Like the NC 110a, the FC 112a is connected to the VME bus 120 via a bidirectional buffer 380 and a 32-bit local data bus 376. A set of control registers 382 are connected to the local data bus 376, and directly addressable across the VME bus 120. The local data bus 376 is also coupled to the microprocessor data bus 312 via a bidirectional buffer 384. This permits the direct addressability of CPU memory 314 from VME bus 120.

FC 112a also includes a command FIFO 390, which includes an input port coupled to the local data bus 376 and which is directly addressable across the VME bus 120. The command FIFO 390 also includes an output port connected to the microprocessor data bus 312. The structure, operation and purpose of command FIFO 390 are the same as those described above with respect to command FIFO 290 on NC 110a.

The FC 112a omits the LAN data buses 232 and 252 which are present in NC 110a, but instead includes a 4 megabyte 32-bit wide FC memory 396 coupled to the microprocessor data bus 312 via a bidirectional buffer 394. As will be seen, FC memory 396 is used as a cache memory for file control information, separate from the file data information cached in system memory 116.

The file controller embodiment shown in Fig. 4 does not include any DMA controllers, and hence cannot act as a master for transmitting or receiving data in any block transfer mode, over the VME bus 120. Block transfers do occur with the CPU memory 314 and the FC memory 396, however, with the FC 112a acting as an VME bus slave. In such transfers, the remote master addresses the CPU memory 314 or the FC memory 396 directly over the VME bus 120 through the bidirectional buffers 384 and, if appropriate, 394.

FILE CONTROLLER OPERATION

The purpose of the FC 112a is basically to provide virtual file system services in response to requests provided in LNFS format by remote processors on the VME bus 120. Most requests will come from a network controller 110, but requests may also come from the local host 118.

The file related commands supported by LNFS are identified above. They are all specified to the FC 112a in terms of logically identified disk data blocks.

For example, the LNFS command for reading data from a file includes a specification of the file from which to read (file system ID (FSID) and file ID (inode)), a byte offset, and a count of the number of bytes to read. The FC 112a converts that identification into physical form, namely disk and sector numbers, in order to satisfy the command.

The FC 112a runs a conventional Fast File System (FFS or UFS), which is based on the Berkeley 4.3 VAX release. This code performs the conversion and also performs all disk data caching and control data caching. However, as previously mentioned, control data caching is performed using the FC memory 396 on FC 112a, whereas disk data caching is performed using the system memory 116 (Fig. 2). Caching this file control information within the FC 112a avoids the VME bus congestion and speed degradation which would result if file control information was cached in system memory 116.

The memory on the FC 112a is directly accessed over the VME bus 120 for three main purposes. First, and by far the most frequent, are accesses to FC memory 396 by an SP 114 to read or write cached file control information. These are accesses requested by FC 112a to write locally modified file control structures through to disk, or to read file control structures from disk. Second, the FC's CPU memory 314 is accessed directly by other processors for message transmissions from the FC 112a to such other processors. For example, if a data block in system memory is to be transferred to an SP 114 for writing to disk, the FC 112a first assembles a message in its local memory 314 requesting such a transfer. The FC 112a then notifies the SP 114, which copies the message directly from the CPU memory 314 and executes the requested transfer.

A third type of direct access to the FC's local memory occurs when an LNFS client reads directory entries. When FC 112a receives an LNFS request to read directory entries, the FC 112a formats the requested directory entries in FC memory 396 and notifies the requestor of their location. The requestor then directly accesses FC memory 396 to read the entries.

The version of the UFS code on FC 112a includes some modifications in order to separate the two caches. In particular, two sets of buffer headers are maintained, one for the FC memory 396 and one for the system memory 116. Additionally, a second set of the system buffer routines (GETBLK(), BRELSE(), BREAD(), BWRITE(), and BREADA()) exist, one for buffer accesses to FC Mem 396 and one for buffer accesses to system memory 116. The UFS code is further modified to call the appropriate buffer routines for FC memory 396 for accesses to file control information, and to call the appropriate buffer routines for the system memory 116 for the caching of disk data. A description of UFS may be found in chapters 2, 6, 7 and 8 of "Kernel Structure and Flow," by Rieken and Webb of .sh consulting (Santa Clara, California: 1988), incorporated herein by reference.

When a read command is sent to the FC by a requestor such as a network controller, the FC first converts the file, offset and count information into disk and sector information. It then locks the system memory buffers which contain that information, instructing the storage processor 114 to read them from disk if necessary. When the buffer is ready, the FC returns a message to the requestor containing both the attributes of the designated file and an array of buffer descriptors that identify the locations in system memory 116 holding the data.

After the requestor has read the data out of the buffers, it sends a release request back to the FC. The release request is the same message that was returned by the FC in response to the read request; the FC 112a uses the information contained therein to determine which buffers to free.

A write command is processed by FC 112a similarly to the read command, but the caller is expected to write to (instead of read from) the locations in system memory 116 identified by the buffer descriptors returned by the FC 112a. Since FC 112a employs write-through caching, when it receives the release command from the requestor, it instructs storage processor 114 to copy the data from system memory 116 onto the appropriate disk sectors before freeing the system memory buffers for possible reallocation.

The READDIR transaction is similar to read and write, but the request is satisfied by the FC 112a directly out of its own FC memory 396 after formatting the requested directory information specifically for this purpose. The FC 112a causes the storage processor read the requested directory information from disk if it is not already locally cached. Also, the specified offset is a "magic cookie" instead of a byte offset, identifying directory entries instead of an absolute byte offset into the file. No file attributes are returned.

The READLINK transaction also returns no file attributes, and since links are always read in their entirety, it does not require any offset or count.

For all of the disk data caching performed through system memory 116, the FC 112a acts as a central authority for dynamically allocating, deallocating and keeping track of buffers. If there are two or more FCs 112, each has exclusive control over its own assigned portion of system memory 116. In all of these transactions, the requested buffers are locked during the period between the initial request and the release request. This prevents corruption of the data by other clients.

Also in the situation where there are two or more FCs, each file system on the disks is assigned to a particular one of the FCs. FC #0 runs a process called FC_VICE_PRESIDENT, which maintains a list of which file systems are assigned to which FC. When a client processor (for example an NC 110) is about to make an LNFS request designating a particular file system, it first sends the fsid in a message to the FC_VICE_PRESIDENT asking which FC controls the specified file system. The FC_VICE_PRESIDENT responds, and the client processor sends the LNFS request to the designated FC. The client processor also maintains its own list of fsid/FC pairs as it discovers them, so as to minimize the number of such requests to the FC_VICE_PRESIDENT.

STORAGE PROCESSOR HARDWARE ARCHITECTURE

In the file server 100, each of the storage processors 114 can interface the VME bus 120 with up to 10 different SCSI buses. Additionally, it can do so at the full usage rate of an enhanced block transfer protocol of 55MB per second.

Fig. 5 is a block diagram of one of the SPs 114a. SP 114b is identical. SP 114a comprises a microprocessor 510, which may be a Motorola 68020 microprocessor operating at 20MHz. The microprocessor 510 is coupled over a 32-bit microprocessor data bus 512 with CPU memory 514, which may include up to 1MB of static RAM. The microprocessor 510 accesses instructions, data and status on its own private bus 512, with no contention from any other source. The microprocessor 510 is the only master of bus 512.

The low order 16 bits of the microprocessor data bus 512 interface with a control bus 516 via a bidirectional buffer 518. The low order 8 bits of the control bus 516 interface with a slow speed bus 520 via another bidirectional buffer 522. The slow speed bus 520 connects to an MFP 524, similar to the MFP 224 in NC 110a (Fig. 3), and with a PROM 526, similar to PROM 220 on NC 110a. The PROM 526 comprises 128K bytes of EPROM which contains the functional code for SP 114a. Due to the width and speed of the PROM 526, the functional code is copied to CPU memory 514 upon reset for faster execution.

MFP 524, like the MFP 224 on NC 110a, comprises a Motorola 68901 multifunction peripheral device. It provides the functions of a vectored interrupt controller, individually programmable I/O pins, four timers and a UART. The UART functions provide serial communications across an RS 232 bus (not shown in Fig. 5) for debug monitors and diagnostics. Two of the four timing functions may be used as general-purpose timers by the microprocessor 510, either independently or in cascaded fashion. A third timer function provides the refresh clock for a DMA controller described below, and the fourth timer generates the UART clock. Additional information on the MFP 524 can be found in "MC 68901 Multi-Function Peripheral Specification," by Motorola, Inc., which is incorporated herein by reference.

The eight general-purpose I/O bits provided by MFP 524 are configured according to the following table:

Bit	Direction	Definition
7	input	Power Failure is Imminent - This functions as an early warning.
6	input	SCSI Attention - A composite of the SCSI. Attentions from all 10 SCSI channels.
5	input	Channel Operation Done - A composite of the channel done bits from all 13 channels of the DMA controller, described below.
4	output	DMA Controller Enable. Enables the DMA Controller to run.
3	input	VMEbus Interrupt Done - Indicates the completion of a VMEbus Interrupt.
2	input	Command Available - Indicates that the SP'S Command Fifo, described below, contains one or more command pointers.
1	output	External Interrupts Disable. Disables externally generated interrupts to the microprocessor 510.
0	output	Command Fifo Enable. Enables operation of the SP'S Command Fifo. Clears the Command Fifo when reset.

Commands are provided to the SP 114a from the VME bus 120 via a bidirectional buffer 530, a local data bus 532, and a command FIFO 534. The command FIFO 534 is similar to the command FIFOs 290 and 390 on NC 110a and FC 112a, respectively, and has a depth of 256 32-bit entries. The command FIFO 534 is a write-only register as seen on the VME bus 120, and as a read-only register as seen by microprocessor 510. If the FIFO is full at the beginning of a write from the VME bus, a VME bus error is generated. Pointers are removed from the command FIFO 534 in the order received, and only by the microprocessor 510. Command available status is provided through I/O bit 4 of the MFP 524, and as a long as one or more command pointers are still within the command FIFO 534, the command available status remains asserted.

As previously mentioned, the SP 114a supports up to 10 SCSI buses or channels 540a-540j. In the typical configuration, buses 540a-540i support up to 3 SCSI disk drives each, and channel 540j supports other SCSI peripherals such as tape drives, optical disks, and so on. Physically, the SP 114a connects to each of the SCSI buses with an ultra-miniature D sub connector and round shielded cables. Six 50-pin cables provide 300 conductors which carry 18 signals per bus and 12 grounds. The cables attach at the front panel of the SP 114a and to a commutator board at the disk drive array. Standard 50-pin cables connect each SCSI device to the commutator board. Termination resistors are installed on the SP 114a.

The SP 114a supports synchronous parallel data transfers up to 5MB per second on each of the SCSI buses 540, arbitration, and disconnect/reconnect services. Each SCSI bus 540 is connected to a respective SCSI adaptor 542, which in the present embodiment is an AIC 6250 controller IC manufactured by

Adaptec Inc., Milpitas, California, operating in the non-multiplexed address bus mode. The AIC 6250 is described in detail in "AIC-6250 Functional Specification," by Adaptec Inc., which is incorporated herein by reference. The SCSI adaptors 542 each provide the necessary hardware interface and low-level electrical protocol to implement its respective SCSI channel.

The 8-bit data port of each of the SCSI adaptors 542 is connected to port A of a respective one of a set of ten parity FIFOs 544a-544j. The FIFOs 544 are the same as FIFOs 240, 260 and 270 on NC 110a, and are connected and configured to provide parity covered data transfers between the 8-bit data port of the respective SCSI adaptors 542 and a 36-bit (32-bit plus 4 bits of parity) common data bus 550. The FIFOs 544 provide handshake, status, word assembly/disassembly and speed matching FIFO buffering for this purpose. The FIFOs 544 also generate and check parity for the 32-bit bus, and for RAID 5 implementations they accumulate and check redundant data and accumulate recovered data.

All of the SCSI adaptors 542 reside at a single location of the address space of the microprocessor 510, as do all of the parity FIFOs 544. The microprocessor 510 selects individual controllers and FIFOs for access in pairs, by first programming a pair select register (not shown) to point to the desired pair and then reading from or writing to the control register address of the desired chip in the pair. The microprocessor 510 communicates with the control registers on the SCSI adaptors 542 via the control bus 516 and an additional bidirectional buffer 546, and communicates with the control registers on FIFOs 544 via the control bus 516 and a bidirectional buffer 552. Both the SCSI adaptors 542 and FIFOs 544 employ 8-bit control registers, and register addressing of the FIFOs 544 is arranged such that such registers alias in consecutive byte locations. This allows the microprocessor 510 to write to the registers as a single 32-bit register, thereby reducing instruction overhead.

The parity FIFOs 544 are each configured in their Adaptec 6250 mode. Referring to the Appendix C, the FIFOs 544 are programmed with the following bit settings in the Data Transfer Configuration Register:

Bit	Definition	Setting
0	WD Mode	(0)
1	Parity Chip	(1)
2	Parity Correct Mode	(0)
3	8/16 bits CPU & PortA interface	(0)
4	Invert Port A address 0	(1)
5	Invert Port A address 1	(1)

6	Checksum Carry Wrap	(0)
7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

Bit	Definition	Setting
0	Enable PortA Req/Ack	(1)
1	Enable PortB Req/Ack	(1)
2	Data Transfer Direction	as desired
3	CPU parity enable	(0)
4	PortA parity enable	(1)
5	PortB parity enable	(1)
6	Checksum Enable	(0)
7	PortA Master	(0)

In addition, bit 4 of the RAM Access Control Register (Long Burst) is programmed for 8-byte bursts.

SCSI adaptors 542 each generate a respective interrupt signal, the status of which are provided to microprocessor 510 as 10 bits of a 16-bit SCSI interrupt register 556. The SCSI interrupt register 556 is connected to the control bus 516. Additionally, a composite SCSI interrupt is provided through the MFP 524 whenever any one of the SCSI adaptors 542 needs servicing.

An additional parity FIFO 554 is also provided in the SP 114a, for message passing. Again referring to the Appendix C, the parity FIFO 554 is programmed with the following bit settings in the Data Transfer Configuration Register:

Bit	Definition	Setting
0	WD Mode	(0)
1	Parity Chip	(1)
2	Parity Correct Mode	(0)
3	8/16 bits CPU & PortA interface	(1)
4	Invert Port A address 0	(1)
5	Invert Port A address 1	(1)
6	Checksum Carry Wrap	(0)
7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

Bit	Definition	Setting
0	Enable PortA Req/Ack	(0)
1	Enable PortB Req/Ack	(1)
2	Data Transfer Direction	as desired
3	CPU parity enable	(0)
4	PortA parity enable	(0)
5	PortB parity enable	(1)
6	Checksum Enable	(0)
7	PortA Master	(0)

In addition, bit 4 of the RAM Access Control Register (Long Burst) is programmed for 8-byte bursts.

Port A of FIFO 554 is connected to the 16-bit control bus 516, and port B is connected to the common data bus 550. FIFO 554 provides one means by which the microprocessor 510 can communicate directly with the VME bus 120, as is described in more detail below.

The microprocessor 510 manages data movement using a set of 15 channels, each of which has an unique status which indicates its current state. Channels are implemented using a channel enable register 560 and a channel status register 562, both connected to the control bus 516. The channel enable register 560 is a 16-bit write-only register, whereas the channel status register 562 is a 16-bit read-only register. The two registers reside at the same address to microprocessor 510. The microprocessor 510 enables a particular channel by setting its respective bit in channel enable register 560, and recognizes completion of the specified operation by testing for a "done" bit in the channel status register 562. The microprocessor 510 then resets the enable bit, which causes the respective "done" bit in the channel status register 562 to be cleared.

The channels are defined as follows:

CHANNEL	FUNCTION
0:9	These channels control data movement to and from the respective FIFOs 544 via the common data bus 550. When a FIFO is enabled and a request is received from it, the channel becomes ready. Once the channel has been serviced a status of done is generated.
11:10	These channels control data movement between a local data buffer 564, described below, and the VME bus 120. When enabled the channel becomes ready. Once the channel has been serviced a status of done is generated.
12	When enabled, this channel causes the DRAM in local data buffer 564 to be refreshed based on a clock which is generated by the MFP 524. The refresh consists of a burst of 16 rows. This channel does not generate a status of done.
13	The microprocessor's communication FIFO 554 is serviced by this channel. When enable is set and the FIFO 554 asserts a request then the channel becomes ready. This channel generates a status of done.
14	Low latency writes from microprocessor 510 onto the VME bus 120 are controlled by this channel. When this channel is enabled data is moved from a special 32 bit register, described below, onto the VME bus 120. This channel generates a done status.
15	This is a null channel for which neither a ready status nor done status is generated.

Channels are prioritized to allow servicing of the more critical requests first. Channel priority is assigned in a descending order starting at channel 14. That is, in the event that all channels are requesting service, channel 14 will be the first one served.

The common data bus 550 is coupled via a bidirectional register 570 to a 36-bit junction bus 572. A second bidirectional register 574 connects the junction bus 572 with the local data bus 532. Local data buffer 564, which comprises 1MB of DRAM, with parity, is coupled bidirectionally to the junction bus 572. It is organized to provide 256K 32-bit words with byte parity. The SP 114a operates the DRAMs in page mode to support a very high data rate, which requires bursting of data instead of random single-word accesses. It will be seen that the local data buffer 564 is used to implement a RAID (redundant array of inexpensive disks) algorithm, and is not used for direct reading and writing between the VME bus 120 and a peripheral on one of the SCSI buses 540.

A read-only register 576, containing all zeros, is also connected to the junction bus 572. This register is used mostly for diagnostics, initialization, and clearing of large blocks of data in system memory 116.

The movement of data between the FIFOs 544 and 554, the local data buffer 564, and a remote entity such as the system memory 116 on the VME bus 120, is all controlled by a VME/FIFO DMA controller 580. The VME/FIFO DMA controller 580 is similar to the VME/FIFO DMA controller 272 on network controller 110a (Fig. 3), and is described in the Appendix A. Briefly, it includes a bit slice engine 582 and a dual-port static RAM 584. One port of the dual-port static RAM 584 communicates over the 32-bit microprocessor data bus 512 with microprocessor 510, and the other port communicates over a separate 16-bit bus with the bit slice engine 582. The microprocessor 510 places command parameters in the dual-port RAM 584, and uses the channel enables 560 to signal the VME/FIFO DMA controller 580 to proceed with the command. The VME/FIFO DMA controller is responsible for scanning the channel status and servicing requests, and returning ending status in the dual-port RAM 584. The dual-port RAM 584 is organized as 1K x 32 bits at the 32-bit port and as 2K x 16 bits at the 16-bit port. An example showing the method by which the microprocessor 510 controls the VME/FIFO DMA controller 580 is as follows. First, the microprocessor 510 writes into the dual-port RAM 584 the desired command and associated parameters for the desired channel. For example, the command might be, "copy a block of data from FIFO 544h out into a block of system memory 116 beginning at a specified VME address." Second, the microprocessor sets the channel enable bit in channel enable register 560 for the desired channel.

At the time the channel enable bit is set, the appropriate FIFO may not yet be ready to send data.

Only when the VME/FIFO DMA controller 580 does receive a "ready" status from the channel, will the controller 580 execute the command. In the meantime, the DMA controller 580 is free to execute commands and move data to or from other channels.

When the DMA controller 580 does receive a status of "ready" from the specified channel, the controller fetches the channel command and parameters from the dual-ported RAM 584 and executes. When the command is complete, for example all the requested data has been copied, the DMA controller writes status back into the dual-port RAM 584 and asserts "done" for the channel in channel status register 562. The microprocessor 510 is then interrupted, at which time it reads channel status register 562 to determine which channel interrupted. The microprocessor 510 then clears the channel enable for the appropriate channel and checks the ending channel status in the dual-port RAM 584.

In this way a high-speed data transfer can take place under the control of DMA controller 580, fully in parallel with other activities being performed by microprocessor 510. The data transfer takes place over busses different from microprocessor data bus 512, thereby avoiding any interference with microprocessor instruction fetches.

The SP 114a also includes a high-speed register 590, which is coupled between the microprocessor data bus 512 and the local data bus 532. The high-speed register 590 is used to write a single 32-bit word to an VME bus target with a minimum of overhead. The register is write only as viewed from the microprocessor 510. In order to write a word onto the VME bus 120, the microprocessor 510 first writes the word into the register 590, and the desired VME target address into dual-port RAM 584. When the microprocessor 510 enables the appropriate channel in channel enable register 560, the DMA controller 580 transfers the data from the register 590 into the VME bus address specified in the dual-port RAM 584. The DMA controller 580 then writes the ending status to the dual-port RAM and sets the channel "done" bit in channel status register 562.

This procedure is very efficient for transfer of a single word of data, but becomes inefficient for large blocks of data. Transfers of greater than one word of data, typically for message passing, are usually performed using the FIFO 554.

The SP 114a also includes a series of registers 592, similar to the registers 282 on NC 110a (Fig. 3) and the registers 382 on FC 112a (Fig. 4). The details of these registers are not important for an understanding of the present invention.

STORAGE PROCESSOR OPERATION

The 30 SCSI disk drives supported by each of the SPs 114 are visible to a client processor, for example one of the file controllers 112, either as three large, logical disks or as 30 independent SCSI drives, depending on configuration. When the drives are visible as three logical disks, the SP uses RAID 5 design algorithms to distribute data for each logical drive on nine physical drives to minimize disk arm contention. The tenth drive is left as a spare. The RAID 5 algorithm (redundant array of inexpensive drives, revision 5) is described in "A Case For a Redundant Arrays of Inexpensive Disks (RAID)", by Patterson et al., published at ACM SIGMOD Conference, Chicago, Ill., June 1-3, 1988, incorporated herein by reference.

In the RAID 5 design, disk data are divided into stripes. Data stripes are recorded sequentially on eight different disk drives. A ninth parity stripe, the exclusive-or of eight data stripes, is recorded on a ninth drive. If a stripe size is set to 8K bytes, a read of 8K of data involves only one drive. A write of 8K of data involves two drives: a data drive and a parity drive. Since a write requires the reading back of old data to generate a new parity stripe, writes are also referred to as modify writes. The SP 114a supports nine small reads to nine SCSI drives concurrently. When stripe size is set to 8K, a read of 64K of data starts all eight SCSI drives, with each drive reading one 8K stripe worth of data. The parallel operation is transparent to the caller client.

The parity stripes are rotated among the nine drives in order to avoid drive contention during write operations. The parity stripe is used to improve availability of data. When one drive is down, the SP 114a can reconstruct the missing data from a parity stripe. In such case, the SP 114a is running in error recovery mode. When a bad drive is repaired, the SP 114a can be instructed to restore data on the repaired drive while the system is on-line.

When the SP 114a is used to attach thirty independent SCSI drives, no parity stripe is created and the client addresses each drive directly.

The SP 114a processes multiple messages (transactions, commands) at one time, up to 200 messages per second. The SP 114a does not initiate any messages after initial system configuration. The following SP 114a operations are defined:

01 No Op

02 Send Configuration Data

03 Receive Configuration Data

05 Read and Write Sectors

06 Read and Write Cache Pages

07 IOCTL Operation

08 Dump SP 114a Local Data Buffer

09 Start/Stop A SCSI Drive

0C Inquiry

0E Read Message Log Buffer

0F Set SP 114a Interrupt

The above transactions are described in detail in the above-identified application entitled MULTIPLE FACILITY OPERATING SYSTEM ARCHITECTURE. For an understanding of the invention, it will be useful to describe the function and operation of only two of these commands: read and write sectors, and read and write cache pages.

Read and Write Sectors

This command, issued usually by an FC 112, causes the SP 114a to transfer data between a specified block of system memory and a specified series of contiguous sectors on the SCSI disks. As previously described in connection with the file controller 112, the particular sectors are identified in physical terms. In particular, the particular disk sectors are identified by SCSI channel number (0-9), SCSI ID on that channel number (0-2), starting sector address on the specified drive, and a count of the number of sectors to read or write. The SCSI channel number is zero if the SP 114a is operating under RAID 5.

The SP 114a can execute up to 30 messages on the 30 SCSI drives simultaneously. Unlike most of the commands to an SP 114, which are processed by microprocessor 510 as soon as they appear on the command FIFO 534, read and write sectors commands (as well as read and write cache memory commands) are first sorted and queued. Hence, they are not served in the order of arrival.

When a disk access command arrives, the microprocessor 510 determines which disk drive is targeted and inserts the message in a queue for that disk drive sorted by the target sector address. The microprocessor 510 executes commands on all the queues simultaneously, in the order present in the queue for each disk drive. In order to minimize disk arm movements, the microprocessor 510 moves back and forth among queue entries in an elevator fashion.

If no error conditions are detected from the SCSI disk drives, the command is completed normally. When a data check error condition occurs and the SP 114a is configured for RAID 5, recovery actions using redundant data begin automatically. When a drive is down while the SP 114a is configured for RAID 5, recovery actions similar to data check recovery take place.

Read/Write Cache Pages

This command is similar to read and write sectors, except that multiple VME addresses are provided for transferring disk data to and from system memory 116. Each VME address points to a cache page in system memory 116, the size of which is also specified in the command. When transferring data from a disk to system memory 116, data are scattered to different cache pages; when writing data to a disk, data are gathered from different cache pages in system memory 116. Hence, this operation is referred to as a scatter-gather function.

The target sectors on the SCSI disks are specified in the command in physical terms, in the same manner that they are specified for the read and write sectors command. Termination of the command with or without error conditions is the same as for the read and write sectors command.

The dual-port RAM 584 in the DMA controller 580 maintains a separate set of commands for each channel controlled by the bit slice engine 582. As each channel completes its previous operation, the microprocessor 510 writes a new DMA operation into the dual-port RAM 584 for that channel in order to satisfy the next operation on a disk elevator queue.

The commands written to the DMA controller 580 include an operation code and a code indicating whether the operation is to be performed in non-block mode, in standard VME block mode, or in enhanced block mode. The operation codes supported by DMA controller 580 are as follows:

OP CODE	OPERATION	
0	NO-OP	
1	ZEROES -> BUFFER	Move zeros from zeros register 576 to local data buffer 564.
2	ZEROES -> FIFO	Move zeros from zeros register 576 to the currently selected FIFO on common data bus 550.
3	ZEROES -> VMEbus	Move zeros from zeros register 576 out onto the VME bus 120. Used for initializing cache buffers in system memory 116.

4	VMEbus -> BUFFER	Move data from the VME bus 120 to the local data buffer 564. This operation is used during a write, to move target data intended for a down drive into the buffer for participation in redundancy generation. Used only for RAID 5 application.
5	VMEbus -> FIFO	New data to be written from VME bus onto a drive. Since RAID 5 requires redundancy data to be generated from data that is buffered in local data buffer 564, this operation will be used only if the SP 114a is not configured for RAID 5.
6	VMEbus -> BUFFER & FIFO	Target data is moved from VME bus 120 to a SCSI device and is also captured in the local data buffer 564 for participation in redundancy generation. Used only if SP 114a is configured for RAID 5 operation.
7	BUFFER -> VMEbus	This operation is not used.
8	BUFFER -> FIFO	Participating data is transferred to create redundant data or recovered data on a disk drive. Used only in RAID 5 applications.
9	FIFO -> VMEbus	This operation is used to move target data directly from a disk drive onto the VME bus 120.

A	FIFO	-> BUFFER	Used to move participating data for recovery and modify operations. Used only in RAID 5 applications.
B	FIFO	-> VMEbus & BUFFER	This operation is used to save target data for participation in data recovery. Used only in RAID 5 applications.

SYSTEM MEMORY

Fig. 6 provides a simplified block diagram of the preferred architecture of one of the system memory cards 116a. Each of the other system memory cards are the same. Each memory card 116 operates as a slave on the enhanced VME bus 120 and therefore requires no on-board CPU. Rather, a timing control block 610 is sufficient to provide the necessary slave control operations. In particular, the timing control block 610, in response to control signals from the control portion of the enhanced VME bus 120, enables a 32-bit wide buffer 612 for an appropriate direction transfer of 32-bit data between the enhanced VME bus 120 and a multiplexer unit 614. The multiplexer 614 provides a multiplexing and demultiplexing function, depending on data transfer direction, for a six megabit by seventy-two bit word memory array 620. An error correction code (ECC) generation and testing unit 622 is also connected to the multiplexer 614 to generate or verify, again depending on transfer direction, eight bits of ECC data. The status of ECC verification is provided back to the timing control block 610.

ENHANCED VME BUS PROTOCOL

VME bus 120 is physically the same as an ordinary VME bus, but each of the NCs and SPs include additional circuitry and firmware for transmitting data using an enhanced VME block transfer protocol. The enhanced protocol is described in detail in the above-identified application entitled ENHANCED VMEBUS PROTOCOL UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA TRANSFER, and summarized in the Appendix B hereto. Typically transfers of LNFS file data between NCs and system memory, or between SPs and system memory, and transfers of packets being routed from one NC to another through system memory, are the only types of transfers that use the enhanced protocol in server 100. All other data transfers on VME bus 120 use either conventional VME block transfer protocols or ordinary non-block transfer protocols.

MESSAGE PASSING

As is evident from the above description, the different processors in the server 100 communicate with each other via certain types of messages. In software, these messages are all handled by the messaging kernel, described in detail in the MULTIPLE FACILITY OPERATING SYSTEM ARCHITECTURE application cited above. In hardware, they are implemented as follows.

Each of the NCs 110, each of the FCs 112, and each of the SPs 114 includes a command or communication FIFO such as 290 on NC 110a. The host 118 also includes a command FIFO, but since the host is an unmodified purchased processor board, the FIFO is emulated in software. The write port of the command FIFO in each of the processors is directly addressable from any of the other processors over VME bus 120.

Similarly, each of the processors except SPs 114 also includes shared memory such as CPU memory 214 on NC 110a. This shared memory is also directly addressable by any of the other processors in the server 100.

If one processor, for example network controller 110a, is to send a message or command to a second processor, for example file controller 112a, then it does so as follows. First, it forms the message in its own shared memory (e.g., in CPU memory 214 on NC 110a). Second, the microprocessor in the sending processor directly writes a message descriptor into the command FIFO in the receiving processor. For a command being sent from network controller 110a to file controller

112a, the microprocessor 210 would perform the write via buffer 284 on NC 110a, VME bus 120, and buffer 384 on file controller 112a.

The command descriptor is a single 32-bit word containing in its high order 30 bits a VME address indicating the start of a quad-aligned message in the sender's shared memory. The low order two bits indicate the message type as follows:

Type	Description
0	Pointer to a new message being sent
1	Pointer to a reply message
2	Pointer to message to be forwarded
3	Pointer to message to be freed; also message acknowledgment

All messages are 128-bytes long.

When the receiving processor reaches the command descriptor on its command FIFO, it directly accesses the sender's shared memory and copies it into the receiver's own local memory. For a command issued from network controller 110a to file controller 112a, this would be an ordinary VME block or non-block mode transfer from NC CPU memory 214, via buffer 284, VME bus 120 and buffer 384, into FC CPU memory 314. The FC microprocessor 310 directly accesses NC CPU memory 214 for this purpose over the VME bus 120.

When the receiving processor has received the command and has completed its work, it sends a reply message back to the sending processor. The reply message may be no more than the original command message unaltered, or it may be a modified version of that message or a completely new message. If the reply message is not identical to the original command message, then the receiving processor directly accesses the original sender's shared memory to modify the original command message or overwrite it completely. For replies from the FC 112a to the NC 110a, this involves an ordinary VME block or non-block mode transfer from the FC 112a, via buffer 384, VME bus 120, buffer 284 and into NC CPU memory 214. Again, the FC microprocessor 310 directly accesses NC CPU memory 214 for this purpose over the VME bus 120.

Whether or not the original command message has been changed, the receiving processor then writes a reply message descriptor directly into the original sender's command FIFO. The reply message descriptor contains the same VME address as the original command message descriptor, and the low order two bits of the word are modified to indicate that this is a reply message. For replies from the FC 112a to the NC 110a, the message descriptor write is accomplished by microprocessor 310 directly accessing command FIFO 290 via buffer 384, VME bus 120 and buffer 280 on the NC. Once this is done, the receiving processor can free the buffer in its local memory containing the copy of the command message.

When the original sending processor reaches the reply message descriptor on its command FIFO, it wakes up the process that originally sent the message and permits it to continue. After examining the reply message, the original sending processor can free the original command message buffer in its own local shared memory.

As mentioned above, network controller 110a uses the buffer 284 data path in order to write message descriptors onto the VME bus 120, and uses VME/FIFO DMA controller 272 together with parity FIFO 270 in order to copy messages from the VME bus 120 into CPU memory 214. Other processors read from CPU memory 214 using the buffer 284 data path.

File controller 112a writes message descriptors onto the VME bus 120 using the buffer 384 data path, and copies messages from other processors' shared memory via the same data path. Both take place under the control of microprocessor 310. Other processors copy messages from CPU memory 314 also via the buffer 384 data path.

Storage processor 114a writes message descriptors onto the VME bus using high-speed register 590 in the manner described above, and copies messages from other processors using DMA controller 580 and FIFO 554. The SP 114a has no shared memory, however, so it uses a buffer in system memory 116 to emulate that function. That is, before it writes a message descriptor into another processor's command FIFO, the SP 114a first copies the message into its own previously allocated buffer in system memory 116 using DMA controller 580 and FIFO 554. The VME address included in the message descriptor then reflects the VME address of the message in system memory 116.

In the host 118, the command FIFO and shared memory are both emulated in software.

The invention has been described with respect to particular embodiments thereof, and it will be understood that numerous modifications and variations are possible within the scope of the invention.

APPENDIX A

VME/FIFO DMA Controller

In storage processor 114a, DMA controller 580 manages the data path under the direction of the microprocessor 510. The DMA controller 580 is a microcoded 16-bit bit-slice implementation executing pipelined instructions at a rate of one each 62.5ns. It is responsible for scanning the channel status 562 and servicing request with parameters stored in the dual-ported ram 584 by the microprocessor 510. Ending status is returned in the ram 584 and interrupts are generated for the microprocessor 510.

Control Store. The control store contains the microcoded instructions which control the DMA controller 580. The control store consists of 6 1K x 8 proms configured to yield a 1K x 48 bit microword. Locations within the control store are addressed by the sequencer and data is presented at the input of the pipeline registers.

Sequencer. The sequencer controls program flow by generating control store addresses based upon pipeline data and various status bits. The control store address consists of 10 bits. Bits 8:0 of the control store address derive from a multiplexer having as its inputs either an ALU output or the output of an incrementer. The incrementer can be preloaded with pipeline register bits 8:0, or it can be incremented as a result of a test condition. The 1K address range is divided into two pages by a latched flag such that the microprogram can execute from either page. Branches, however remain within the selected page. Conditional sequencing is performed by having the test condition increment the pipeline provided address. A false condition allows execution from the pipeline address while a true condition causes execution from the address + 1. The alu output is selected as an address source in order to directly vector to a routine or in order to return to a calling routine. Note that when calling a subroutine the calling routine must reside within the same page as the subroutine or the wrong page will be selected on the return.

ALU. The alu comprises a single IDT49C402A integrated circuit. It is 16 bits in width and most closely resembles four 2901s with 64 registers. The alu is used primarily for incrementing, decrementing, addition and bit manipulation. All necessary control signals originate in the control store. The IDT HIGH PERFORMANCE CMOS 1988 DATA BOOK, incorporated by reference herein, contains additional information about the alu.

Microword. The 48 bit microword comprises several fields which control various functions of the

DMA controller 580. The format of the microword is defined below along with mnemonics and a description of each function.

AI<8:0> 47:39 (Alu Instruction bits 8:0) The AI bits provide the instruction for the 49C402A alu. Refer to the IDT data book for a complete definition of the alu instructions. Note that the I9 signal input of the 49C402A is always low.

CIN 38 (Carry INput) This bit forces the carry input to the alu.

RA<5:0> 37:32 (Register A address bits 5:0) These bits select one of 64 registers as the "A" operand for the alu. These bits also provide literal bits 15:10 for the alu bus.

RB<5:0> 31:26 (Register B address bits 5:0) These bits select one of 64 registers as the "B" operand for the alu. These bits also provide literal bits 9:4 for the alu bus.

LFD 25 (Latched Flag Data) When set this bit causes the selected latched flag to be set. When reset this bit causes the selected latched flag to be cleared. This bits also functions as literal bit 3 for the alu bus.

LFS<2:0> 24:22 (Latched Flag Select bits 2:0) The meaning of these bits is dependent upon the selected source for the alu bus. In the event that the literal field is selected as the bus source then LFS<2:0> function as literal bits <2:0> otherwise the bits are used to select one of the latched flags.

LFS<2:0>	SELECTED FLAG
0	This value selects a null flag.
1	When set this bit enables the buffer clock. When reset this bit disables the buffer clock.
2	When this bit is cleared VME bus transfers, buffer operations and RAS are all disabled.
3	NOT USED
4	When set this bit enables VME bus transfers.
5	When set this bit enables buffer operations.
6	When set this bit asserts the row address strobe to the dram buffer.
7	When set this bit selects page 0 of the control store.

SRC<1,0> 20,21 (alu bus SouRCe select bits 1,0) These bits select the data source to be enabled onto the alu bus.

SRC<1,0>	Selected Source
0	alu
1	dual ported ram
2	literal
3	reserved-not defined

PF<2:0> 19:17 (Pulsed Flag select bits 2:0) These bits select a flag/signal to be pulsed.

PF<2:0>	Flag
0	null
1	SGL_CLK generates a single transition of buffer clock.
2	SET_VB forces vme and buffer enable to be set.
3	CL_PERR clears buffer parity error status.
4	SET_DN set channel done status for the currently selected channel.
5	INC_ADR increment dual ported ram address.
6:7	RESERVED - NOT DEFINED

DEST<3:0> 16:13 (DESTination select bits 3:0) These bits select one of 10 destinations to be loaded from the alu bus.

DEST<3:0>	Destination
0	null
1	WR_RAM causes the data on the alu bus to be written to the dual ported ram. D<15:0> -> ram<15:0>
2	WR_BADD loads the data from the alu bus into the dram address counters. D<14:7> -> mux addr<8:0>

3 WR_VADL
loads the data from the alu bus into the least significant 2 bytes of the VME address register.
D<15:2> -> VME addr<15:2>
D1 -> ENB_ENH
D0 -> ENB_BLK

4 WR_VADH
loads the most significant 2 bytes of the VME address register.
D<15:0> -> VME addr<31:16>

5 WR_RADD
loads the dual ported ram address counters.
D<10:0> -> ram addr <10:0>

6 WR_WCNT
loads the word counters.
D15 -> count enable*
D<14:8> -> count <6:0>

7 WR_CO
loads the co-channel select register.
D<7:4> -> CO<3:0>

8 WR_NXT
loads the next-channel select register.
D<3:0> -> NEXT<3:0>

9 WR_CUR
loads the current-channel select register.
D<3:0> -> CURR <3:0>

10:14 RESERVED - NOT DEFINED

15 JUMP
causes the control store sequencer to select the alu data bus.
D<8:0> -> CS_A<8:0>

TEST<3:0> 12:9 (TEST condition select bits 3:0) Select one of 16 inputs to the test multiplexor to be used as the carry input to the incrementer.

TEST<3:0>	Condition	
0	FALSE	-always false
1	TRUE	-always true
2	ALU_COUT	-carry output of alu
3	ALU_EQ	-equals output of alu
4	ALU_OVR	-alu overflow
5	ALU_NEG	-alu negative
6	XFR_DONE	-transfer complete
7	PAR_ERR	-buffer parity error
8	TIMOUT	-bus operation timeout
9	ANY_ERR	-any error status
14:10	RESERVED	-NOT DEFINED
15	CH_RDY	-next channel ready

NEXT_A<8:0> 8:0 (NEXT Address bits 8:0)- Selects an instructions from the current page of the control store for execution.

Dual Ported Ram. The dual ported ram is the medium by which command, parameters and status are communicated between the DMA controller 580 and the microprocessor 510. The ram is organized as 1K x 32 at the master port and as 2K x 16 at the DMA port. The ram may be both written and read at either port.

The ram is addressed by the DMA controller 580 by loading an 11 bit address into the address counters.

Data is then read into bidirectional registers and the address counter is incremented to allow read of the next location.

Writing the ram is accomplished by loading data from the processor into the registers after loading the ram address. Successive writes may be performed on every other processor cycle.

The ram contains current block pointers, ending status, high speed bus address and parameter blocks. The following is the format of the ram:

```
OFFSET  31                              0
        --------------------------------
  0     ¦CURR POINTER 0 ¦   STATUS 0    ¦
        --------------------------------
  4     ¦       INITIAL POINTER 0       ¦
        --------------------------------
                       .
                       .
        --------------------------------
 58     ¦CURR POINTER B ¦   STATUS B    ¦
        --------------------------------
 5C     ¦       INITIAL POINTER B       ¦
        --------------------------------
 60     ¦   not used    ¦   not used    ¦
        --------------------------------
 64     ¦   not used    ¦   not used    ¦
        --------------------------------
 68     ¦CURR POINTER D ¦   STATUS D    ¦
        --------------------------------
 6C     ¦       INITIAL POINTER D       ¦
        --------------------------------
 70     ¦   not used    ¦   STATUS E    ¦
        --------------------------------
 74     ¦HIGH SPEED BUS ADDRESS 31:2¦0¦0¦
        --------------------------------
 78     ¦      PARAMETER BLOCK 0        ¦
        --------------------------------
                       .
                       .
        --------------------------------
 ??     ¦      PARAMETER BLOCK n        ¦
        --------------------------------
```

The Initial Pointer is a 32 bit value which points the first command block of a chain. The current pointer is a sixteen bit value used by the DMA controller 580 to point to the current command block. The current command block pointer should be initialized to 0x0000 by the microprocessor 510 before enabling the channel. Upon detecting a value of 0x0000 in the current block pointer the DMA controller 580 will copy the lower 16 bits from the initial pointer to the current pointer. Once the DMA controller 580 has completed the specified operations for the parameter block the current pointer will be updated to point to the next block. In the event that no further parameter blocks are available the pointer will be set to 0x0000.

The status byte indicates the ending status for the last channel operation performed. The following status bytes are defined:

STATUS	MEANING
0	NO ERRORS
1	ILLEGAL OP CODE
2	BUS OPERATION TIMEOUT
3	BUS OPERATION ERROR
4	DATA PATH PARITY ERROR

The format of the parameter block is:

OFFSET	31		0
0	FORWARD LINK		
4	NOT USED		WORD COUNT
8	VME ADDRESS 31:2, ENH, BLK		
C	TERM 0	OP 0	BUF ADDR 0
	. . . .		
C+(4Xn)	TERM n	OP n	BUF ADDR n

FORWARD LINK - The forward link points to the first word of the next parameter block for execution. It allows several parameter blocks to be initialized and chained to create a sequence of operations for execution. The forward pointer has the following format:

A31:A2,0,0

The format dictates that the parameter block must start on a quad byte boundary. A pointer of 0x00000000 is a special case which indicates no forward link exists.

WORD COUNT - The word count specifies the number of quad byte words that are to be transferred to or from each buffer address or to/from the VME address. A word count of 64K words may be specified by initializing the word count with the value of 0. The word count has the following format:

¦D15¦D14¦D13¦D12¦D11¦D10¦D9¦D8¦D7¦D6¦D5¦D4¦D3¦D2¦D1¦D0¦

The word count is updated by the DMA controller 580 at the completion of a transfer to/from the last specified buffer address. Word count is not updated after transferring to/from each buffer address and is therefore not an accurate indicator of the total data moved to/from the buffer. Word count represents the amount of data transferred to the VME bus or one of the FIFOs 544 or 554.

VME ADDRESS - The VME address specifies the starting address for data transfers. Thirty bits allows the address to start at any quad byte boundary.

ENH - This bit when set selects the enhanced block transfer protocol described in the above-cited ENHANCED VMEBUS PROTOCOL UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA TRANSFER application, to be used during the VME bus transfer. Enhanced protocol will be disabled automatically when performing any transfer to or from 24 bit or 16 bit address space, when the starting address is not 8 byte aligned or when the word count is not even.

BLK - This bit when set selects the conventional VME block mode protocol to be used during the VME bus transfer. Block mode will be disabled automatically when performing any transfer to or from 16 bit address space.

BUF ADDR - The buffer address specifies the starting buffer address for the adjacent operation. Only 16 bits are available for a 1M byte buffer and as a result the starting address always falls on a 16 byte boundary. The programmer must ensure that the starting address is on a modulo 128 byte boundary. The buffer address is updated by the DMA controller 580 after completion of each data burst.

¦A19¦A18¦A17¦A16¦A15¦A14¦A13¦A12¦A11¦A10¦A9¦A8¦A7¦A6¦A5¦A4¦

TERM - The last buffer address and operation within a parameter block is identified by the terminal bit. The DMA controller 580 continues to fetch buffer addresses and operations to perform until this bit is encountered. Once the last operation within the parameter block is executed the word counter is updated and if not equal to zero the series of operations is repeated. Once the word counter reaches zero the forward link pointer is used to access the next parameter block.

|0|0|0|0|0|0|0|0|T|

OP - Operations are specified by the op code. The op code byte has the following format:

|0|0|0|0|OP3|OP2|OP1|OP0|

The op codes are listed below ("FIFO" refers to any of the FIFOs 544 or 554):

OP CODE	OPERATION
0	NO-OP
1	ZEROES -> BUFFER
2	ZEROES -> FIFO
3	ZEROES -> VMEbus
4	VMEbus -> BUFFER
5	VMEbus -> FIFO
6	VMEbus -> BUFFER & FIFO
7	BUFFER -> VMEbus
8	BUFFER -> FIFO
9	FIFO -> VMEbus
A	FIFO -> BUFFER
B	FIFO -> VMEbus & BUFFER
C	RESERVED
D	RESERVED
E	RESERVED
F	RESERVED

APPENDIX B

Enhanced VME Block Transfer Protocol

The enhanced VME block transfer protocol is a VMEbus compatible pseudo-synchronous fast transfer handshake protocol for use on a VME backplane bus having a master functional module and a slave functional module logically interconnected by a data transfer bus. The data transfer bus includes a data strobe signal line and a data transfer acknowledge signal line. To accomplish the handshake, the master transmits a data strobe signal of a given duration on the data strobe line. The master then awaits the reception of a data transfer acknowledge signal from the slave module on the data transfer acknowledge signal line. The slave then responds by transmitting data transfer acknowledge signal of a given duration on the data transfer acknowledge signal line.

Consistent with the pseudo-synchronous nature of the handshake protocol, the data to be transferred is referenced to only one signal depending upon whether the transfer operation is a READ or WRITE operation. In transferring data from the master functional unit to the slave, the master broadcasts the data to be transferred. The master asserts a data strobe signal and the slave, in response to the data strobe signal, captures the data broadcast by the master. Similarly, in transferring data from the slave to the master, the slave broadcasts the data to be transferred to the master unit. The slave then asserts a data transfer acknowledge signal and the master, in response to the data transfer acknowledge signal, captures the data broadcast by the slave.

The fast transfer protocol, while not essential to the present invention, facilitates the rapid transfer of large amounts of data across a VME backplane bus by substantially increasing the data transfer rate. These data rates are achieved by using a handshake wherein the data strobe and data transfer acknowledge signals are functionally decoupled and by specifying high current drivers for all data and control lines.

The enhanced pseudo-synchronous method of data transfer (hereinafter referred to as "fast transfer mode") is implemented so as to comply and be compatible with the IEEE VME backplane bus standard. The protocol utilizes user-defined address modifiers, defined in the VMEbus standard, to indicate use of the fast transfer mode. Conventional VMEbus functional units, capable only of implementing standard VMEbus protocols, will ignore transfers made using the fast transfer mode and, as a result, are fully compatible with functional units capable of implementing the fast transfer mode.

The fast transfer mode reduces the number of bus propagations required to accomplish a handshake from four propagations, as required under conventional VMEbus protocols, to only two bus propagations. Likewise, the number of bus propagations required to effect a BLOCK READ or BLOCK WRITE data transfer is reduced. Consequently, by reducing the propagations across the VMEbus to accomplish handshaking and data transfer functions, the transfer rate is materially increased.

The enhanced protocol is described in detail in the above-cited ENHANCED VMEBUS PROTOCOL application, and will only be summarized here. Familiarity with the conventional VME bus standards is assumed.

In the fast transfer mode handshake protocol, only two bus propagations are used to accomplish a handshake, rather than four as required by the conventional protocol. At the initiation of a data transfer cycle, the master will assert and deassert DSO* in the form of a pulse of a given duration. The deassertion of DSO* is accomplished without regard as to whether a response has been received from the slave. The master then waits for an acknowledgement from the slave. Subsequent pulsing of DSO* cannot occur until a responsive DTACK* signal is received from the slave. Upon receiving the slave's assertion of DTACK*, the master can then immediately reassert data strobe, if so desired. The fast transfer mode protocol does not require the master to wait for the deassertion of DTACK* by the slave as a condition precedent to subsequent assertions of DS0*. In the fast transfer mode, only the leading edge (i.e., the assertion) of a signal is significant. Thus, the deassertion of either DS0* or DTACK*_ is completely irrelevant for completion of a handshake. The fast transfer protocol does not employ the DS1* line for data strobe purposes at all.

The fast transfer mode protocol may be characterized as pseudo-synchronous as it includes both synchronous and asynchronous aspects. The fast transfer mode protocol is synchronous in character due to the fact that DS0* is asserted and deasserted without regard to a response from the slave. The asynchronous aspect of the fast transfer mode protocol is attributable to the fact that the master may not subsequently assert DS0* until a response to the prior strobe is received from the slave. Consequently, because the protocol includes both synchronous and asynchronous components, it is most accurately classified as "pseudo-synchronous."

The transfer of data during a BLOCK WRITE cycle in the fast transfer protocol is referenced only to DS0*. The master first broadcasts valid data to the slave, and then asserts DS0 to the slave. The slave is given a predetermined period of time after the assertion of DS0* in which to capture the data. Hence, slave modules must be prepared to capture data at any time, as DTACK* is not referenced during the transfer cycle.

Similarly, the transfer of data during a BLOCK READ cycle in the fast transfer protocol is referenced only to DTACK*. The master first asserts DS0*. The slave then broadcasts data to the master and then asserts DTACK*. The master is given a predetermined period of time after the assertion of DTACK in which to capture the data. Hence, master modules must be prepared to capture data at any time as DS0 is not referenced during the transfer cycle.

Fig. 7, parts A through C, is a flowchart illustrating the operations involved in accomplishing the fast transfer protocol BLOCK WRITE cycle. To initiate a BLOCK WRITE cycle, the master broadcasts the memory address of the data to be transferred and the address modifier across the DTB bus. The master also drives interrupt acknowledge signal (IACK*) high and the LWORD* signal low 701. A special address modifier, for example "1F," broadcast by the master indicates to the slave module that the fast transfer protocol will be used to accomplish the BLOCK WRITE.

The starting memory address of the data to be transferred should reside on a 64-bit boundary and the size of block of data to be transferred should be a multiple of 64 bits. In order to remain in compliance with the VMEbus standard, the block must not cross a 256 byte boundary without performing a new address cycle.

The slave modules connected to the DTB receive the address and the address modifier broadcast by the master across the bus and receive LWORD* low and IACK* high 703. Shortly after broadcasting the address and address modifier 701, the master drives the AS* signal low 705. The slave modules receive the AS* low signal 707. Each slave individually determines whether it will participate in the data transfer by determining whether the broadcasted address is valid for the slave in question 709. If the address is not valid, the data transfer does not involve that particular slave and it ignores the remainder of the data transfer cycle.

The master drives WRITE* low to indicate that the transfer cycle about to occur is a WRITE operation 711. The slave receives the WRITE* low signal 713 and, knowing that the data transfer operation is a WRITE operation, awaits receipt of a high to low transition on the DS0* signal line 715. The master will wait until both DTACK* and BERR* are high 718, which indicates that the previous slave is no longer driving the DTB.

The master proceeds to place the first segment of the data to be transferred on data lines D00 through D31, 719. After placing data on D00 through D31, the master drives DS0* low 721 and, after a predetermined interval, drives DS0* high 723.

In response to the transition of DS0* from high to low, respectively 721 and 723, the slave latches the data being transmitted by the master over data lines D00 through D31, 725. The master places the next segment of the data to be transferred on data lines D00 through D31, 727, and awaits receipt of a DTACK* signal in the form of a high to low transition signal, 729 in Fig. 7B.

Referring to Fig. 7B, the slave then drives DTACK* low, 731, and, after a predetermined period of time, drives DTACK high, 733. The data latched by the slave, 725, is written to a device, which has been selected to store the data 735. The slave also increments the device address 735. The slave then waits for another transition of DS0* from high to low 737.

To commence the transfer of the next segment of the block of data to be transferred, the master drives DS0* low 739 and, after a predetermined period of time, drives DS0* high 741. In response to the transition of DSO* from high to low, respectively 739 and 741, the slave latches the data being broadcast by the master over data lines D00 through D31, 743. The master places the next segment of the data to be transferred on data lines D00 through D31, 745, and awaits receipt of a DTACK* signal in the form of a high to low transition, 747.

The slave then drives DTACK* low, 749, and, after a predetermined period of time, drives DTACK* high, 751. The data latched by the slave, 743, is written to the device selected to store the data and the device address is incremented 753. The slave waits for another transition of DSO* from high to low 737.

The transfer of data will continue in the above-described manner until all of the data has been transferred from the master to the slave. After all of the data has been transferred, the master will release the address lines, address modifier lines, data lines, IACK* line, LWORD* line and DSO* line, 755. The master will then wait for receipt of a DTACK* high to low transition 757. The slave will drive DTACK* low, 759 and, after a predetermined period of time, drive DTACK* high 761. In response to the receipt of the DTACK* high to low transition, the master will drive AS* high 763 and then release the AS* line 765.

Fig. 8, parts A through C, is a flowchart illustrating the operations involved in accomplishing the fast transfer protocol BLOCK READ cycle. To initiate a BLOCK READ cycle, the master broadcasts the memory address of the data to be transferred and the address modifier across the DTB bus 801. The master drives the LWORD* signal low and the IACK* signal high 801. As noted previously, a special address modifier indicates to the slave module that the fast transfer protocol will be used to accomplish the BLOCK READ.

The slave modules connected to the DTB receive the address and the address modifier broadcast by the master across the bus and receive LWORD* low and IACK* high 803. Shortly after broadcasting the address and address modifier 801, the master drives the AS* signal low 805. The slave modules receive the AS* low signal 807. Each slave individually determines whether it will participate in the data transfer by determining whether the broadcasted address is valid for the slave in question 809. If the address is not valid, the data transfer does not involve that particular slave and it ignores the remainder of the data transfer cycle.

The master drives WRITE* high to indicate that the transfer cycle about to occur is a READ operation 811. The slave receives the WRITE* high signal 813 and, knowing that the data transfer operation is a READ operation, places the first segment of the data to be transferred on data lines D00 through D31 819. The master will wait until both DTACK* and BERR* are high 818, which indicates that the previous slave is no longer driving the DTB.

The master then drives DS0* low 821 and, after a predetermined interval, drives DS0* high 823. The master then awaits a high to low transition on the DTACK* signal line 824. As shown in Fig. 8B, the slave then drives the DTACK* signal low 825 and, after a predetermined period of time, drives the DTACK* signal high 827.

In response to the transition of DTACK* from high to low, respectively 825 and 827, the master latches the data being transmitted by the slave over data lines D00 through D31, 831. The data latched by the master, 831, is written to a device, which has been selected to store the data the device address is incremented 833.

The slave places the next segment of the data to be transferred on data lines D00 through D31, 829, and then waits for another transition of DS0* from high to low 835.

To commence the transfer of the next segment of the block of data to be transferred, the master drives DS0* low 839 and, after a predetermined period of time, drives DSO* high 841. The master then waits for the DTACK* line to transition from high to low, 843.

The slave drives DTACK* low, 845, and, after a predetermined period of time, drives DTACK* high, 847. In response to the transition of DTACK* from high to low, respectively 839 and 841, the master latches the data being transmitted by the slave over data lines D00 through D31, 845. The data latched by the master, 845, is written to the device selected to store the data, 851 in Fig. 8C, and the device address is incremented. The slave places the next segment of the data to be transferred on data lines D00 through D31, 849.

The transfer of data will continue in the above-described manner until all of the data to be transferred from the slave to the master has been written into the device selected to store the data. After all of the data to be transferred has been written into the storage device, the master will release the address lines, address modifier lines, data lines, the IACK* line, the LWORD line and DSO* line 852. The master will then wait for receipt of a DTACK* high to low transition 853. The slave will drive DTACK* low 855 and, after a predetermined period of time, drive DTACK* high 857. In response to the receipt of the DTACK* high to low transition, the master will drive AS* high 859 and release the AS* line 861.

To implement the fast transfer protocol, a conventional 64 mA tri-state driver is substituted for the 48 mA open collector driver conventionally used in VME slave modules to drive DTACK*. Similarly, the conventional VMEbus data drivers are replaced with 64 mA tri-state drivers in SO-type packages. The latter modification reduces the ground lead inductance of the actual driver package itself and, thus, reduces "ground bounce" effects which contribute to skew between data, DSO* and DTACK*. In addition, signal return inductance along the bus backplane is reduced by using a connector system having a greater number of ground pins so as to minimize signal return and mated-pair pin inductance. One such connector system is the "High Density Plus" connector, Model No. 420-8015-000, manufactured by Teradyne Corporation.

APPENDIX C

Parity FIFO

The parity FIFOs 240, 260 and 270 (on the network controllers 110), and 544 and 554 (on storage processors 114) are each implemented as an ASIC. All the parity FIFOs are identical, and are configured on power-up or during normal operation for the particular function desired. The parity FIFO is designed to allow speed matching between buses of different speed, and to perform the parity generation and correction for the parallel SCSI drives.

The FIFO comprises two bidirectional data ports, Port A and Port B, with 36 x 64 bits of RAM buffer between them. Port A is 8 bits wide and Port B is 32 bits wide. The RAM buffer is divided into two parts, each 36 x 32 bits, designated RAM X and RAM Y. The two ports access different halves of the buffer alternating to the other half when available. When the chip is configured as a parallel parity chip (e.g. one of the FIFOs 544 on SP 114a), all accesses on Port B are monitored and parity is accumulated in RAM X and RAM Y alternately.

The chip also has a CPU interface, which may be 8 or 16 bits wide. In 16 bit mode the Port A pins are used as the most significant data bits of the CPU interface and are only actually used when reading or writing to the Fifo Data Register inside the chip.

A REQ, ACK handshake is used for data transfer on both Ports A and B. The chip may be configured as either a master or a slave on Port A in the sense that, in master mode the Port A ACK / RDY output signifies that the chip is ready to transfer data on Port A, and the Port A REQ input specifies that the slave is responding. In slave mode, however, the Port A REQ input specifies that the master requires a data transfer, and the chip responds with Port A ACK / RDY when data is available. The chip is a master on Port B since it raises Port B REQ and waits for Port B ACK to indicate completion of the data transfer.

SIGNAL DESCRIPTIONS

Port A 0-7, P

Port A is the 8 bit data port. Port A P, if used, is the odd parity bit for this port.

A Req, A Ack/Rdy

These two signals are used in the data transfer mode to control the handshake of data on Port A.

uP Data 0-7, uP Data P, uPAdd 0-2, CS

These signals are used by a microprocessor to address the programmable registers within the chip. The odd parity signal uP Data P is only checked when data is written to the Fifo Data or Checksum Registers and microprocessor parity is enabled.

Clk

The clock input is used to generate some of the chip timing. It is expected to be in the 10-20 Mhz range.

Read En, Write En

During microprocessor accesses, while CS is true, these signals determine the direction of the microprocessor accesses. During data transfers in the WD mode these signals are data strobes used in conjunction with Port A Ack.

Port B 00-07, 10-17, 20-27, 30-37, 0P-3P

Port B is a 32 bit data port. There is one odd parity bit for each byte. Port B 0P is the parity of bits 00-07, PortB 1P is the parity of bits 10-17, Port B 2P is the parity of bits 20-27, and Port B 3P is the parity of bits 30-37.

B Select, B Req, B Ack, Parity Sync, B Output Enable

These signals are used in the data transfer mode to control the handshake of data on Port B. Port B Req and Port B Ack are both gated with Port B Select. The Port B Ack signal is used to strobe the data on the Port B data lines. The parity sync signal is used to indicate to a chip configured as the parity chip to indicate that the last words of data involved in the parity accumulation are on Port B. The Port B data lines will only be driven by the Fifo chip if all of the following conditions are met:

a. the data transfer is from Port A to Port B;

b. the Port B select signal is true;

c. the Port B output enable signal is true; and

d. the chip is not configured as the parity chip or it is in parity correct mode and the Parity Sync signal is true.

Reset

This signal resets all the registers within the chip and causes all bidirectional pins to be in a high impedance state.

DESCRIPTION OF OPERATION

Normal Operation. Normally the chip acts as a simple FIFO chip. A FIFO is simulated by using two RAM buffers in a simple ping-pong mode. It is intended, but not mandatory, that data is burst into or out of the FIFO on Port B. This is done by holding Port B Sel signal low and pulsing the Port B Ack signal. When transferring data from Port B to Port A, data is first written into RAM X and when this is full, the data paths will be switched such that Port B may start writing to RAM Y. Meanwhile the chip will begin emptying RAM X to Port A. When RAM Y is full and RAM X empty the data paths will be switched again such that Port B may reload RAM X and Port A may empty RAM Y.

Port A Slave Mode. This is the default mode and the chip is reset to this condition. In this mode the chip waits for a master such as one of the SCSI adapter chips 542 to raise Port A Request for data transfer. If data is available the Fifo chip will respond with Port A Ack/Rdy.

Port A WD Mode. The chip may be configured to run in the WD or Western Digital mode. In this mode the chip must be configured as a slave on Port A. It differs from the default slave mode in that the chip responds with Read Enable or Write Enable as appropriate together with Port A Ack/Rdy. This mode is intended to allow the chip to be interfaced to the Western Digital 33C93A SCSI chip or the NCR 53C90 SCSI chip.

Port A Master Mode. When the chip is configured as a master, it will raise Port A Ack/Rdy when it is ready for data transfer. This signal is expected to be tied to the Request input of a DMA controller which will respond with Port A Req when data is available. In order to allow the DMA controller to burst, the Port A Ack/Rdy signal will only be negated after every 8 or 16 bytes transferred.

Port B Parallel Write Mode. In parallel write mode, the chip is configured to be the parity chip for a parallel transfer from Port B to Port A. In this mode, when Port B Select and Port B Request are asserted, data is written into RAM X or RAM Y each time the Port B Ack signal is received. For the first block of 128 bytes data is simply copied into the selected RAM. The next 128 bytes driven on Port B will be exclusive-ORed with the first 128 bytes. This procedure will be repeated for all drives such that the parity is accumulated in this chip. The Parity Sync signal should be asserted to the parallel chip together with the last block of 128 bytes. This enables the chip to switch access to the other RAM and start accumulating a new 128 bytes of parity.

Port B Parallel Read Mode - Check Data. This mode is set if all drives are being read and parity is to be checked. In this case the Parity Correct bit in the Data Transfer Configuration Register is not set. The parity chip will first read 128 bytes on Port A as in a normal read mode and then raise Port B Request. While it has this signal asserted the chip will monitor the Port B Ack signals and exclusive-or the data on Port B with the data in its selected RAM. The Parity Sync should again be asserted with the last block of 128 bytes. In this mode the chip will not drive the Port B data lines but will check the output of its exclusive-or logic for zero. If any bits are set at this time a parallel parity error will be flagged.

Port B Parallel Read Mode - Correct Data. This mode is set by setting the Parity Correct bit in the Data Transfer Configuration Register. In this case the chip will work exactly as in the check mode except that when Port B Output Enable, Port B Select and Parity Sync are true the data is driven onto the Port B data lines and a parallel parity check for zero is not performed.

Byte Swap. In the normal mode it is expected that Port B bits 00-07 are the first byte, bits 10-17 the second byte, bits 20-27 the third byte, and bits 30-37 the last byte of each word. The order of these bytes may be changed by writing to the byte swap bits in the configuration register such that the byte address bits are inverted. The way the bytes are written and read also depend on whether the CPU interface is configured as 16 or 8 bits. The following table shows the byte alignments for the different possibilities for data transfer using the Port A Request / Acknowledge handshake:

CPU I/F	Invert Addr 1	Invert Addr 0	Port B 00-07	Port B 10-17	Port B 20-27	Port B 30-37
8	False	False	Port A byte 0	Port A byte 1	Port A byte 2	Port A byte 1
8	False	True	Port A byte 1	Port A byte 0	Port A byte 3	Port A byte 2
8	True	False	Port A byte 2	Port A byte 3	Port A byte 0	Port A byte 1
8	True	True	Port A byte 3	Port A byte 2	Port A byte 1	Port A byte 0
16	False	False	Port A byte 0	uProc byte 0	Port A byte 1	uProc byte 1
16	False	True	uProc byte 0	Port A byte 0	uProc byte 1	Port A byte 1
16	True	False	Port A byte 1	uProc byte 1	Port A byte 0	uProc byte 0
16	True	True	uProc byte 1	Port A byte 1	uProc byte 0	Port A byte 0

When the Fifo is accessed by reading or writing the Fifo Data Register through the microprocessor port in 8 bit mode, the bytes are in the same order as the table above but the uProc data port is used instead of Port A. In 16 bit mode the table above applies.

Odd Length Transfers. If the data transfer is not a multiple of 32 words, or 128 bytes, the microprocessor must manipulate the internal registers of the chip to ensure all data is transferred. Port A Ack and Port B Req are normally not asserted until all

32 words of the selected RAM are available. These signals may be forced by writing to the appropriate RAM status bits of the Data Transfer Status Register.

When an odd length transfer has taken place the microprocessor must wait until both ports are quiescent before manipulating any registers. It should then reset both of the Enable Data Transfer bits for Port A and Port B in the Data Transfer Control Register. It must then determine by reading their Address Registers and the RAM Access Control Register whether RAM X or RAM Y holds the odd length data. It should then set the corresponding Address Register to a value of 20 hexadecimal, forcing the RAM full bit and setting the address to the first word. Finally the microprocessor should set the Enable Data Transfer bits to allow the chip to complete the transfer.

At this point the Fifo chip will think that there are now a full 128 bytes of data in the RAM and will transfer 128 bytes if allowed to do so. The fact that some of these 128 bytes are not valid must be recognized externally to the FIFO chip.

PROGRAMMABLE REGISTERS

Data Transfer Configuration Register (Read/Write)

Register Address 0. This register is cleared by the reset signal.

Bit 0 WD Mode. Set if data transfers are to use the Western Digital WD33C93A protocol, otherwise the Adaptec 6250 protocol will be used.

Bit 1 Parity Chip. Set if this chip is to accumulate Port B parities.

Bit 2 Parity Correct Mode. Set if the parity chip is to correct parallel parity on Port B.

Bit 3 CPU Interface 16 bits wide. If set, the microprocessor data bits are combined with the Port A data bits to effectively produce a 16 bit Port. All accesses by the microprocessor as well as all data transferred using the Port A Request and Acknowledge handshake will transfer 16 bits.

Bit 4 Invert Port A byte address 0. Set to invert the least significant bit of Port A byte address.

Bit 5 Invert Port A byte address 1. Set to invert the most significant bit of Port A byte address.

Bit 6 Checksum Carry Wrap. Set to enable the carry out of the 16 bit checksum adder to carry back into the least significant bit of the adder.

Bit 7 Reset. Writing a 1 to this bit will reset the other registers. This bit resets itself after a maximum of 2 clock cycles and will therefore normally be read as a 0. No other register should be written for a minimum of 4 clock cycles after writing to this bit.

Data Transfer Control Register (Read/Write)

Register Address 1. This register is cleared by the reset signal or by writing to the reset bit.

Bit 0 Enable Data Transfer on Port A. Set to enable the Port A Req/Ack handshake.

Bit 1 Enable Data Transfer on Port B. Set to enable the Port B Req/Ack handshake.

Bit 2 Port A to Port B. If set, data transfer is from Port A to Port B. If reset, data transfer is from Port B to Port A. In order to avoid any glitches on the request lines, the state of this bit should not be altered at the same time as the enable data transfer bits 0 or 1 above.

Bit 3 uProcessor Parity Enable. Set if parity is to be checked on the microprocessor interface. It will only be checked when writing to the Fifo Data Register or reading from the Fifo Data or Checksum Registers, or during a Port A Request/Acknowledge transfer in 16 bit mode. The chip will, however, always re-generate parity ensuring that correct parity is written to the RAM or read on the microprocessor interface.

Bit 4 Port A Parity Enable. Set if parity is to be checked on Port A. It is checked when accessing the Fifo Data Register in 16 bit mode, or during a Port A Request/Acknowledge transfer. The chip will, however, always re-generate parity ensuring that correct parity is written to the RAM or read on the Port A interface.

Bit 5 Port B Parity Enable. Set if Port B data has valid byte parities. If it is not set, byte parity is generated internally to the chip when writing to the RAMs. Byte parity is not checked when writing from Port B, but always checked when reading to Port B.

Bit 6 Checksum Enable. Set to enable writing to the 16 bit checksum register. This register accumulates a 16 bit checksum for all RAM accesses, including accesses to the Fifo Data Register, as well as all writes to the checksum register. This bit must be reset before reading from the Checksum Register.

Bit 7 Port A Master. Set if Port A is to operate in the master mode on Port A during the data transfer.

Data Transfer Status Register (Read Only)

Register Address 2. This register is cleared by the reset signal or by writing to the reset bit.

Bit 0 Data in RAM X or RAM Y. Set if any bits are true in the RAM X, RAM Y, or Port A byte address registers.

Bit 1 uProc Port Parity Error. Set if the uProc Parity Enable bit is set and a parity error is detected on the microprocessor interface during any RAM access or write to the Checksum Register in 16 bit mode.

Bit 2 Port A Parity Error. Set if the Port A Parity Enable bit is set and a parity error is detected on the Port A interface during any RAM access or write to the Checksum Register.

Bit 3 Port B Parallel Parity Error. Set if the chip is configured as the parity chip, is not in parity correct mode, and a non zero result is detected when the Parity Sync signal is true. It is also set whenever data is read out onto Port B and the data being read back through the bidirectional buffer does not compare.

Bits 4-7 Port B Bytes 0-3 Parity Error. Set whenever the data being read out of the RAMs on the Port B side has bad parity.

Ram Access Control Register (Read/Write)

Register Address 3. This register is cleared by the reset signal or by writing to the reset bit. The Enable Data Transfer bits in the Data Transfer Control Register must be reset before attempting to write to this register, else the write will be ignored.

Bit 0 Port A byte address 0. This bit is the least significant byte address bit. It is read directly bypassing any inversion done by the invert bit in the Data Transfer Configuration Register.

Bit 1 Port A byte address 1. This bit is the most significant byte address bit. It is read directly bypassing any inversion done by the invert bit in the Data Transfer Configuration Register.

Bit 2 Port A to RAM Y. Set if Port A is accessing RAM Y, and reset if it is accessing RAM X .

Bit 3 Port B to RAM Y. Set if Port B is accessing RAM Y, and reset if it is accessing RAM X .

Bit 4 Long Burst. If the chip is configured to transfer data on Port A as a master, and this bit is reset, the chip will only negate Port A Ack/Rdy after every 8 bytes, or 4 words in 16 bit mode, have been transferred. If this bit is set, Port A Ack/Rdy will be negated every 16 bytes, or 8 words in 16 bit mode.

Bits 5-7 Not Used.

RAM X Address Register (Read/Write)

Register Address 4. This register is cleared by the reset signal or by writing to the reset bit. The Enable Data Transfer bits in the Data Transfer Control

Register must be reset before attempting to write to this register, else the write will be ignored.

Bits 0-4	RAM X word address
Bit 5	RAM X full
Bits 6-7	Not Used

RAM Y Address Register (Read/Write)

Register Address 5. This register is cleared by the reset signal or by writing to the reset bit. The Enable Data Transfer bits in the Data Transfer Control Register must be reset before attempting to write to this register, else the write will be ignored.

Bits 0-4	RAM Y word address
Bit 5	RAM Y full
Bits 6-7	Not Used

Fifo Data Register (Read/Write)

Register Address 6. The Enable Data Transfer bits in the Data Transfer Control Register must be reset before attempting to write to this register, else the write will be ignored. The Port A to Port B bit in the Data Transfer Control register must also be set before writing this register. If it is not, the RAM controls will be incremented but no data will be written to the RAM. For consistency, the Port A to PortB should be reset prior to reading this register.

Bits 0-7 are Fifo Data. The microprocessor may access the FIFO by reading or writing this register. The RAM control registers are updated as if the access was using Port A. If the chip is configured with a 16 bit CPU Interface the most significant byte will use the Port A 0-7 data lines, and each Port A access will increment the Port A byte address by 2.

Port A Checksum Register (Read/Write)

Register Address 7. This register is cleared by the reset signal or by writing to the reset bit.

Bits 0-7 are Checksum Data. The chip will accumulate a 16 bit checksum for all Port A accesses. If the chip is configured with a 16 bit CPU interface, the most significant byte is read on the Port A 0-7 data lines. If data is written directly to this register it is added to the current contents rather than overwriting them. It is important to note that the Checksum Enable bit in the Data Transfer Control Register must be set to write this register and reset to read it.

PROGRAMMING THE FIFO CHIP

In general the fifo chip is programmed by writing to the data transfer configuration and control registers to enable a data transfer, and by reading the data transfer status register at the end of the transfer to check the completion status. Usually the data transfer itself will take place with both the Port A and the Port B handshakes enabled, and in this case the data transfer itself should be done without any other microprocessor interaction. In some applications, however, the Port A handshake may not be enabled, and it will be necessary for the microprocessor to fill or empty the fifo by repeatedly writing or reading the Fifo Data Register.

Since the fifo chip has no knowledge of any byte counts, there is no way of telling when any data transfer is complete by reading any register within this chip itself. Determination of whether the data transfer has been completed must therefore be done by some other circuitry outside this chip.

The following C language routines illustrate how the parity FIFO chip may be programmed. The routines assume that both Port A and the microprocessor port are connected to the system microprocessor, and return a size code of 16 bits, but that the hardware addresses the Fifo chip as long 32 bit registers.

```
struct FIFO_regs {
  unsigned char config,a1,a2,a3 ;
  unsigned char control,b1,b2,b3;
  unsigned char status,c1,c2,c3;
  unsigned char ram_access_control,d1,d2,d3;
  unsigned char ram_X_addr,e1,e2,e3;
  unsigned char ram_Y_addr,f1,f2,f3;
  unsigned long data;
  unsigned int checksum,h1;
  };

#define FIFO1 ((struct  FIFO_regs*) FIFO_BASE_ADDRESS)

#define FIFO_RESET 0x80
#define FIFO_16_BITS 0x08
#define FIFO_CARRY_WRAP 0x40
#define FIFO_PORT_A_ENABLE 0x01
#define FIFO_PORT_B_ENABLE 0x02
#define FIFO_PORT_ENABLES 0x03
#define FIFO_PORT_A_TO_B 0x04
#define FIFO_CHECKSUM_ENABLE 0x40
#define FIFO_DATA_IN_RAM 0x01
#define FIFO_FORCE_RAM_FULL 0x20

#define PORT_A_TO_PORT_B(fifo) ((fifo-> control ) & 0x04)
#define PORT_A_BYTE_ADDRESS(fifo) ((fifo->ram_access_control) &
  0x03)
#define PORT_A_TO_RAM_Y(fifo)  ((fifo->ram_access_control ) &
  0x04)
#define PORT_B_TO_RAM_Y(fifo)  ((fifo-> ram_access_control ) &
  0x08)

/************************************************************
       The following routine initiates a Fifo data transfer using
two values passed to it.

  config_data    This is the data to be written to the
                 configuration register.

  control_data   This is the data to be written to the Data
                 Transfer Control Register.  If the data transfer
                 is to take place automatically using both the
                 Port Aand Port B handshakes, both data transfer
                 enables bits should be set in this parameter.
************************************************************/

FIFO_initiate_data_transfer(config_data, control_data)
unsigned char config_data, control_data;
{
  FIFO1->config = config_data ¦ FIFO_RESET;     /* Set
       Configuration value & Reset */
  FIFO1->control = control_data & (~FIFO_PORT_ENABLES);    /* Set
       everything but enables */
  FIFO1->control = control_data ;            /* Set data transfer
enables */
}

/************************************************************
       The following routine forces the transfer of any odd bytes
that have been left in the Fifo at the end of a data transfer.
It first disables both ports, then forces the Ram Full bits, and
then re-enables the appropriate Port.
************************************************************/

FIFO_force_odd_length_transfer()
{
  FIFO1->control &= ~FIFO_PORT_ENABLES; /* Disable Ports A & B */
  if (PORT_A_TO_PORT_B(FIFO1))  {
       if (PORT_A_TO_RAM_Y(FIFO1))  {
            FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL; /* Set RAM Y
full */
       }
       else FIFO1->ram_X_addr = FIFO_FORCE_RAM_FULL ;  /* Set RAM
X full */
       FIFO1->control |= FIFO_PORT_B_ENABLE ;     /* Re-Enable
Port B */
  }
  else {
       if (PORT_B_TO_RAM_Y(FIFO1))  {
            FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL ;       /* Set
RAM Y full */
       }
       else FIFO1->ram_X_addr = FIFO_FORCE_RAM_FULL ;  /* Set RAM
X full */
       FIFO1->control |= FIFO_PORT_A_ENABLE ;     /* Re-Enable
Port A */
  }
}

/************************************************************
       The following routine returns how many odd bytes have been
left in the Fifo at the end of a data transfer.
************************************************************/

int FIFO_count_odd_bytes()
{
  int number_odd_bytes;
  number_odd_bytes=0;
  if (FIFO1->status & FIFO_DATA_IN_RAM)   {
       if (PORT_A_TO_PORT_B(FIFO1))  {
            number_odd_bytes = (PORT_A_BYTE_ADDRESS(FIFO1)) ;
            if (PORT_A_TO_RAM_Y(FIFO1))
                 number_odd_bytes += (FIFO1->ram_Y_addr) * 4 ;
            else number_odd_bytes += (FIFO1->ram_X_addr) * 4 ;
       }
       else {
            if (PORT_B_TO_RAM_Y(FIFO1))
                 number_odd_bytes = (FIFO1->ram_Y_addr) * 4 ;
            else number_odd_bytes = (FIFO1->ram_X_addr) * 4 ;
       }
  }
  return (number_odd_bytes);
}

/************************************************************
       The following routine tests the microprocessor interface of
the chip.  It first writes and reads the first 6 registers.  It
then writes 1s, 0s, and an address pattern to the RAM, reading the
data back and checking it.

       The test returns a bit significant error code where each
bit represents the address of the registers that failed.

  Bit 0 = config register failed
  Bit 1 = control register failed
  Bit 2 = status register failed
  Bit 3 = ram access control register failed
  Bit 4 = ram X address register failed
  Bit 5 = ram Y address register failed
  Bit 6 = data register failed
  Bit 7 = checksum register failed
************************************************************/

#define RAM_DEPTH 64       /* number of long words in Fifo Ram */

reg_expected_data[6] = { 0x7F, 0xFF, 0x00, 0x1F, 0x3F, 0x3F };

char  FIFO_uprocessor_interface_test()
{
  unsigned long test_data;
  char *register_addr;
  int i;
  char j,error;
  FIFO1->config = FIFO_RESET;        /* reset the chip */
  error=0;
  register_addr =(char *) FIFO1;
  j=1;

  /* first test registers 0 thru 5 */

  for  (i=0; i<6; i++)  {
       *register_addr = 0xFF;        /* write test data */
       if (*register_addr != reg_expected_data[i]) error |= j;
       *register_addr = 0;      /* write 0s to register */
       if (*register_addr) error |= j;
       *register_addr = 0xFF;        /* write test data again */
       if (*register_addr != reg_expected_data[i]) error |= j;
       FIFO1->config = FIFO_RESET;        /* reset the chip */
       if (*register_addr) error |= j; /* register should be 0 */
       register_addr++;          /* go to next register */
       j <<=1;
  }

       /* now test Ram data & checksum registers
       test 1s throughout Ram & then test 0s */

  for (test_data = -1; test_data != 1; test_data++)   {  /* test
for 1s & 0s */
       FIFO1->config = FIFO_RESET | FIFO_16_BITS ;
       FIFO1->control = FIFO_PORT_A_TO_B;
       for (i=0;i<RAM_DEPTH;i++)          /* write data to RAM */
            FIFO1->data = test_data;
       FIFO1->control = 0;
       for (i=0;i<RAM_DEPTH;i++)
            if (FIFO1->data != test_data) error |= j;     /* read
& check data */
       if (FIFO1->checksum) error |= 0x80;         /* checksum
should = 0 */
  }

       /* now test Ram data with address pattern
        uses a different pattern for every byte */

  test_data=0x00010203;          /* address pattern start */
  FIFO1->config = FIFO_RESET | FIFO_16_BITS | FIFO_CARRY_WRAP;
  FIFO1->control = FIFO_PORT_A_TO_B | FIFO_CHECKSUM_ENABLE;
  for (i=0;i<RAM_DEPTH;i++) {
       FIFO1->data = test_data;      /* write address pattern */
       test_data += 0x04040404;
  }
  test_data=0x00010203;          /* address pattern start */
  FIFO1->control = FIFO_CHECKSUM_ENABLE;
  for (i=0;i<RAM_DEPTH;i++) {
       if (FIFO1->status != FIFO_DATA_IN_RAM)
            error |= 0x04;      /* should be data in ram */
       if (FIFO1->data != test_data) error |= j;     /* read &
check address pattern */
       test_data += 0x04040404;
  }
  if (FIFO1->checksum != 0x0102) error |= 0x80;     /* test
checksum of address pattern */
  FIFO1->config = FIFO_RESET | FIFO_16_BITS ;  /* inhibit carry
wrap */
  FIFO1->checksum = 0xFEFE;     /* writing adds to checksum */
```

```
if (FIFO1->checksum) error |=0x80;  /* checksum should be 0 */
if (FIFO1->status) error |= 0x04;   /* status should be 0 */
return (error);
  }
```

CLAIMS

1. Network server apparatus for use with a data network and a mass storage device, comprising:

an interface processor unit coupleable to said network and to said mass storage device;

a host processor unit capable of running remote procedures defined by a client node on said network;

means in said interface processor unit for satisfying requests from said network to store data from said network on said mass storage device;

means in said interface processor unit for satisfying requests from said network to retrieve data from said mass storage device to said network; and

means in said interface processor unit for transmitting predefined categories of messages from said network to said host processor unit for processing in said host processor unit, said transmitted messages including all requests by a network client to run client-defined procedures on said network server apparatus.

2. Apparatus according to claim 1, wherein said interface processor unit comprises:

a network control unit coupleable to said network;

a data control unit coupleable to said mass storage device;

a buffer memory;

means in said network control unit for transmitting to said data control unit requests from said network to store specified storage data from said network on said mass storage device;

means in said network control unit for transmitting said specified storage data from said network to said buffer memory and from said buffer memory to said data control unit;

means in said network control unit for transmitting to said data control unit requests from said network to retrieve specified retrieval data from said mass storage device to said network;

means in said network control unit for transmitting said specified retrieval data from said data control unit to said buffer memory and from said buffer memory to said network; and

means in said network control unit for transmitting said predefined categories of messages from said network to said host processing unit for processing by said host processing unit.

3. Apparatus according to claim 2, wherein said data control unit comprises:

a storage processor unit coupleable to said mass storage device;

a file processor unit;

means on said file processor unit; for translating said file system level storage requests from said network into requests to store data at specified physical storage locations in said mass storage device;

means on said file processor unit for instructing said storage processor unit to write data from said buffer memory into said specified physical storage locations in said mass storage device;

means on said file processor unit for translating file system level retrieval requests from said network into requests to retrieve data from specified physical retrieval locations in said mass storage device;

means on said file processor unit for instructing said storage processor unit to retrieve data from said specified physical retrieval locations in said mass storage device to said buffer memory if said data from said specified physical locations is not already in said buffer memory; and

means in said storage processor unit for transmitting data between said buffer memory and said mass storage device.

4. Network server apparatus for use with a data network and a mass storage device, comprising:

a network control unit coupleable to said network;

a data control unit coupleable to said mass storage device;

a buffer memory;

means for transmitting from said network control unit to said data control unit requests from said network to store specified storage data from said network on said mass storage device;

means for transmitting said specified storage data by DMA from said network control unit to said buffer memory and by DMA from said buffer memory to said data control unit;

means for transmitting from said network control unit to said data control unit requests from said network to retrieve specified retrieval data from said mass storage device to said network; and

means for transmitting said specified retrieval data by DMA from said data control unit to said buffer memory and by DMA from said buffer memory to said network control unit.

5. Apparatus according to claim 1, for use further with a buffer memory, and wherein said requests from said network to store and retrieve data include file system level storage and retrieval requests respectively, and wherein said interface processor unit comprises:

a storage processor unit coupleable to said mass storage device;

a file processor unit;

means on said file processor unit for translating said file system level storage requests into requests to store data at specified physical storage locations in said mass storage device;

means on said file processor unit for instructing said storage processor unit to write data from said buffer memory into said specified physical storage locations in said mass storage device;

means on said file processor unit for translating said file system level retrievable requests into requests to retrieve data from specified physical retrievable locations in said mass storage device;

means on said file processor unit for instructing said storage processor unit to retrieve data from said specified physical retrievable locations in said mass storage device to said buffer memory if said data from said specified physical locations is not already in said buffer memory; and

means in said storage processor unit for transmitting data between said buffer memory and said mass storage device.

6. Network server apparatus for use with a data network, comprising:

a network controller coupleable to said network to receive incoming information packets over said network, said incoming information packets including certain packets which contain part or all of a request to said server apparatus, said request being in either a first or a second class of requests to said server apparatus;

a first additional processor;

an interchange bus different from said network and coupled between said network controller and said first additional processor;

means in said network controller for detecting and satisfying requests in said first class of requests contained in said certain incoming information packets, said network controller lacking means in said network controller for satisfying requests in said second class of requests;

means in said network controller for detecting and assembling into assembled requests, requests in said second class of requests contained in said certain incoming information packets;

means for delivering said assembled requests from said network controller to said first additional processor over said interchange bus; and

means in said first additional processor for further processing said assembled requests in said second class of requests.

7. Apparatus according to claim 6 wherein said packets each include a network node destination address, and wherein said means in said network controller for detecting and assembling into assembled requests, assembles said assembled requests in a format which omits said network node destination addresses.

8. Apparatus according to claim 6 wherein said means in said network controller for detecting and satisfying requests in said first class of requests, assembles said requests in said first class of requests into assembled requests before satisfying said requests in said first class of requests.

9. Apparatus according to claim 6, wherein said packets each include a network node destination address, wherein said means in said network controller for detecting and assembling into assembled requests, assembles said assembled requests in a format which omits said network node destination addresses, and wherein said means in said network controller for detecting and satisfying requests in said first class of requests, assembles said requests in said first class of requests, in a format which omits said network node destination addresses, before satisfying said requests in said first class of requests.

10. Apparatus according to claim 6, wherein said means in said network controller for detecting and satisfying requests in said first class includes means for preparing an outgoing message in response to one of said first class of requests, means for packaging said outgoing message in outgoing information packets suitable for transmission over said network, and means for transmitting said outgoing information packets over said network.

11. Apparatus according to claim 6, further comprising a buffer memory coupled to said interchange bus, and wherein said means for delivering said assembled requests comprises:

means for transferring the contents of said assembled requests over said interchange bus into said buffer memory; and

means for notifying said first additional processor of the presence of said contents in said buffer memory.

12. Apparatus according to claim 6, wherein said means in said first additional processor for further processing said assembled requests includes means for preparing an outgoing message in response to one of said second class of requests, said apparatus further comprising means for delivering said outgoing message from said first additional processor to said network controller over said interchange bus, said network controller further comprising means in said network controller for packaging said outgoing message in outgoing information packets suitable for transmission over said network, and means in said network controller for transmitting said outgoing information packages over said network.

13. Apparatus according to claim 6, wherein said first class of requests comprises requests for an address of said server apparatus, and wherein said

means in said network controller for detecting and satisfying requests in said first class comprises means for preparing a response packet to such an address request and means for transmitting said response packet over said network.

14. Apparatus according to claim 6 for use further with a second data network, said network controller being coupleable further to said second network, wherein said first class of requests comprises requests to route a message to a destination reachable over said second network, and wherein said means in said network controller for detecting and satisfying requests in said first class comprises means for detecting that one of said certain packets comprises a request to route a message contained in said one of said certain packets to a destination reachable over said second network, and means for transmitting said message over said second network.

15. Apparatus according to claim 14 for use further with a third data network, said network controller further comprising means in said network controller for detecting particular requests in said incoming information packets to route a message contained in said particular requests, to a destination reachable over said third network, said apparatus further comprising:

a second network controller coupled to said interchange bus and coupleable to said third data network;

means for delivering said message contained in said particular requests to said second network controller over said interchange bus; and

means in said second network controller for transmitting said message contained in said particular requests over said third network.

16. Apparatus according to claim 6, for use further with a third data network, said network controller further comprising means in said network controller for detecting particular requests in said incoming information packets to route a message contained in said particular requests, to a destination reachable over said third network, said apparatus further comprising:

a second network controller coupled to said interchange bus and coupleable to said third data network;

means for delivering said message contained in said particular requests to said second network controller over said interchange bus; and

means in said second network controller for transmitting said message contained in said particular requests over said third network.

17. Apparatus according to claim 6 for use further with a mass storage device, wherein said first additional processor comprises a data control unit coupleable to said mass storage device, wherein said second class of requests comprises remote calls to procedures for managing a file system in said mass storage device, and wherein said means in said first additional processor for further processing said assembled requests in said second class of requests comprises means for executing file system procedures on said mass storage device in response to said assembled requests.

18. Apparatus according to claim 17 wherein said file system procedures include a read procedure for reading data from said mass storage device,

said means in said first additional processor for further processing said assembled requests including means for reading data from a specified location in said mass storage device in response to a remote call to said read procedure,

said apparatus further including means for delivering said data to said network controller,

said network controller further comprising means on said network controller for packaging said data in outgoing information packets suitable for transmission over said network, and means for transmitting said outgoing information packets over said network.

19. Apparatus according to claim 18, wherein said means for delivering comprises:

a system buffer memory coupled to said interchange bus;

means in said data control unit for transferring said data over said interchange bus into said buffer memory; and

means in said network controller for transferring said data over said interchange bus from said system buffer memory to said network controller.

20. Apparatus according to claim 17, wherein said file system procedures include a read procedure for reading a specified number of bytes of data from said mass storage device beginning at an address specified in logical terms including a file system ID and a file ID, said means for executing file system procedures comprising:

means for converting the logical address specified in a remote call to said read procedure to a physical address; and

means for reading data from said physical address in said mass storage device.

21. Apparatus according to claim 20, wherein said mass storage device comprises a disk drive having a numbered tracks and sectors, wherein said logical address specifies said file system ID, said file ID,

and a byte offset, and wherein said physical address specifies a corresponding track and sector number.

22. Apparatus according to claim 17, wherein said file system procedures include a read procedure for reading a specified number of bytes of data from said mass storage device beginning at an address specified in logical terms including a file system ID and a file ID,

said data control unit comprising a file processor coupled to said interchange bus and a storage processor coupled to said interchange bus and coupleable to said mass storage device,

said file processor comprising means for converting the logical address specified in a remote call to said read procedure to a physical address,

said apparatus further comprising means for delivering said physical address to said storage processor,

said storage processor comprising means for reading data from said physical address in said mass storage device and for transferring said data over said interchange bus into said buffer memory; and

means in said network controller for transferring said data over said interchange bus from said system buffer memory to said network controller.

23. Apparatus according to claim 17, wherein said file system procedures include a write procedure for

writing data contained in an assembled request, to said mass storage device,

said means in said first additional processor for further processing said assembled requests including means for writing said data to a specified location in said mass storage device in response to a remote call to said read procedure.

24. Apparatus according to claim 6, wherein said first additional processor comprises a host computer coupled to said interchange bus, wherein said second class of requests comprises remote calls to procedures other than procedures for managing a file system, and wherein said means in said first additional processor for further processing said assembled requests in said second class of requests comprises means for executing remote procedure calls in response to said assembled requests.

25. Apparatus according to claim 24, for use further with a mass storage device and a data control unit coupleable to said mass storage device and coupled to said interchange bus, wherein said network controller further comprises means in said network controller for detecting and assembling remote calls, received over said network, to procedures for managing a file system in said mass storage device, and wherein said data control unit comprises means for executing file system procedures on said mass storage device in

response to said remote calls to procedures for managing a file system in said mass storage device.

26. Apparatus according to claim 24, further comprising means for delivering all of said incoming information packets not recognized by said network controller to said host computer over said interchange bus.

27. Apparatus according to claim 26, wherein said network controller comprises:

a microprocessor;

a local instruction memory containing local instruction code;

a local bus coupled between said microprocessor and said local instruction memory;

bus interface means for interfacing said microprocessor with said interchange bus at times determined by said microprocessor in response to said local instruction code; and

network interface means for interfacing said microprocessor with said data network,

said local instruction memory including all instruction code necessary for said microprocessor to perform said function of detecting and satisfying requests in said first class of requests, and all instruction code necessary for said microprocessor to perform said function of detecting and assembling into assembled requests, requests in said second class of requests.

28. Network server apparatus for use with a data network, comprising:

a network controller coupleable to said network to receive incoming information packets over said network, said incoming information packets including certain packets which contain part or all of a message to said server apparatus, said message being in either a first or a second class of messages to said server apparatus, said messages in said first class of messages including certain messages containing requests;

a host computer;

an interchange bus different from said network and coupled between said network controller and said host computer;

means in said network controller for detecting and satisfying said requests in said first class of messages ;

means for delivering messages in said second class of messages from said network controller to said host computer over said interchange bus; and

means in said host computer for further processing said messages in said second class of messages.

29. Apparatus according to claim 28, wherein said packets each include a network node destination address, and wherein said means for delivering messages

in said second class of messages comprises means in said network controller for detecting said messages in said second class of messages and assembling them into assembled messages in a format which omits said network node destination addresses.

30. Apparatus according to claim 28, wherein said means in said network controller for detecting and satisfying requests in said first class includes means for preparing an outgoing message in response to one of said requests in said first class of messages, means for packaging said outgoing message in outgoing information packets suitable for transmission over said network, and means for transmitting said outgoing information packets over said network.

31. Apparatus according to claim 28, for use further with a second data network, said network controller being coupleable further to said second network, wherein said first class of messages comprises messages to be routed to a destination reachable over said second network, and wherein said means in said network controller for detecting and satisfying requests in said first class comprises means for detecting that one of said certain packets includes a request to route a message contained in said one of said certain packets to a destination reachable over said second network, and means for transmitting said message over said second network.

32. Apparatus according to claim 28, for use further with a third data network, said network controller further comprising means in said network controller for detecting particular messages in said incoming information packets to be routed to a destination reachable over said third network, said apparatus further comprising:

a second network controller coupled to said interchange bus and coupleable to said third data network;

means for delivering said particular messages to said second network controller over said interchange bus, substantially without involving said host computer; and

means in said second network controller for transmitting said message contained in said particular requests over said third network, substantially without involving said host computer.

33. Apparatus according to claim 28, for use further with a mass storage device, further comprising a data control unit coupleable to said mass storage device,

said network controller further comprising means in said network controller for detecting ones of said incoming information packets containing remote calls to procedures for managing a file system in said mass storage device, and means in said network controller for assembling said remote calls from said incoming packets into assembled calls, substantially without involving said host computer,

said apparatus further comprising means for delivering said assembled file system calls to said data control unit over said interchange bus substantially without involving said host computer, and said data control unit comprising means in said data control unit for executing file system procedures on said mass storage device in response to said assembled file system calls, substantially without involving said host computer.

34. Apparatus according to claim 28, further comprising means for delivering all of said incoming information packets not recognized by said network controller to said host computer over said interchange bus.

35. Apparatus according to claim 28, wherein said network controller comprises:

a microprocessor;

a local instruction memory containing local instruction code;

a local bus coupled between said microprocessor and said local instruction memory;

bus interface means for interfacing said microprocessor with said interchange bus at times determined by said microprocessor in response to said local instruction code; and

network interface means for interfacing said microprocessor with said data network,

said local instruction memory including all instruction code necessary for said microprocessor to perform said function of detecting and satisfying requests in said first class of requests.

For the Applicant,

Sanford T. Colb & Co.
C:11071

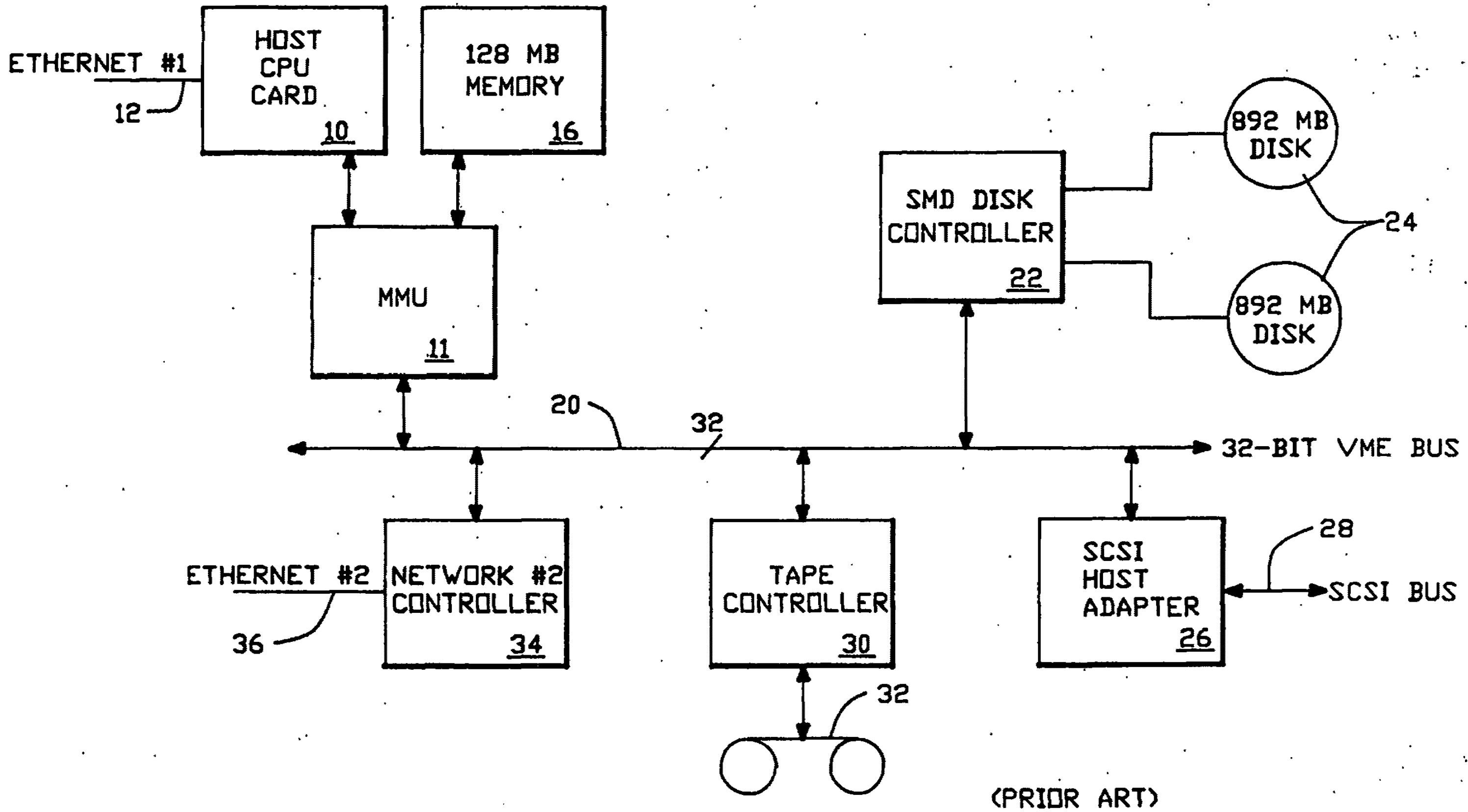

ETHERNET #1
12
HOST CPU CARD
10
128 MB MEMORY
16
MMU
11
SMD DISK CONTROLLER
22
892 MB DISK
892 MB DISK
24
20
32
32-BIT VME BUS
ETHERNET #2
36
NETWORK #2 CONTROLLER
34
TAPE CONTROLLER
30
32
SCSI HOST ADAPTER
26
28
SCSI BUS
(PRIOR ART)

FIG.-1

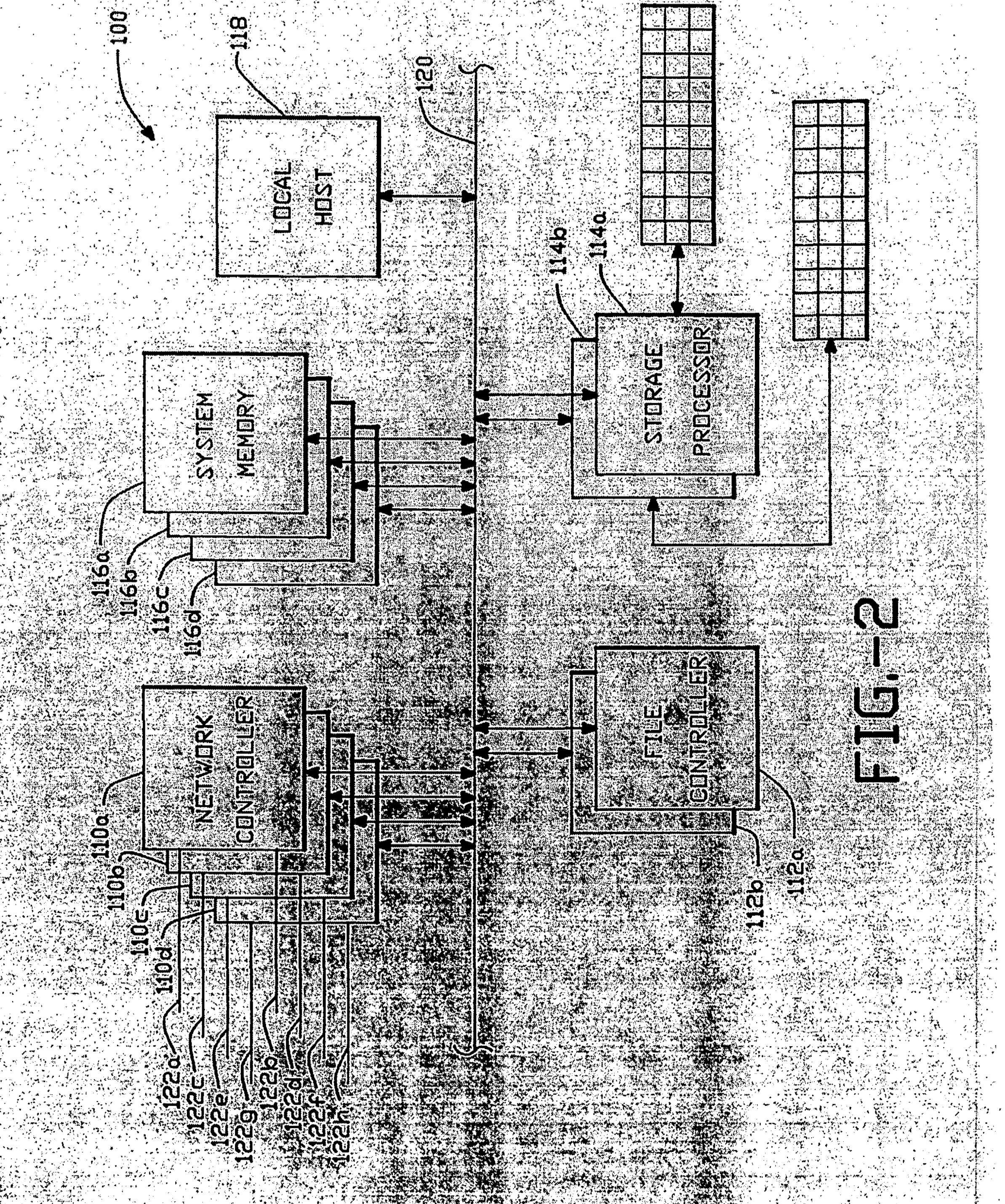

100
118
LOCAL HOST
120
114b
114a
STORAGE PROCESSOR
SYSTEM MEMORY
116a
116b
116c
116d
NETWORK CONTROLLER
110a
110b
110c
110d
FILE CONTROLLER
112b
112a
122a
122c
122e
122g
122b
122d
122f
122h

FIG.-2

AUSPEX SYSTEMS, INC.

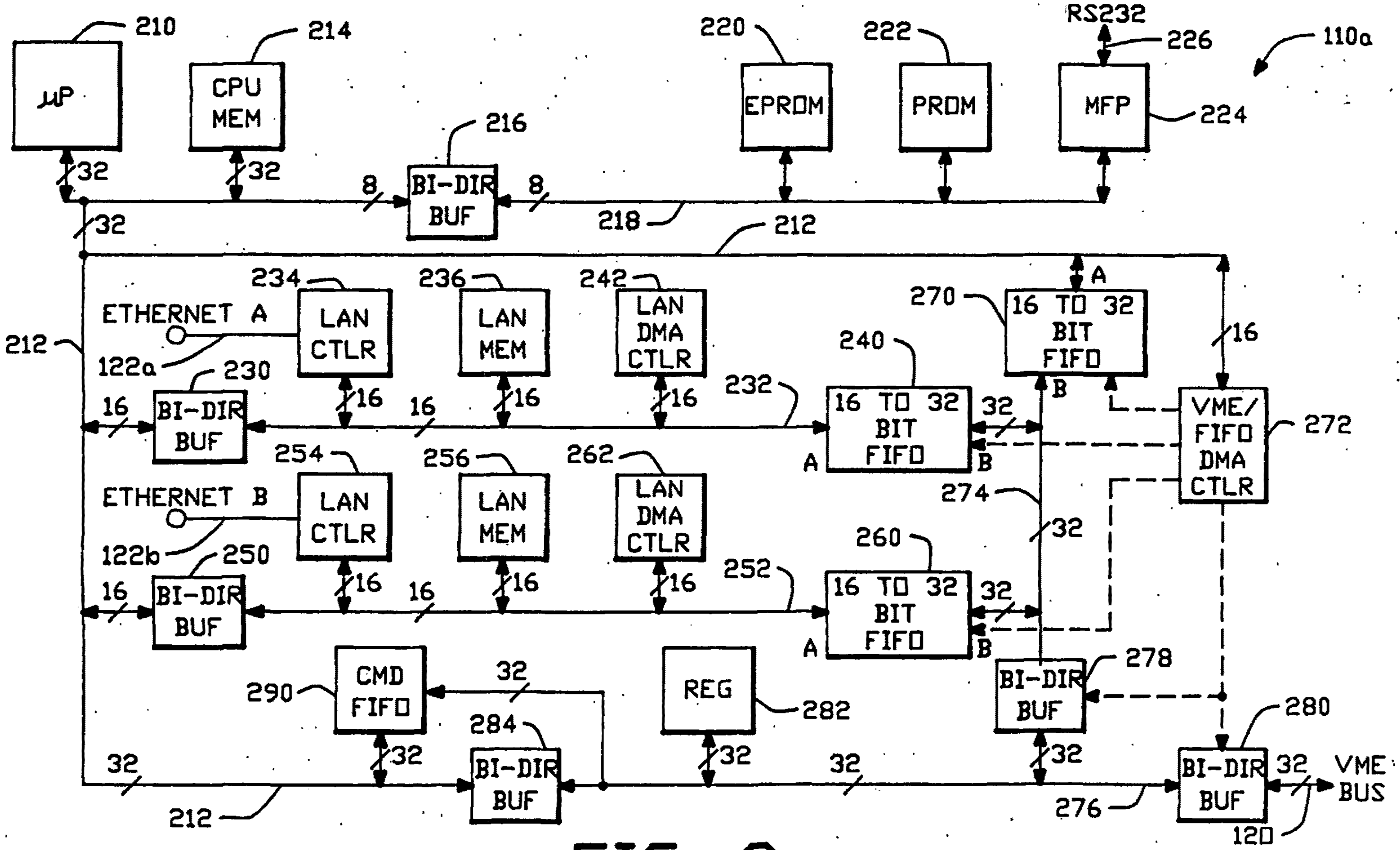


FIG.-3 (NETWORK CONTROLLER)

TWELVE SHEETS
SHEET NO. THREE

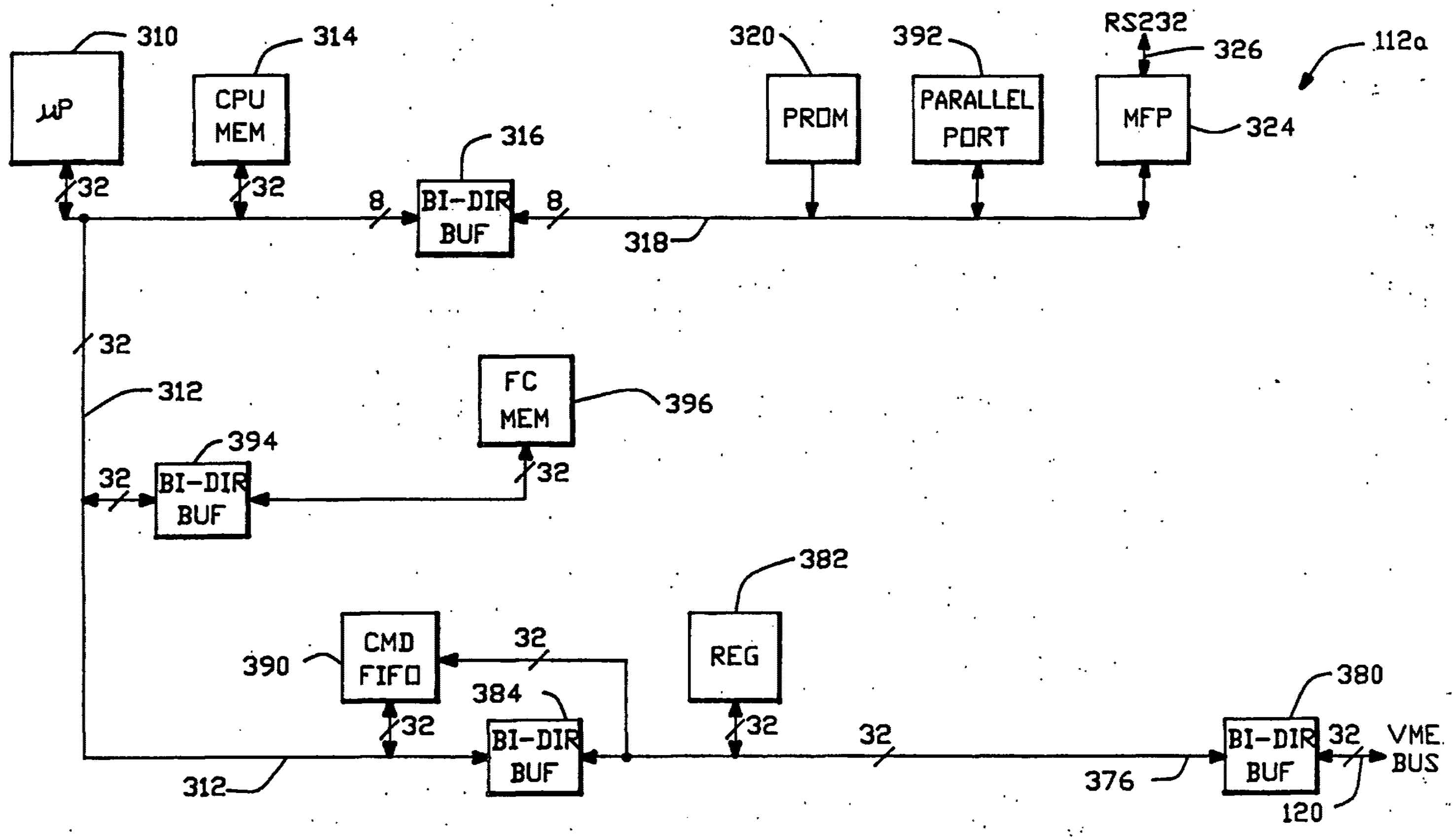


FIG.-4 (FILE CONTROLLER)

AUSPEX SYSTEMS, INC.

TWELVE SHEETS
SHEET NO. FOUR

AUSPEX SYSTEMS, INC.

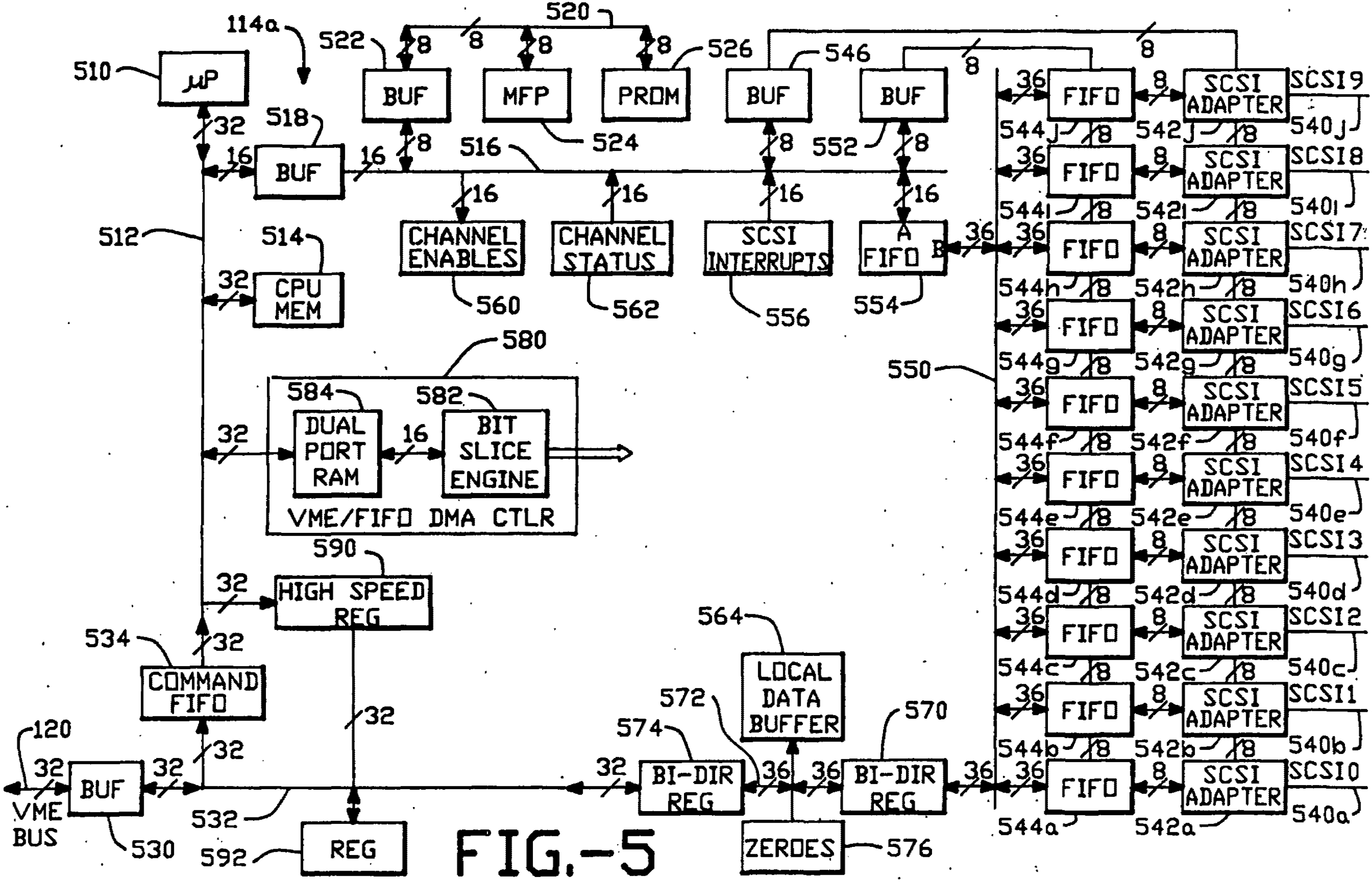
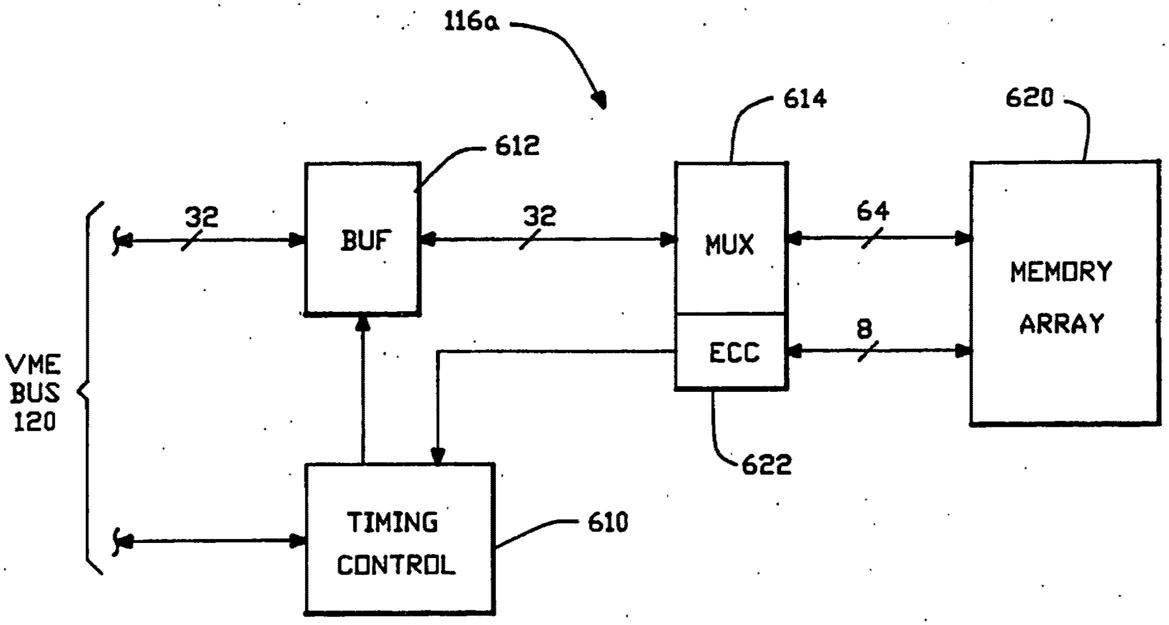


FIG.-5

TWELVE SHEETS
SHEET NO. FIVE

AUSPEX SYSTEMS, INC.



(SYSTEM MEMORY)

FIG.-6

TWELVE SHEETS
SHEET NO. SIX

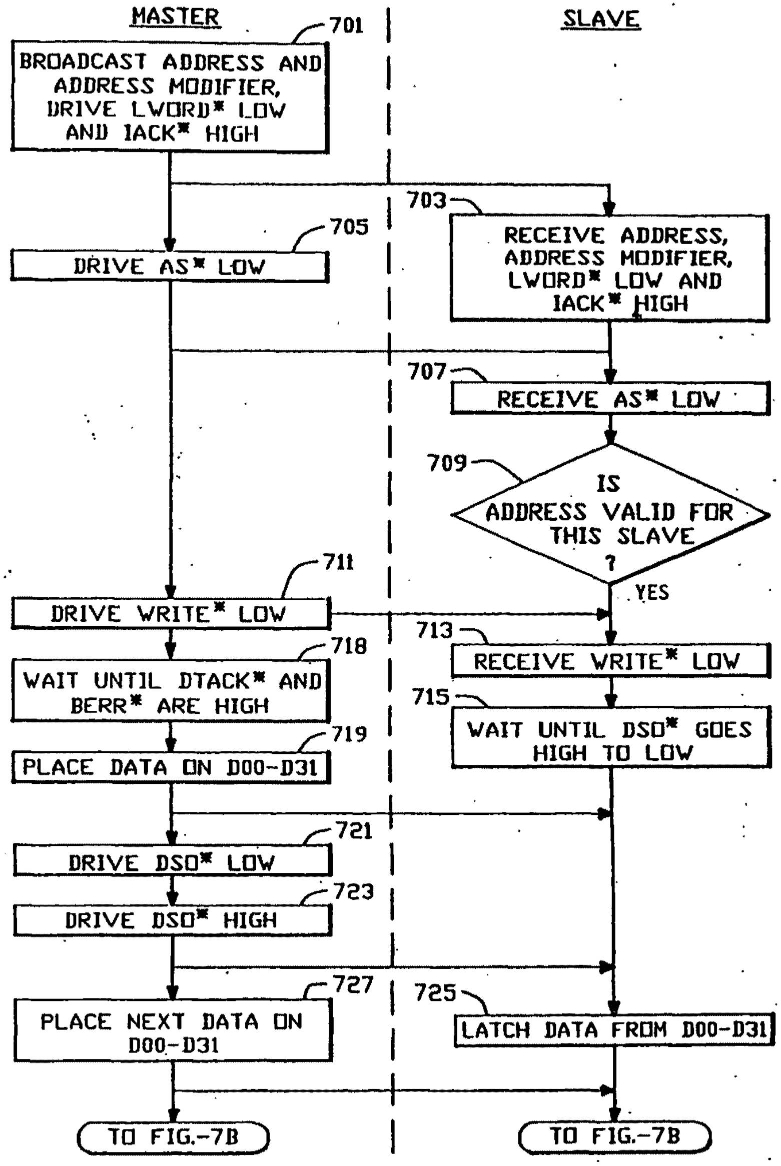

MASTER
SLAVE
701
BROADCAST ADDRESS AND ADDRESS MODIFIER, DRIVE LWORD* LOW AND IACK* HIGH
703
RECEIVE ADDRESS, ADDRESS MODIFIER, LWORD* LOW AND IACK* HIGH
705
DRIVE AS* LOW
707
RECEIVE AS* LOW
709
IS ADDRESS VALID FOR THIS SLAVE ?
YES
711
DRIVE WRITE* LOW
713
RECEIVE WRITE* LOW
718
WAIT UNTIL DTACK* AND BERR* ARE HIGH
715
WAIT UNTIL DSO* GOES HIGH TO LOW
719
PLACE DATA ON D00-D31
721
DRIVE DSO* LOW
723
DRIVE DSO* HIGH
727
PLACE NEXT DATA ON D00-D31
725
LATCH DATA FROM D00-D31
TO FIG.-7B
TO FIG.-7B

FIG.-7A

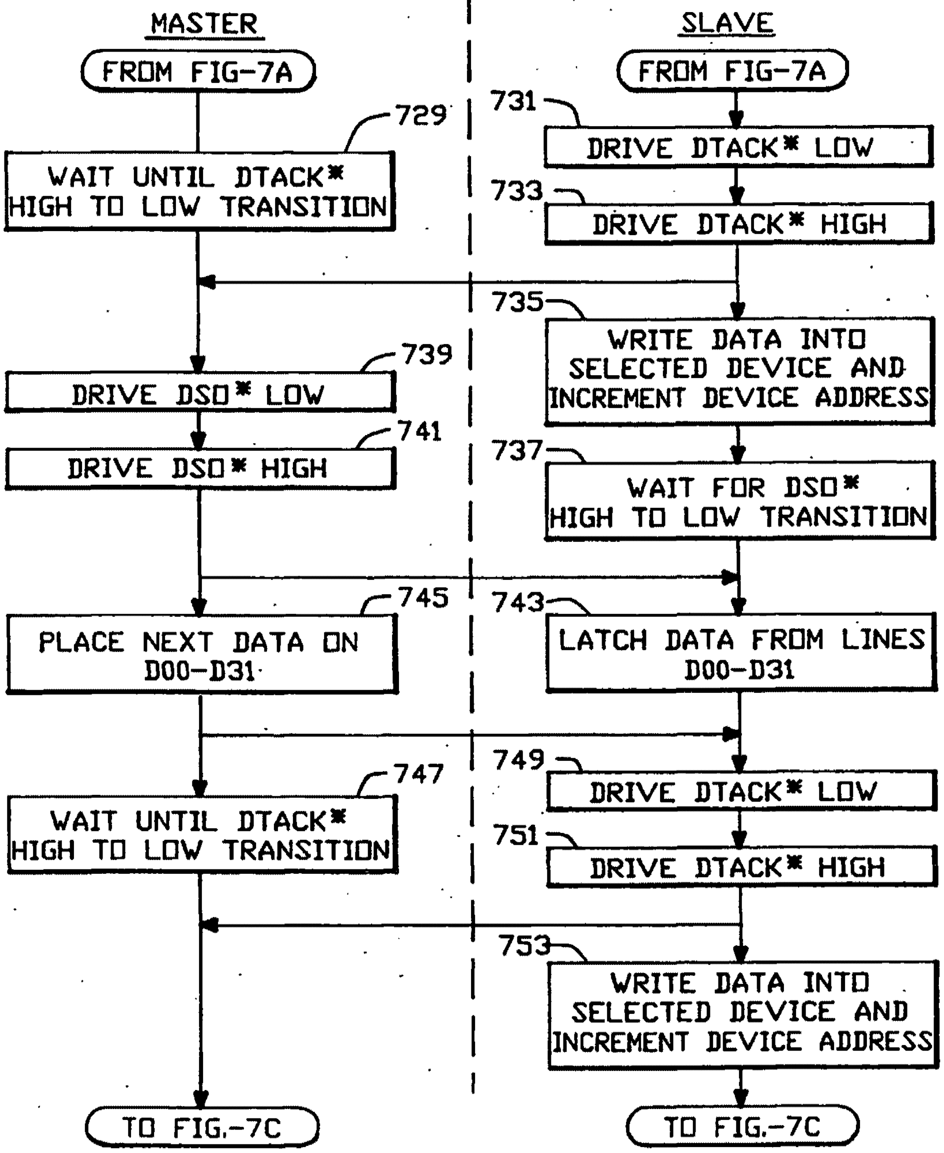

MASTER
SLAVE
FROM FIG-7A
FROM FIG-7A
729
WAIT UNTIL DTACK* HIGH TO LOW TRANSITION
731
DRIVE DTACK* LOW
733
DRIVE DTACK* HIGH
735
WRITE DATA INTO SELECTED DEVICE AND INCREMENT DEVICE ADDRESS
739
DRIVE DSO* LOW
741
DRIVE DSO* HIGH
737
WAIT FOR DSO* HIGH TO LOW TRANSITION
745
PLACE NEXT DATA ON D00-D31
743
LATCH DATA FROM LINES D00-D31
749
DRIVE DTACK* LOW
747
WAIT UNTIL DTACK* HIGH TO LOW TRANSITION
751
DRIVE DTACK* HIGH
753
WRITE DATA INTO SELECTED DEVICE AND INCREMENT DEVICE ADDRESS
TO FIG.-7C
TO FIG.-7C

FIG.-7B

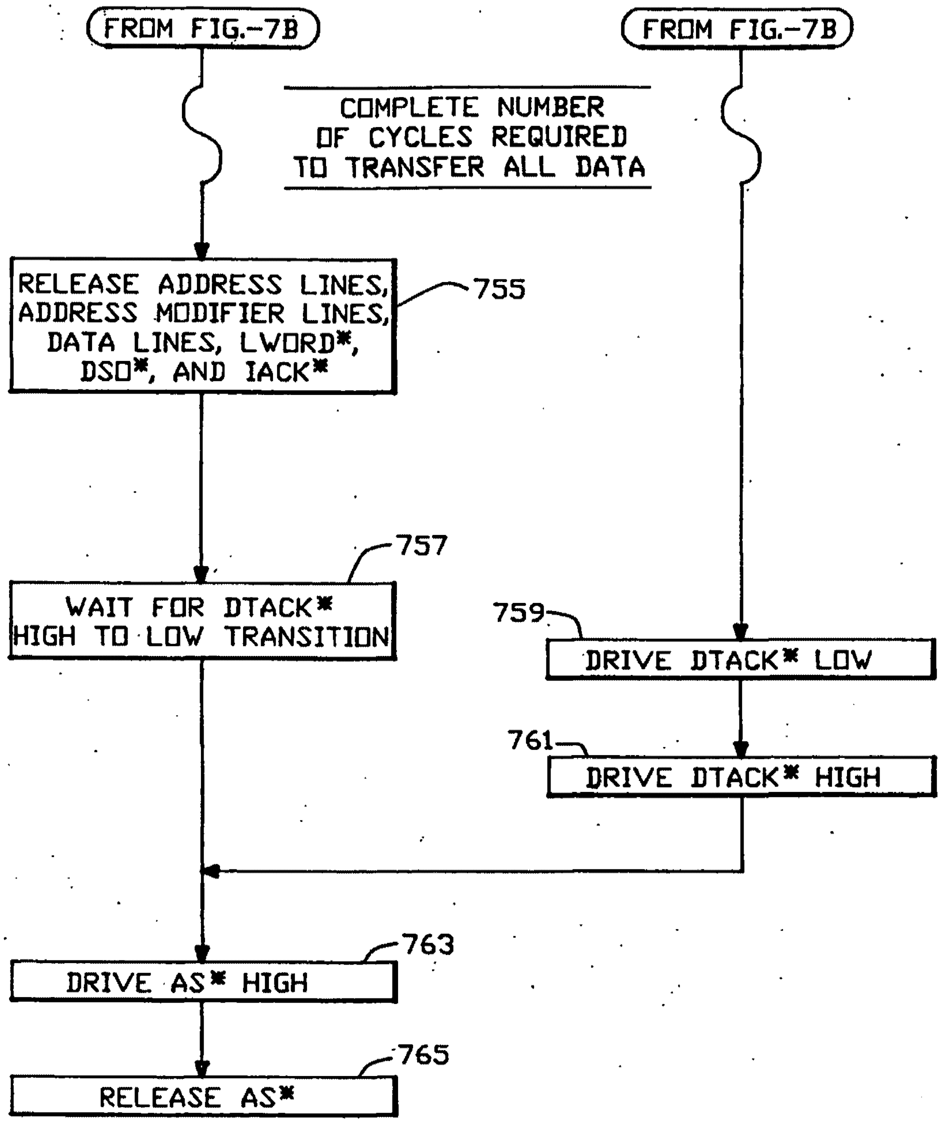

FROM FIG.-7B
FROM FIG.-7B
COMPLETE NUMBER OF CYCLES REQUIRED TO TRANSFER ALL DATA
RELEASE ADDRESS LINES, ADDRESS MODIFIER LINES, DATA LINES, LWORD*, DSO*, AND IACK*
755
757
WAIT FOR DTACK* HIGH TO LOW TRANSITION
759
DRIVE DTACK* LOW
761
DRIVE DTACK* HIGH
763
DRIVE AS* HIGH
765
RELEASE AS*

FIG.-7C

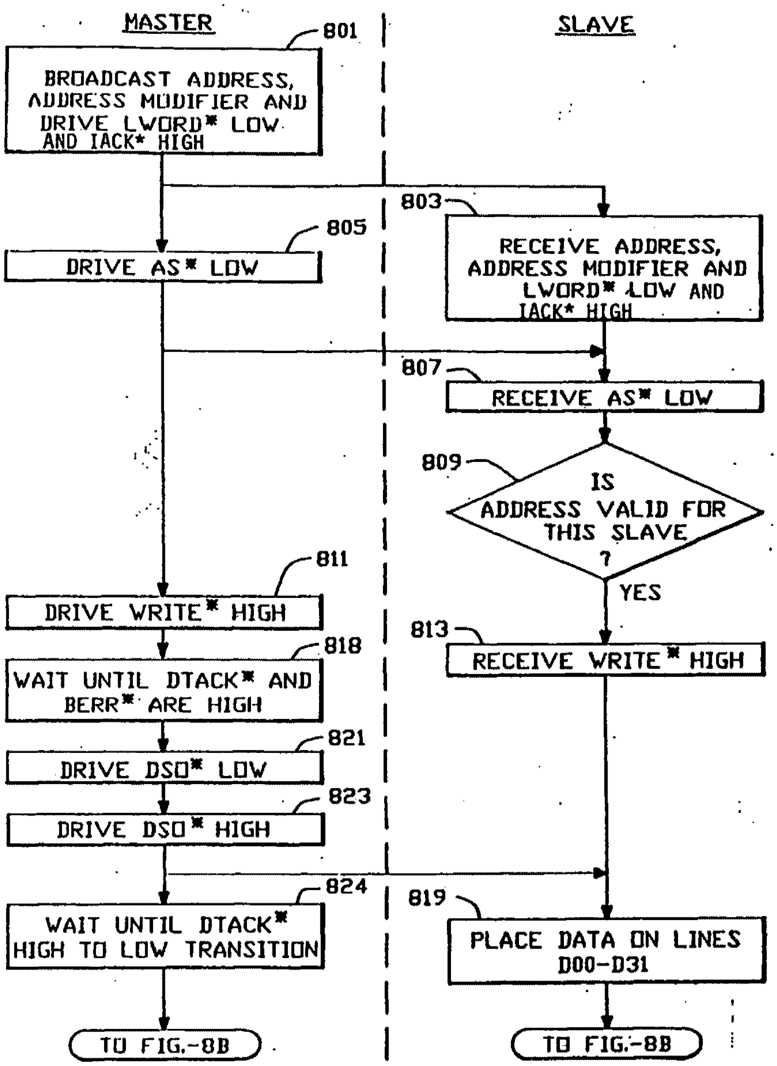

MASTER
SLAVE
801
BROADCAST ADDRESS, ADDRESS MODIFIER AND DRIVE LWORD* LOW AND IACK* HIGH
803
805
DRIVE AS* LOW
RECEIVE ADDRESS, ADDRESS MODIFIER AND LWORD* LOW AND IACK* HIGH
807
RECEIVE AS* LOW
809
IS ADDRESS VALID FOR THIS SLAVE ?
811
YES
DRIVE WRITE* HIGH
813
RECEIVE WRITE* HIGH
818
WAIT UNTIL DTACK* AND BERR* ARE HIGH
821
DRIVE DS0* LOW
823
DRIVE DS0* HIGH
824
819
WAIT UNTIL DTACK* HIGH TO LOW TRANSITION
PLACE DATA ON LINES D00-D31
TO FIG.-8B
TO FIG.-8B

FIG.-8A

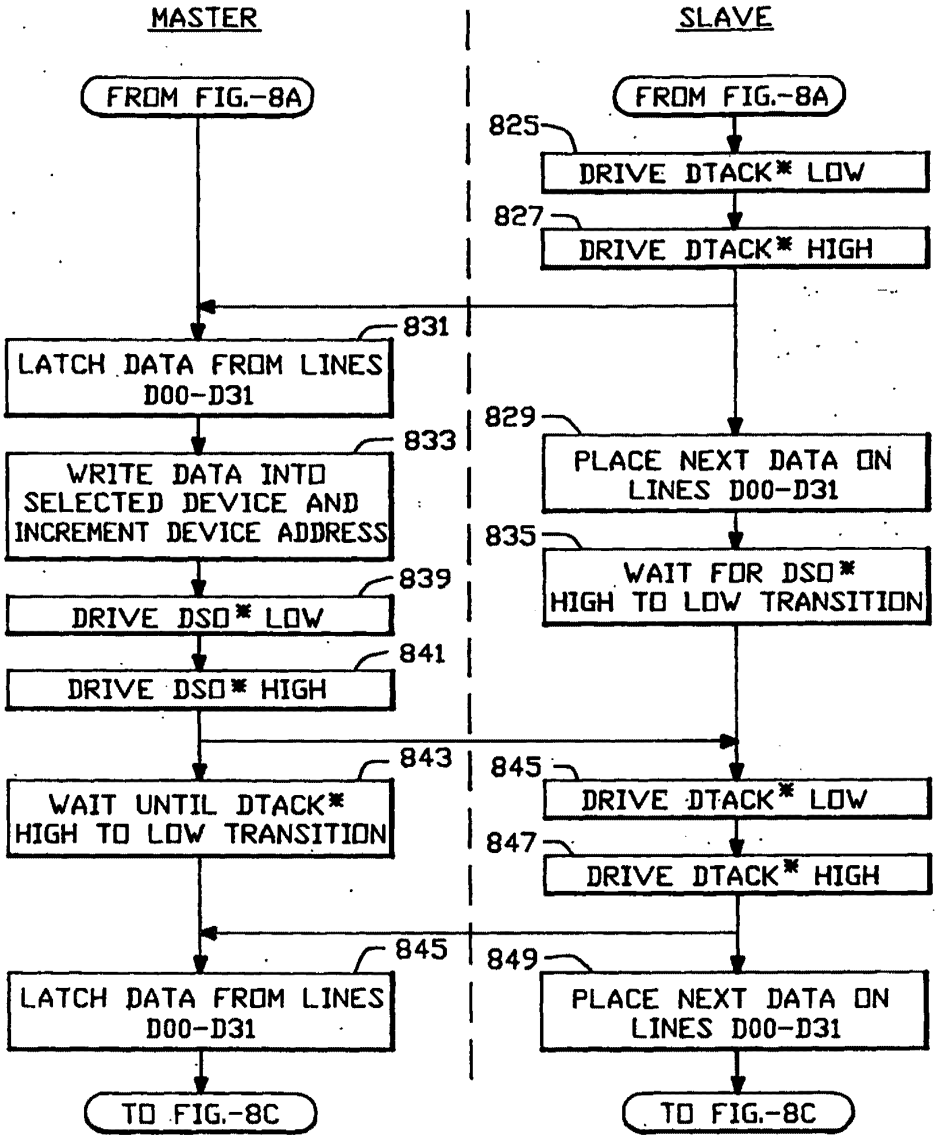

MASTER
SLAVE
FROM FIG.-8A
FROM FIG.-8A
825
DRIVE DTACK* LOW
827
DRIVE DTACK* HIGH
831
LATCH DATA FROM LINES D00-D31
829
PLACE NEXT DATA ON LINES D00-D31
833
WRITE DATA INTO SELECTED DEVICE AND INCREMENT DEVICE ADDRESS
835
WAIT FOR DSO* HIGH TO LOW TRANSITION
839
DRIVE DSO* LOW
841
DRIVE DSO* HIGH
843
WAIT UNTIL DTACK* HIGH TO LOW TRANSITION
845
DRIVE DTACK* LOW
847
DRIVE DTACK* HIGH
845
LATCH DATA FROM LINES D00-D31
849
PLACE NEXT DATA ON LINES D00-D31
TO FIG.-8C
TO FIG.-8C

FIG.-8B

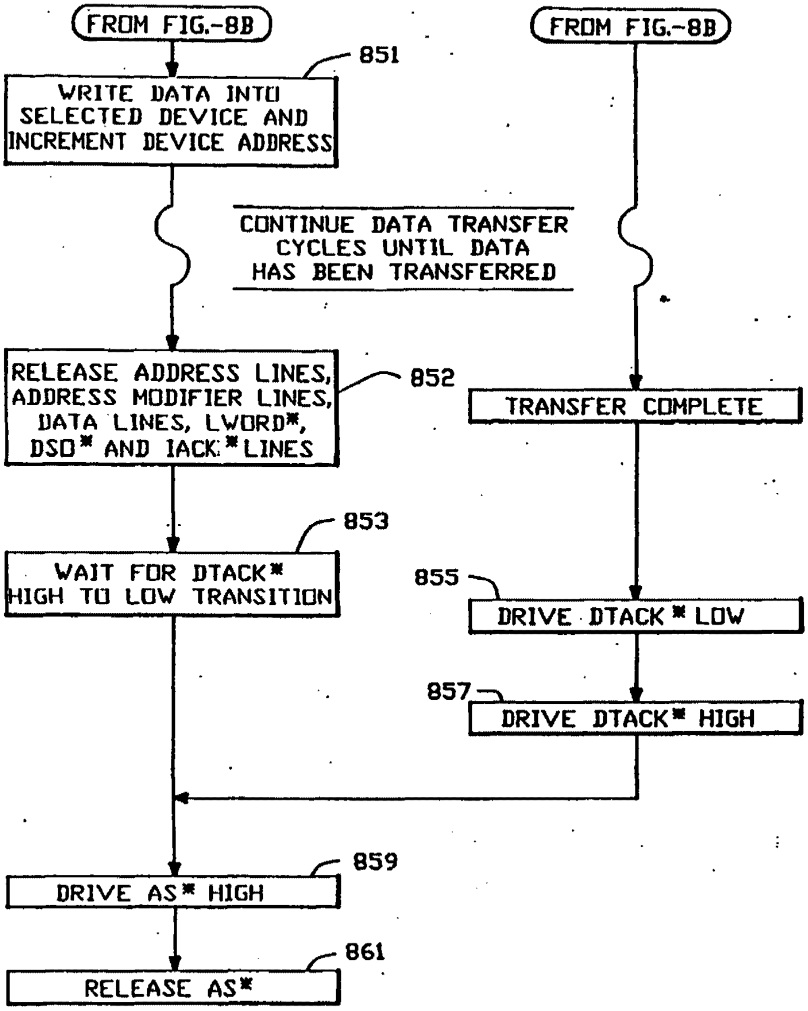

FROM FIG.-8B
FROM FIG.-8B
851
WRITE DATA INTO SELECTED DEVICE AND INCREMENT DEVICE ADDRESS
CONTINUE DATA TRANSFER CYCLES UNTIL DATA HAS BEEN TRANSFERRED
RELEASE ADDRESS LINES, ADDRESS MODIFIER LINES, DATA LINES, LWORD*, DS0* AND IACK* LINES
852
TRANSFER COMPLETE
853
WAIT FOR DTACK* HIGH TO LOW TRANSITION
855
DRIVE DTACK* LOW
857
DRIVE DTACK* HIGH
859
DRIVE AS* HIGH
861
RELEASE AS*

FIG.-8C