

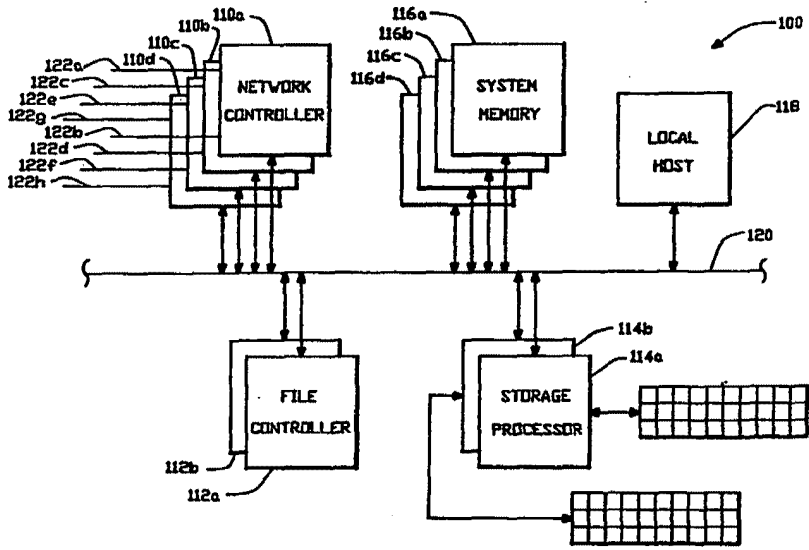


INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification <sup>5</sup> :  G06F 15/16	A1	(11) International Publication Number: <b>WO 91/03788</b>  (43) International Publication Date: 21 March 1991 (21.03.91)
---	----	--

<p>(21) International Application Number: PCT/US90/04711</p> <p>(22) International Filing Date: 20 August 1990 (20.08.90)</p> <p>(30) Priority data: 404,959 8 September 1989 (08.09.89) US</p> <p>(71) Applicant: AUSPEX SYSTEMS, INC. [US/US]; 2952 Bunker Hill Lane, Santa Clara, CA 95054 (US).</p> <p>(72) Inventors: ROW, Edward, John ; 468 Mountain Laurel Court, Mountain View, CA 94064 (US). BOUCHER, Laurence, B. ; 20605 Montalvo Heights Drive, Saratoga, CA 95070 (US). PITTS, William, M. ; 780 Mora Drive, Los Altos, CA 94022 (US). BLIGHTMAN, Stephen, E. ; 775 Salt Lake Drive, San Jose, CA 95133 (US).</p>	<p>(74) Agents: FLIESLER, Martin, C. et al.; Fliesler, Dubb, Meyer &amp; Lovejoy, 4 Embarcadero Center, Suite 400, San Francisco, CA 94111 (US).</p> <p>(81) Designated States: AT (European patent), AU, BE (European patent), CA, CH (European patent), DE (European patent)*, DK (European patent), ES (European patent), FR (European patent), GB (European patent), IT (European patent), JP, KR, LU (European patent), NL (European patent), SE (European patent).</p> <p><b>Published</b> <i>With international search report.</i> <i>Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i></p>
--	---

(54) Title: PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE



(57) Abstract

A file server architecture is disclosed, comprising as separate processors, a network controller unit (110), a file controller unit (112) and a storage processor unit (114). These units incorporate their own processors, and operate in parallel with a local Unix host processor (118). All networks are connected to the network controller unit (110), which performs all protocol processing up through the NFS layer. The virtual file system is implemented in the file controller unit (112) and the storage processor (114) provides high-speed multiplexed access to an array of mass storage devices. The file controller unit (112) controls file information caching through its own local cache buffer, and controls disk data caching through a large system memory which is accessible on a bus by any of the processors.

\* See back of page

### DESIGNATIONS OF "DE"

Until further notice, any designation of "DE" in any international application whose international filing date is prior to October 3, 1990, shall have effect in the territory of the Federal Republic of Germany with the exception of the territory of the former German Democratic Republic.

#### *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.

AT	Austria	ES	Spain	MC	Monaco
AU	Australia	FI	Finland	MG	Madagascar
BB	Barbados	FR	France	ML	Mali
BE	Belgium	GA	Gabon	MR	Mauritania
BF	Burkina Fasso	GB	United Kingdom	MW	Malawi
BG	Bulgaria	GR	Greece	NL	Netherlands
BJ	Benin	HU	Hungary	NO	Norway
BR	Brazil	IT	Italy	PL	Poland
CA	Canada	JP	Japan	RO	Romania
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo	KR	Republic of Korea	SE	Sweden
CH	Switzerland	LJ	Liechtenstein	SN	Senegal
CM	Cameroon	LK	Sri Lanka	SU	Soviet Union
DE	Germany	LJ	Luxembourg	TD	Chad
DK	Denmark			TC	Togo
				US	United States of America

-1-

PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE

5

10           The present application is related to the following  
U.S. Patent Applications, all filed concurrently  
herewith:

15           1.    MULTIPLE    FACILITY    OPERATING    SYSTEM  
ARCHITECTURE, invented by David Hitz, Allan Schwartz,  
James Lau and Guy Harris;

          2.    ENHANCED    VMEBUS    PROTOCOL    UTILIZING  
PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA  
TRANSFER, invented by Daryl Starr; and

20           3.    BUS LOCKING FIFO MULTI-PROCESSOR COMMUNICATIONS  
SYSTEM UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND  
BLOCK MODE DATA TRANSFER invented by Daryl D. Starr,  
William Pitts and Stephen Blightman.

25           The above applications are all assigned to the  
assignee of the present invention and are all  
expressly incorporated herein by reference.

**SUBSTITUTE SHEET**

-2-

BACKGROUND OF THE INVENTIONField of the Invention

5 The invention relates to computer data networks, and more particularly, to network file server architectures for computer networks.

Description of the Related Art

10 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.

15 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.

20 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); 25 AppleTalk for MacIntoshes; and TCP/IP with NFS or RFS

SUBSTITUTE SHEET

-3-

for Unix. These protocols are all well-known in the industry.

5 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; 10 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, 15 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 20 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. 25 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 30 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 35 and now widely adopted by the Unix community. NFS is defined in a document entitled, "NFS: Network File

**SUBSTITUTE SHEET**

-4-

System Protocol Specification," Request For Comments (RFC) 1094, by Sun Microsystems, Inc. (March 1989). This document is incorporated herein by reference in its entirety.

5           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. 10 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. 15

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 20 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 25 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, 30 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 35 local networks together.

**SUBSTITUTE SHEET**

-5-

In addition to performance considerations, the low client-to-server ratio creates computing problems in several additional ways:

5           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.

10           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.

15           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).

20           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 40% greater, than the cost of the workstation alone.

25           The widening I/O gap, as well as administrative and economic considerations, demonstrates a need for higher-performance, larger-capacity Unix file servers.  
30           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  
35           a client. Adding larger disks, more network adaptors,

**SUBSTITUTE SHEET**

-6-

extra primary memory, or even a faster processor do not resolve basic architectural I/O constraints; I/O throughput does not increase sufficiently.

5 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.

10  
15  
20 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.

25  
30  
35 In yet another computer architecture, a plurality of CPUs, each having its own cache memory for data and

**SUBSTITUTE SHEET**



-7-

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 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, among other things, each processor runs the entire operating system. Interprocessor communication is also not optimal for file server applications.

#### 25 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

**SUBSTITUTE SHEET**

-8-

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  
5 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  
10 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  
15 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  
20 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  
25 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.

30 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:

35 Fig. 1. is a block diagram of a prior art file server architecture;

**SUBSTITUTE SHEET**

-9-

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;

5 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;

10 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

15 Figs. 8A-C are a flowchart illustrating the operation of a fast transfer protocol BLOCK READ cycle.

#### DETAILED DESCRIPTION

20 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

25 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

30 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

35 controller 30 connected to a quarter-inch tape drive 32, and possibly a network #2 controller 34 connected

**SUBSTITUTE SHEET**

-10-

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 format 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

**SUBSTITUTE SHEET**

-11-

includes a local network destination address (translated from the internet address) as well as other parameters.

5 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, 10 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 15 incorporated herein by reference.

20 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; 25 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.

30 The prior art file server of Fig. 1 uses both TCP and UDP. It uses UDP for file server-related 35 services, and uses TCP for certain other services

**SUBSTITUTE SHEET**

-12-

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.

**SUBSTITUTE SHEET**

-13-

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

**SUBSTITUTE SHEET**

-14-

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

**SUBSTITUTE SHEET**



-15-

processes are instead merely time sliced. The CPU 10 therefore represents a major bottleneck in the processing of file server requests.

5 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.

10 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

15 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  
20 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  
25 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  
30 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  
35 each support up to three SCSI disk drives each. The tenth SCSI channel on each of the storage processors

SUBSTITUTE SHEET

-16-

114 is used for tape drives and other SCSI peripherals.

5 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  
10 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  
15 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.

20 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  
25 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:  
30 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  
35 the use of a messaging kernel running on each of the

**SUBSTITUTE SHEET**

-17-

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.

5  
10 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. 15 The processors of Fig. 2 have special hardware, discussed below, that can expediently exchange and buffer inter-processor messaging kernel messages.

20 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.

25 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 35 100; the external network protocol supported by the

**SUBSTITUTE SHEET**

-18-

server is precisely standard, licensed NFS. LNFS is described in more detail below.

5 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  
10 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  
15 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.

20 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  
25 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  
30 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  
35 in the Network Controller 110, not in the host 118.

**SUBSTITUTE SHEET**

-19-

The server 100 uses conventional NFS ported from Sun Microsystems, Inc., Mountain View, CA, and is NFS protocol compatible.

5 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.

10  
15  
20

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.

25 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.

30

**SUBSTITUTE SHEET**

-20-

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.

**SUBSTITUTE SHEET**

-21-

5 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,  
10 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  
15 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  
20 devices and system memory 116.

#### Host Processor 118

The local host 118 has three main purposes: to run  
25 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  
30 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  
35 demons operate transparently.

**SUBSTITUTE SHEET**

-22-

The host processor 118 runs the following software modules:

5        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.

10        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.

15        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 20 112. Normally, the host 118 uses LNFS and goes through the file controller.

25        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.

30        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.

35        Yellow Pages and Authentication Demon. The Yellow Pages and Authentication services are Sun-ONC standard

**SUBSTITUTE SHEET**



-23-

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

**SUBSTITUTE SHEET**

-24-

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

**SUBSTITUTE SHEET**

-25-

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  
5 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  
10 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  
15 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.  
20 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.  
25

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  
30 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.  
35 One of the remote boards can copy data directly from

**SUBSTITUTE SHEET**

the CPU memory 214 via the bidirectional buffer 284. LAN memories 236 and 256 are not directly addressed over VME bus 120.

5 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 message passing. Referring to the Appendix, the FIFOs 240 and 260 are programmed with  
10 the following bit settings in the Data Transfer Configuration Register:

<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
0	WD Mode	N/A
1	Parity Chip	N/A
15 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)
20 7	Reset	no (0)

The Data Transfer Control Register is programmed as follows:

<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
0	Enable PortA Req/Ack	yes (1)
25 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)
30 6	Checksum Enable	yes (1)
7	PortA Master	yes (1)

35 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

**SUBSTITUTE SHEET**

-27-

data transfer is disabled by unsetting bits 0 (Enable PortA Req/Ack) and 7 (PortA Master) of the Data Transfer Control Register.

5 The remainder of the control settings in FIFO 270 are the same as those in FIFOs 240 and 260 described above.

10 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.

25 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.

30 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 35 236 on the LAN data bus 232 (or 256 on the LAN data

**SUBSTITUTE SHEET**

-28-

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.

**SUBSTITUTE SHEET**

-29-

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

**SUBSTITUTE SHEET**

-30-

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.

5 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.

20 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

**SUBSTITUTE SHEET**



-31-

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.

5  
10  
15  
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.

20  
25  
30  
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.)

35  
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

**SUBSTITUTE SHEET**

-32-

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

**SUBSTITUTE SHEET**

-33-

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 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

**SUBSTITUTE SHEET**

-34-

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.

5           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  
10       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  
15       NFS: get attributes of a file (GETATTR); set attributes of a file (SETATTR); look up a file (LOOKUP); create a file (CREATE); remove a file (REMOVE); rename a file (RENAME); create a new linked file (LINK); create a symlink (SYMLINK); remove a  
20       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  
25       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.

30           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  
35       LAN DMA controller 242, parity FIFO 240 and VME/FIFO DMA controller 272 to transmit the entire block of

**SUBSTITUTE SHEET**

-35-

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

**SUBSTITUTE SHEET**

-36-

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 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

**SUBSTITUTE SHEET**

-37-

bus, and only the desired number of 32-bit words will be unloaded from port A.

5 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  
10 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.

#### 15 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  
20 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  
25 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  
30 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  
35 as those described above with respect to EPROM 220 and

**SUBSTITUTE SHEET**

-38-

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.

5        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  
10       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.

15       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  
20       390 are the same as those described above with respect to command FIFO 290 on NC 110a.

25       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  
30       116.

30       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  
35       memory 396, however, with the FC 112a acting as an VME bus slave. In such transfers, the remote master

**SUBSTITUTE SHEET**



-39-

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.

5     FILE CONTROLLER OPERATION

10     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.

15     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.

20     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

25  
30  
35

**SUBSTITUTE SHEET**

-40-

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

**SUBSTITUTE SHEET**

-41-

consulting (Santa Clara, California: 1988), incorporated herein by reference.

5 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.

10 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.

20 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.

25 30 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,

**SUBSTITUTE SHEET**

-42-

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.

5       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.

10       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.

15       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

35       In the file server 100, each of the storage processors 114 can interface the VME bus 120 with up

**SUBSTITUTE SHEET**

-43-

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., which is  
 5 incorporated herein by reference. The eight  
 general-purpose I/O bits provided by MFP 524 are  
 configured according to the following table:

	<u>Bit</u>	<u>Direction</u>	<u>Definition</u>
10	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.
15	5	input	Channel Operation Done - A composite of the channel done bits from all 13 channels of the DMA controller, described below.
20	4	output	DMA Controller Enable. Enables the DMA Controller to run.
25	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.
30	1	output	External Interrupts Disable. Disables externally generated interrupts to the microprocessor 510.
35	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  
 40 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-  
 45 only register as seen by microprocessor 510. If the

**SUBSTITUTE SHEET**

-45-

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 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.

**SUBSTITUTE SHEET**

-46-

5 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.

10  
15 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.

20  
25  
30  
35 The parity FIFOs 544 are each configured in their Adaptec 6250 mode. Referring to the Appendix, the

**SUBSTITUTE SHEET**



FIFOs 544 are programmed with the following bit settings in the Data Transfer Configuration Register:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
	0	WD Mode	(0)
5	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)
10	6	Checksum Carry Wrap	(0)
	7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
15	0	Enable PortA Req/Ack	(1)
	1	Enable PortB Req/Ack	(1)
	2	Data Transfer Direction	as desired
	3	CPU parity enable	(0)
20	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

**SUBSTITUTE SHEET**

-48-

the following bit settings in the Data Transfer Configuration Register:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
	0	WD Mode	(0)
5	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)
10	6	Checksum Carry Wrap	(0)
	7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
15	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)
20	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.

25 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.

30 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  
35 the control bus 516. The channel enable register 560

**SUBSTITUTE SHEET**

-49-

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 communication 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.

**SUBSTITUTE SHEET**

-50-

15 This is a null channel for which neither a ready status nor done status is generated.

5 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.

10 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.

20 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.

30 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

**SUBSTITUTE SHEET**

-51-

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

**SUBSTITUTE SHEET**

-52-

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.

5 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  
10 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  
20 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.

30 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.

35 The SP 114a also includes a series of registers 592, similar to the registers 282 on NC 110a (Fig. 3)

**SUBSTITUTE SHEET**

-53-

and the registers 382 on FC 112a (Fig. 4). The details of these registers are not important for an understanding of the present invention.

5        STORAGE PROCESSOR OPERATION

10        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  
15        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  
20        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  
25        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  
30        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  
35        parallel operation is transparent to the caller client.

**SUBSTITUTE SHEET**

-54-

5 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.

10 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
- 20 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
- 25 09 Start/Stop A SCSI Drive
- 0C Inquiry
- 0E Read Message Log Buffer
- 0F Set SP 114a Interrupt

30 The above transactions are described in detail in the above-identified application entitled MULTIPLE FACILITY OPERATING SYSTEM ARCHITECTURE. For and 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.

35

**SUBSTITUTE SHEET**



-55-

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,

**SUBSTITUTE SHEET**

-56-

recovery actions similar to data check recovery take place.

#### Read/Write Cache Pages

5           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  
10          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-  
15          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  
20          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  
25          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  
30          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:

**SUBSTITUTE SHEET**

	<u>OP CODE</u>	<u>OPERATION</u>	
	0	NO-OP	
5	1	ZEROES -> BUFFER	Move zeros from zeros register 576 to local data buffer 564.
10	2	ZEROES -> FIFO	Move zeros from zeros register 576 to the currently selected FIFO on common data bus 550.
15	3	ZEROES -> VMEbus	Move zeros from zeros register 576 out onto the VME bus 120. Used for initializing cache buffers in system memory 116.
20	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.
25			
30			
35	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.
40			
45			
50	6	VMEbus -> BUFFER & FIFO	Target data is moved from VME bus 120 to a SCSI

**SUBSTITUTE SHEET**

-58-

5			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.
10	7	BUFFER -> VMEbus	This operation is not used.
15	8	BUFFER -> FIFO	Participating data is transferred to create redundant data or recovered data on a disk drive. Used only in RAID 5 applications.
20	9	FIFO -> VMEbus	This operation is used to move target data directly from a disk drive onto the VME bus 120.
25	A	FIFO -> BUFFER	Used to move participating data for recovery and modify operations. Used only in RAID 5 applications.
30	B	FIFO -> VMEbus & BUFFER	This operation is used to save target data for participation in data recovery. Used only in RAID 5 applications.
35			

SYSTEM MEMORY

40 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

45

**SUBSTITUTE SHEET**

-59-

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.

#### 15 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.

#### 30 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

**SUBSTITUTE SHEET**

-60-

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.

5 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.

10 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.

15 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.

20 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:

25  
30  
35  
**SUBSTITUTE SHEET**

-61-

	<u>Type</u>	<u>Description</u>
	0	Pointer to a new message being sent
	1	Pointer to a reply message
	2	Pointer to message to be forwarded
5	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

**SUBSTITUTE SHEET**

-62-

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

**SUBSTITUTE SHEET**



-63-

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  
5 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.

10 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.

**SUBSTITUTE SHEET**

-64-

APPENDIX AVME/FIFO DMA Controller

5 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  
10 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.

15 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  
20 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  
25 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  
30 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  
35 false condition allows execution from the pipeline  
address while a true condition causes execution from

**SUBSTITUTE SHEET**

-65-

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.

5            ALU. The alu comprises a single IDT49C402A integrated circuit. It is 16 bits in width and most  
10           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  
15           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  
20           below along with mnemonics and a description of each function.

25	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.
30	CIN           38	(Carry INput) This bit forces the carry input to the alu.
35	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.
40	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.

**SUBSTITUTE SHEET**

5 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.

10 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.

	<u>LFS&lt;2:0&gt;</u>	<u>SELECTED FLAG</u>
20	0	This value selects a null flag.
25	1	When set this bit enables the buffer clock. When reset this bit disables the buffer clock.
30	2	When this bit is cleared VME bus transfers, buffer operations and RAS are all disabled.
	3	NOT USED
35	4	When set this bit enables VME bus transfers.
	5	When set this bit enables buffer operations.
40	6	When set this bit asserts the row address strobe to the dram buffer.
45	7	When set this bit selects page 0 of the control store.

50 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.

-67-

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

**SUBSTITUTE SHEET**

-68-

loads the data from the alu bus  
into the dram address counters.

5	3	<p>D&lt;14:7&gt; -&gt; mux addr&lt;8:0&gt; WR_VADL loads the data from the alu bus into the least significant 2 bytes of the VME address register. D&lt;15:2&gt; -&gt; VME addr&lt;15:2&gt; D1 -&gt; ENB_tional registers D&lt;15:2&gt; -&gt; VME addr&lt;15:2&gt; D1 -&gt; ENB_ENH D0 -&gt; ENB_BLK</p>
10		
15	4	<p>WR_VADH loads the most significant 2 bytes of the VME address register. D&lt;15:0&gt; -&gt; VME addr&lt;31:16&gt;</p>
20		
25	5	<p>WR_RADD loads the dual ported ram address counters. D&lt;10:0&gt; -&gt; ram addr &lt;10:0&gt;</p>
30	6	<p>WR_WCNT loads the word counters. D15 -&gt; count enable* D&lt;14:8&gt; -&gt; count &lt;6:0&gt;</p>
35	7	<p>WR_CO loads the co-channel select register. D&lt;7:4&gt; -&gt; CO&lt;3:0&gt;</p>
40	8	<p>WR_NXT loads the next-channel select register. D&lt;3:0&gt; -&gt; NEXT&lt;3:0&gt;</p>
45	9	<p>WR_CUR loads the current-channel select register. D&lt;3:0&gt; -&gt; CURR &lt;3:0&gt;</p>
50	10:14	RESERVED - NOT DEFINED
	15	<p>JUMP causes the control store sequencer to select the alu data bus. D&lt;8:0&gt; -&gt; CS_A&lt;8:0&gt;</p>

SUBSTITUTE SHEET

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.

5

TEST<3:0> Condition

10	0	FALSE	-always false
	1	TRUE	-always true
	2	ALU_COUT	-carry output of alu
	3	ALU_EQ	-equals output of alu
15	4	ALU_OVR	-alu overflow
	5	ALU_NEG	-alu negative
20	6	XFR_DONE	-transfer complete
	7	PAR_ERR	-buffer parity error
	8	TIMOUT	-bus operation timeout
25	9	ANY_ERR	-any error status
	14:10	RESERVED	-NOT DEFINED
30	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.

35 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  
 40 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  
 45 next location.

**SUBSTITUTE SHEET**

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.

5 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
10	0	CURR POINTER 0   STATUS 0
	4	INITIAL POINTER 0
15		
	58	CURR POINTER B   STATUS B
	5C	INITIAL POINTER B
20	60	not used   not used
	64	not used   not used
25	68	CURR POINTER D   STATUS D
	6C	INITIAL POINTER D
	70	not used   STATUS E
30	74	HIGH SPEED BUS ADDRESS 31:2 0 0
	78	PARAMETER BLOCK 0
35		
	??	PARAMETER BLOCK n

40 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  
 45 initialized to 0x0000 by the microprocessor 510 before enabling the channel. Upon detecting a value of 0x0000

**SUBSTITUTE SHEET**



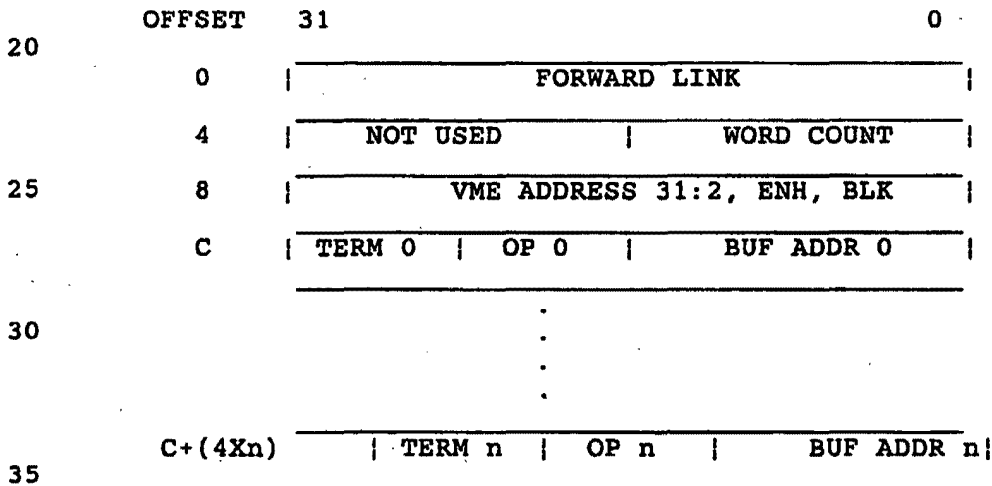
in the current block pointer the DMA controller 580 will copy the lower 16 bits from the initial pointer to the current pointer. Once the DMA controller 580 has completed the specified operations for the parameter block the current pointer will be updated to point to the next block. In the event that 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
- 15 2 BUS OPERATION TIMEOUT
- 3 BUS OPERATION ERROR
- 4 DATA PATH PARITY ERROR

The format of the parameter block is:



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:

**SUBSTITUTE SHEET**

-72-

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.

5

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:

10

|D15|D14|D13|D12|D11|D10|D9|D8|D7|D6|D5|D4|D3|D2|D1|D0|

15

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.

20

VME ADDRESS - The VME address specifies the starting address for data transfers. Thirty bits allows the address to start at any quad byte boundary.

25

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.

30

35

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

**SUBSTITUTE SHEET**

-73-

when performing any transfer to or from 16 bit address space.

5           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.

10           |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|0|T|

15           OP - Operations are specified by the op code. The op code byte has the following format:

25           |0|0|0|0|OP3|OP2|OP1|OP0|

The op codes are listed below ("FIFO" refers to any of the FIFOs 544 or 554):

**SUBSTITUTE SHEET**

	<u>OP_CODE</u>	<u>OPERATION</u>
	0	NO-OP
	1	ZEROES -> BUFFER
	2	ZEROES -> FIFO
5	3	ZEROES -> VMEbus
	4	VMEbus -> BUFFER
	5	VMEbus -> FIFO
	6	VMEbus -> BUFFER & FIFO
	7	BUFFER -> VMEbus
10	8	BUFFER -> FIFO
	9	FIFO -> VMEbus
	A	FIFO -> BUFFER
	B	FIFO -> VMEbus & BUFFER
	C	RESERVED
15	D	RESERVED
	E	RESERVED
	F	RESERVED

**SUBSTITUTE SHEET**

-75-

APPENDIX BEnhanced VME Block Transfer Protocol

5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 of large amounts of data across a VME backplane bus by  
substantially increasing the data transfer rate.

**SUBSTITUTE SHEET**

-76-

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.

5       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,  
10       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  
15       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  
20       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  
25       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  
30       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  
35       transfer cycle, the master will assert and deassert DSO\* in the form of a pulse of a given duration. The

**SUBSTITUTE SHEET**

-77-

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

**SUBSTITUTE SHEET**

-78-

modules must be prepared to capture data at any time, as DTACK\* is not referenced during the transfer cycle.

5 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  
10 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  
15 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  
20 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  
25 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.

30 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  
35 low 705. The slave modules receive the AS\* low signal 707. Each slave individually determines whether it

**SUBSTITUTE SHEET**



-79-

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.

5 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.

10 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.

15 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.

20 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.

**SUBSTITUTE SHEET**

-80-

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

**SUBSTITUTE SHEET**

-81-

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.

5       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 10 807. 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 15 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. 20 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 25 818, 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 30 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 35 to low, respectively 825 and 827, the master latches the data being transmitted by the slave over data

**SUBSTITUTE SHEET**

-82-

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.

5       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.

10       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.

15       The slave drives DTACK\* low, 845, and, after a predetermined period of time, drives DTACK\* high, 847.

20       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.

25       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

30

35

**SUBSTITUTE SHEET**

-83-

receipt of the DTACK\* high to low transition, the master will drive AS\* high 859 and release the AS\* line 861.

5 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  
10 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  
15 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.

**SUBSTITUTE SHEET**

-84-

APPENDIX C  
Parity FIFO

5       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  
10       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  
15       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  
20       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  
25       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  
30       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  
35       A REQ input specifies that the master requires a data

**SUBSTITUTE SHEET**

-85-

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.

5        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.

10       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

15       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.

20       Clk

The clock input is used to generate some of the chip timing. It is expected to be in the 10-20 Mhz range.

25       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  
30       conjunction with Port A Ack.

**SUBSTITUTE SHEET**

-86-

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, Port B 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,

**SUBSTITUTE SHEET**



-87-

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

**SUBSTITUTE SHEET**

-88-

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  
5 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.

10 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  
15 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  
20 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.

25 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  
30 Parity Sync are true the data is driven onto the Port B data lines and a parallel parity check for zero is not performed.

35 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

**SUBSTITUTE SHEET**

5 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:

10	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
15	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
20	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
25	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
30	16	True	True	uProc byte 1	Port A byte 1	uProc byte 0	Port A byte 0

35 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.

40 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

**SUBSTITUTE SHEET**

-90-

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.

5 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 10 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. 15 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 20 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

##### 25 Data Transfer Configuration Register (Read/Write)

Register Address 0. This register is cleared by the reset signal.

- 30 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.
- 35 Bit 2 Parity Correct Mode. Set if the parity chip is to correct parallel parity on Port B.

**SUBSTITUTE SHEET**

-91-

- 5 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.
- 10 Bit 4 Invert Port A byte address 0. Set to invert the least significant bit of Port A byte address.
- 15 Bit 5 Invert Port A byte address 1. Set to invert the most significant bit of Port A byte address.
- 20 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.
- 25 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.
- 30 Data Transfer Control Register (Read/Write)  
 Register Address 1. This register is cleared by the reset signal or by writing to the reset bit.
- 35 Bit 0 Enable Data Transfer on Port A. Set to enable the Port A Req/Ack handshake.
- 40 Bit 1 Enable Data Transfer on Port B. Set to enable the Port B Req/Ack handshake.
- 45 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.

SUBSTITUTE SHEET

-92-

- 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.
- 5
- 10
- 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.
- 15
- 20
- 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.
- 25
- 30
- 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.
- 35
- 40
- 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.

45

- 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.

SUBSTITUTE SHEET

- 5 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.
- 10 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.
- 15 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.
- 20
- 25 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)

30 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.

- 35 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.
- 40 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.
- 45 Bit 2 Port A to RAM Y. Set if Port A is accessing RAM Y, and reset if it is accessing RAM X.

**SUBSTITUTE SHEET**

-94-

- Bit 3 Port B to RAM Y. Set if Port B is accessing RAM Y, and reset if it is accessing RAM X .
- 5 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.

15 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

25 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

35 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

**SUBSTITUTE SHEET**



-95-

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  
5 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  
10 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)

15 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,  
20 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  
25 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  
30 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  
35 Port A and the Port B handshakes enabled, and in this case the data transfer itself should be done without

**SUBSTITUTE SHEET**

-96-

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)

```

**SUBSTITUTE SHEET**

```

/*****
    The following routine initiates a Fifo data transfer using two
    values passed to it.
5       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
10                      automatically using both the Port A and Port B
                    handshakes, both data transfer enables bits should be
                    set in this parameter.
*****/

FIFO_initiate_data_transfer(config_data, control_data)
15  unsigned char config_data, control_data;
    {
        FIFO1->config = config_data | FIFO_RESET;    /* Set
        Configuration value & Reset */
        FIFO1->control = control_data & (~FIFO_PORT_ENABLES); /* Set
20                      everything but enables */
        FIFO1->control = control_data;                /* Set data transfer
        enables */
    }

25  /*****
    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.
30  *****/

FIFO_force_odd_length_transfer()
    {
        FIFO1->control &= ~FIFO_PORT_ENABLES; /* Disable Ports A & B
35  */
        if (PORT_A_TO_PORT_B(FIFO1)) {
            if (PORT_A_TO_RAM_Y(FIFO1)) {
                FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL; /*
40  Set RAM Y full */
            }
            else FIFO1->ram_X_addr = FIFO_FORCE_RAM_FULL; /* Set
        RAM X full */
            FIFO1->control |= FIFO_PORT_B_ENABLE;    /*
45  Re-Enable Port B */
        }
        else {
            if (PORT_B_TO_RAM_Y(FIFO1)) {
                FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL; /*
50  Set RAM Y full */
            }
            else FIFO1->ram_X_addr = FIFO_FORCE_RAM_FULL; /* Set
        RAM X full */
    }

```

**SUBSTITUTE SHEET**

-98-

```

        FIFO1->control |= FIFO_PORT_A_ENABLE; /*
Re-Enable Port A */
    }
}

5  /*****
    The following routine returns how many odd bytes have been
left in the Fifo at the end of a data transfer.
*****/

10 int FIFO_count_odd_bytes()
    {
        int number_odd_bytes;
        number_odd_bytes=0;
        if (FIFO1->status & FIFO_DATA_IN_RAM) {
15         if (PORT_A_TO_PORT_B(FIFO1)) {
                number_odd_bytes =
(PORT_A_BYTE_ADDRESS(FIFO1));
                if (PORT_A_TO_RAM_Y(FIFO1))
20                 number_odd_bytes += (FIFO1->ram_Y_addr) *
4;
                else number_odd_bytes += (FIFO1->ram_X_addr) * 4;
            }
            else {
25                 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);
30    }

    /*****
    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
35 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.

40     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
45     Bit 6 = data register failed
        Bit 7 = checksum register failed
    *****/

50 #define RAM_DEPTH 64 /* number of long words in Fifo Ram */
    reg_expected_data[6] = { 0x7F, 0xFF, 0x00, 0x1F, 0x3F, 0x3F };

```

SUBSTITUTE SHEET

-99-

```

char FIFO_uprocessor_interface_test()
{
    unsigned long test_data;
    char *register_addr;
    5   int i;
        char j,error;
        FIFO1->config = FIFO_RESET;          /* reset the chip */
        error=0;
        register_addr =(char *) FIFO1;
    10   j=1;

        /* first test registers 0 thru 5 */

        for (i=0; i<6; i++) {
    15   *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 */
    20   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;
    25   }

        /* now test Ram data & checksum registers
        test 1s throughout Ram & then test 0s */

    30   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
    35   */
            FIFO1->data = test_data;
            FIFO1->control = 0;
            for (i=0;i<RAM_DEPTH;i++)
                if (FIFO1->data != test_data) error |= j; /* read &
    40   check data */
            if (FIFO1->checksum) error |= 0x80; /* checksum
        should = 0 */
        }

    45   /* 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 |
    50   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 */

```

SUBSTITUTE SHEET

- 100 -

```

    test_data += 0x04040404;
}
test_data=0x00010203;          /* address pattern start */
FIFO1->control = FIFO_CHECKSUM_ENABLE;
5   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 |= ; /* read & check
address pattern */
10    test_data += 0x04040404;
    }
    if (FIFO1->checksum != 0x0102) error |= 0x80; /* test checksum of
address pattern */
FIFO1->config = FIFO_RESET | FIFO_16_BITS ; /* inhibit carry wrap
15 */
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 */
20 return (error);
}

```

SUBSTITUTE SHEET

-101-

CLAIMS

What is claimed is:

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; and

means in said interface processor unit for satisfying requests from said network to store data from said network on said mass storage device, for satisfying 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 coupleable to said network;

a data control unit coupleable to said mass storage device;

a buffer memory; and

means:

for transmitting to said data control unit requests from said network to store specified storage data from said network on 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 requests from said network to retrieve specified

**SUBSTITUTE SHEET**

-102-

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 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,

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 file system level 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 in said storage processor unit for transmitting data between said buffer memory and said mass storage device.

**SUBSTITUTE SHEET**



-103-

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; and

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,

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,

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 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. A data control unit for use with a data network, a mass storage device and a buffer memory, and in response to file system level storage and retrieval requests from said data network, comprising:

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,

**SUBSTITUTE SHEET**

-104-

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 file system level retrieval requests 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 in said storage processor unit for transmitting data between said buffer memory and said mass storage device.

6. A data control unit for use with a data network and a mass storage device, and in response to file system level storage and retrieval requests from said data network, comprising:

a data bus;

a buffer memory bank coupled to said bus;

storage processor apparatus coupled to said bus and coupleable to said mass storage device;

file processor apparatus coupled to said bus, said file processor apparatus including a local memory bank;

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 and for translating said file system level retrieval requests into requests to retrieve data from specified physical retrieval locations in said mass storage device, said means including means for caching file control information

**SUBSTITUTE SHEET**

-105-

through said local memory bank in said file processor unit; and

means for caching the file data, to be stored or retrieved according to said storage and retrieval requests, through said buffer memory bank.

7. A network node for use with a data network and a mass storage device, comprising:

a system buffer memory;

a host processor unit having direct memory access to said system buffer memory;

a network control unit coupleable to said network and having direct memory access to said system buffer memory;

a data control unit coupleable to said mass storage device and having direct memory access to said system buffer memory;

means for satisfying requests from said network to store data from said network on said mass storage device, for satisfying 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 means comprising means

for transmitting from said network control unit to said system memory bank by direct memory access file data from said network for storage on said mass storage device,

for transmitting from said system memory bank to said data control unit by direct memory access said file data from said network for storage on said mass storage device,

for transmitting from said data control unit to said system memory bank by direct memory access file data for retrieval from said mass storage device to said network,

**SUBSTITUTE SHEET**

-106-

and for transmitting from said system memory bank to said network control unit said file data for retrieval from said mass storage device to said network;

at least said network control unit including a microprocessor and local instruction storage means distinct from said system buffer memory, all instructions for said microprocessor residing in said local instruction storage means.

8. A network file server for use with a data network and a mass storage device, comprising:

a host processor unit running a Unix operating system;

an interface processor unit coupleable to said network and to said mass storage device, said interface processor unit including means for decoding all NFS requests from said network, for performing all procedures for satisfying said NFS requests, for encoding any NFS reply messages for return transmission on said network, and for transmitting predefined non-NFS categories of messages from said network to said host processor unit for processing in said host processor unit.

**SUBSTITUTE SHEET**

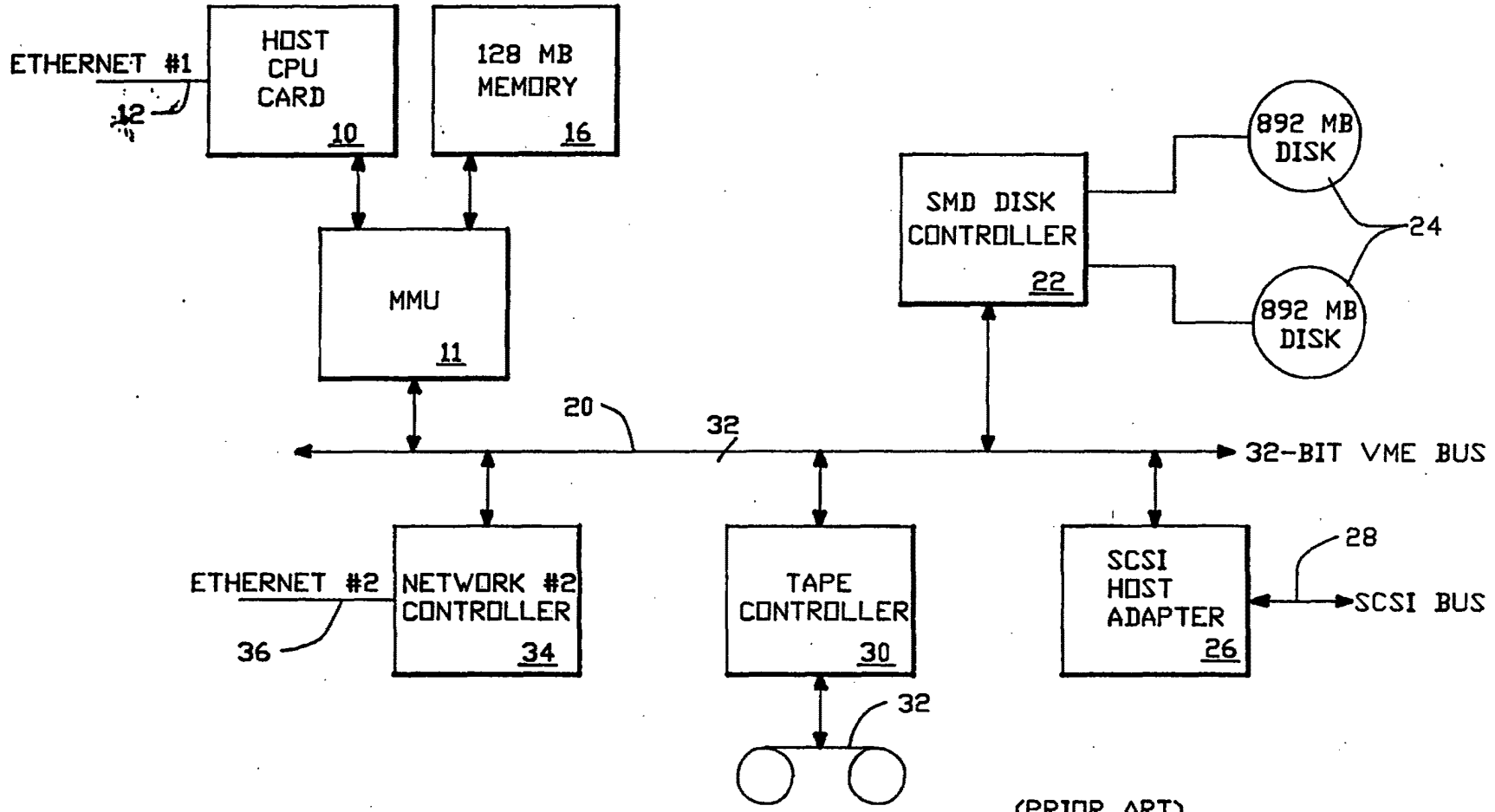


FIG.-1

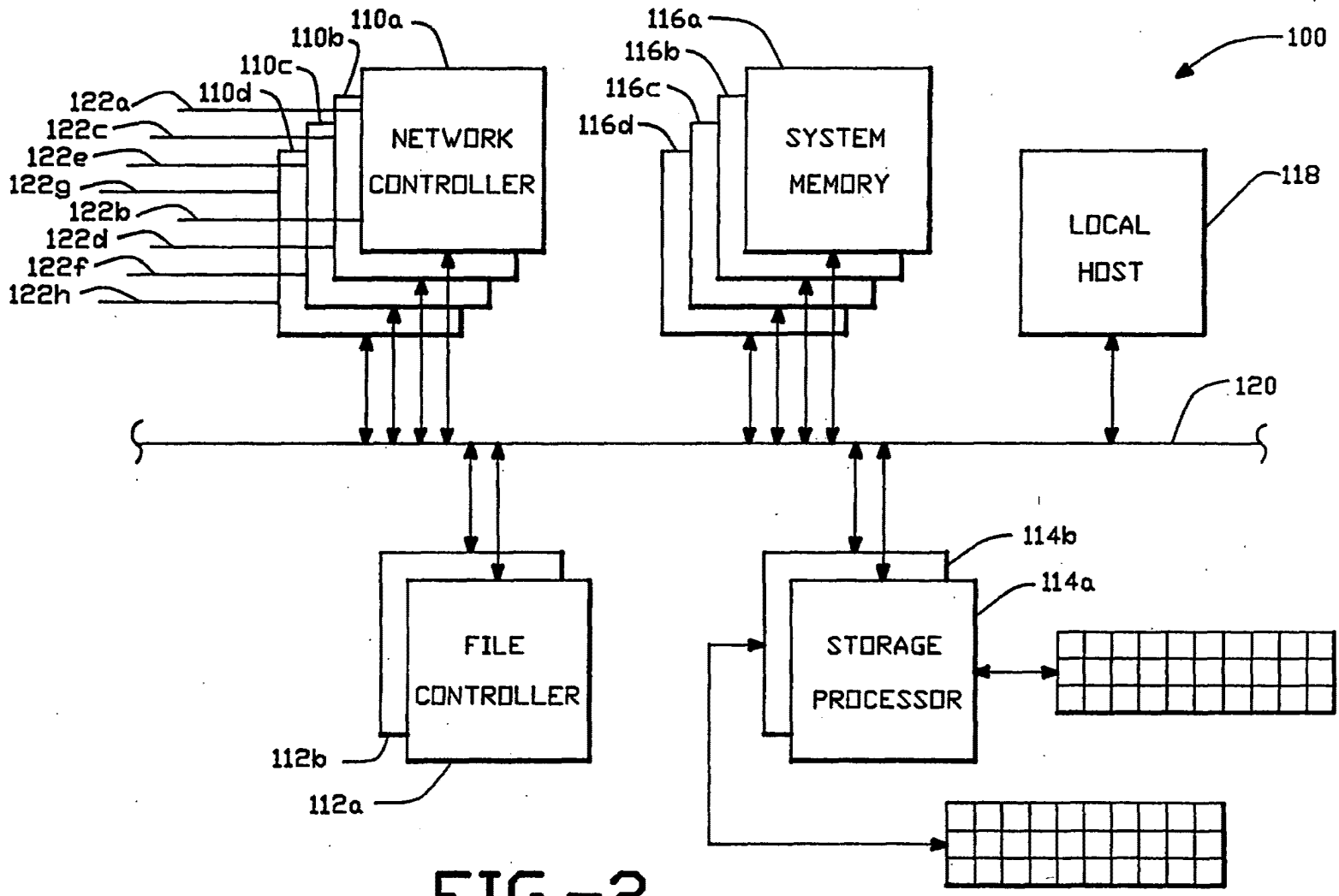


FIG.-2

SUBSTITUTE SHEET

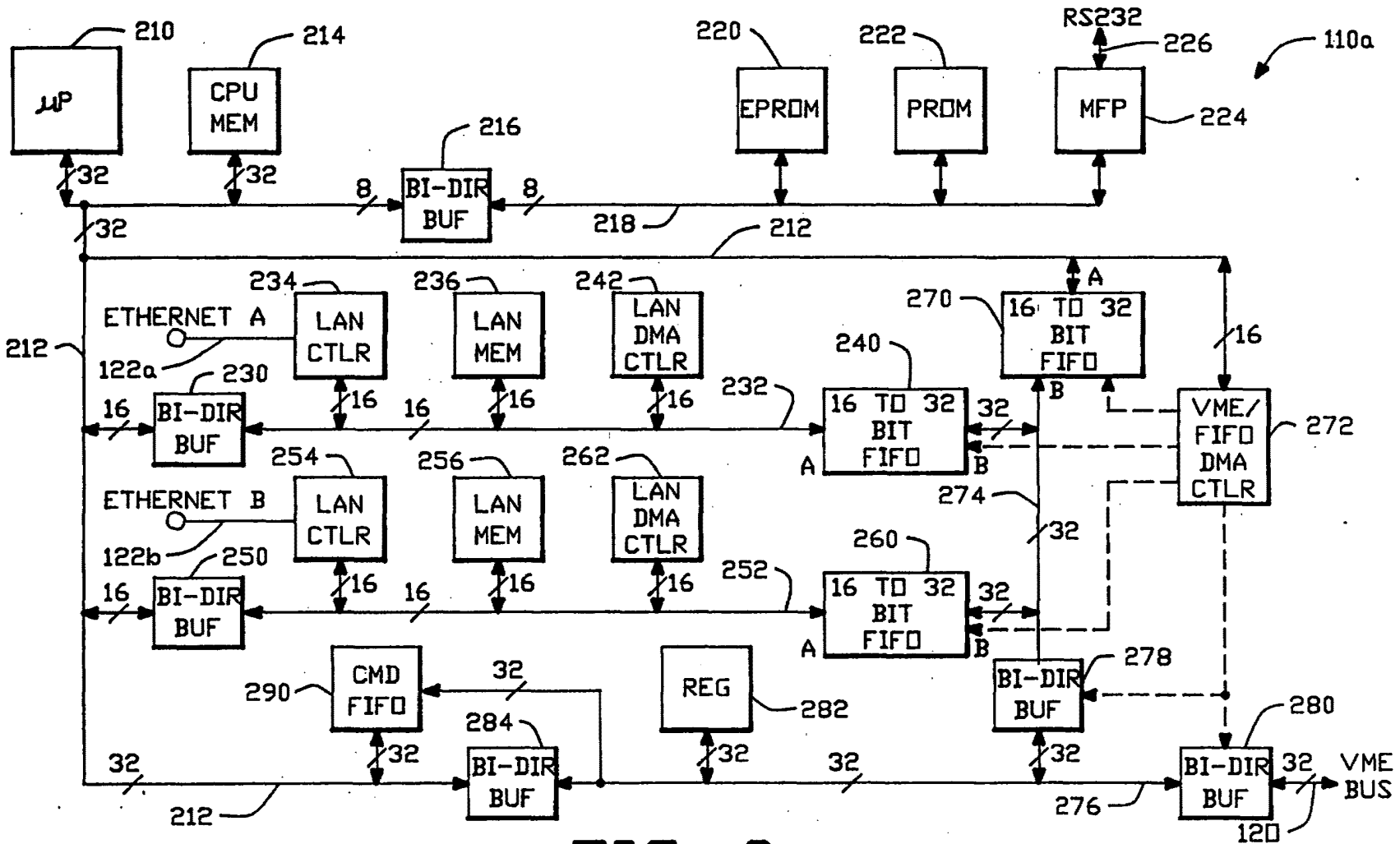


FIG.-3 (NETWORK CONTROLLER)

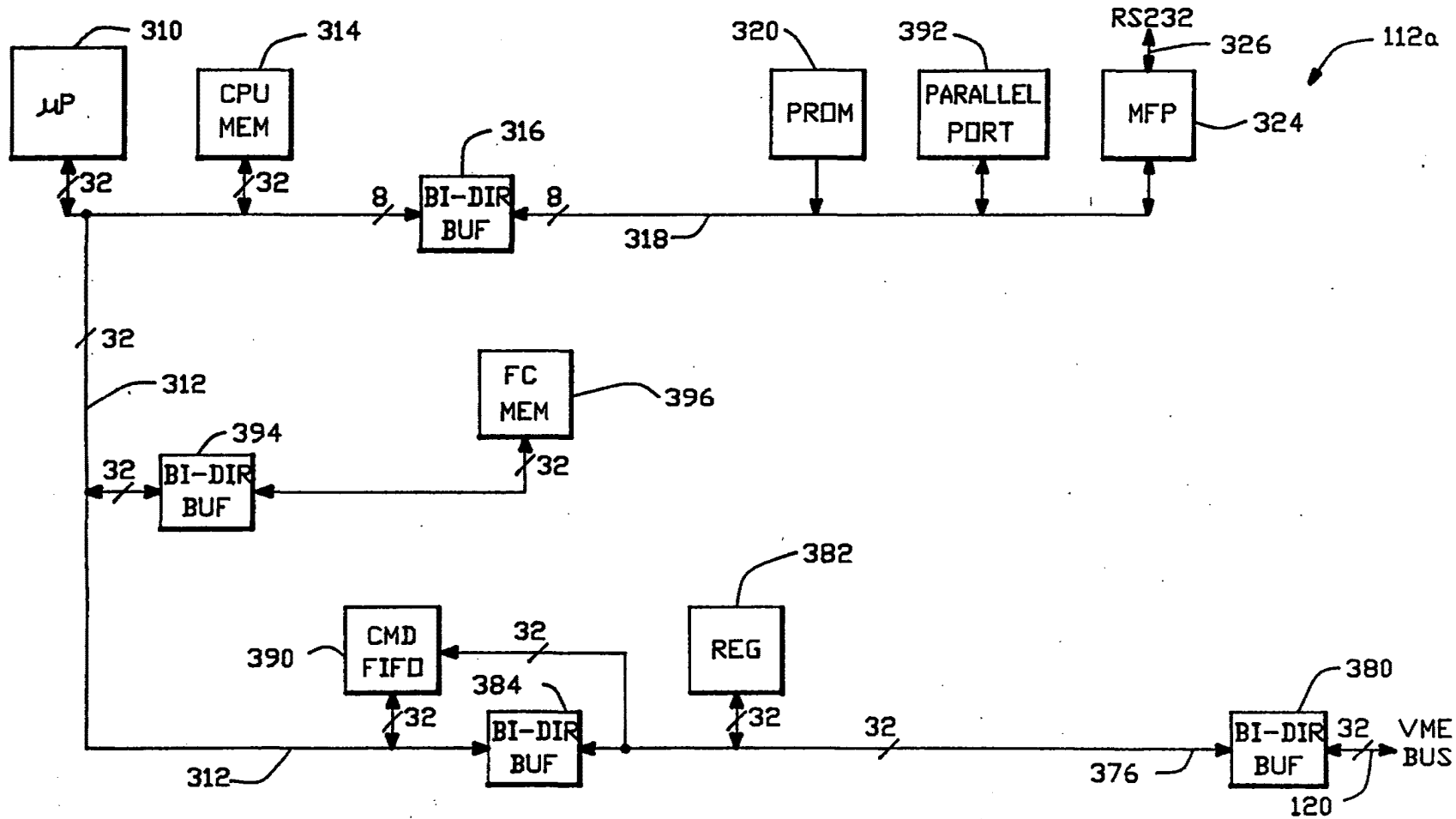


FIG.-4 (FILE CONTROLLER)



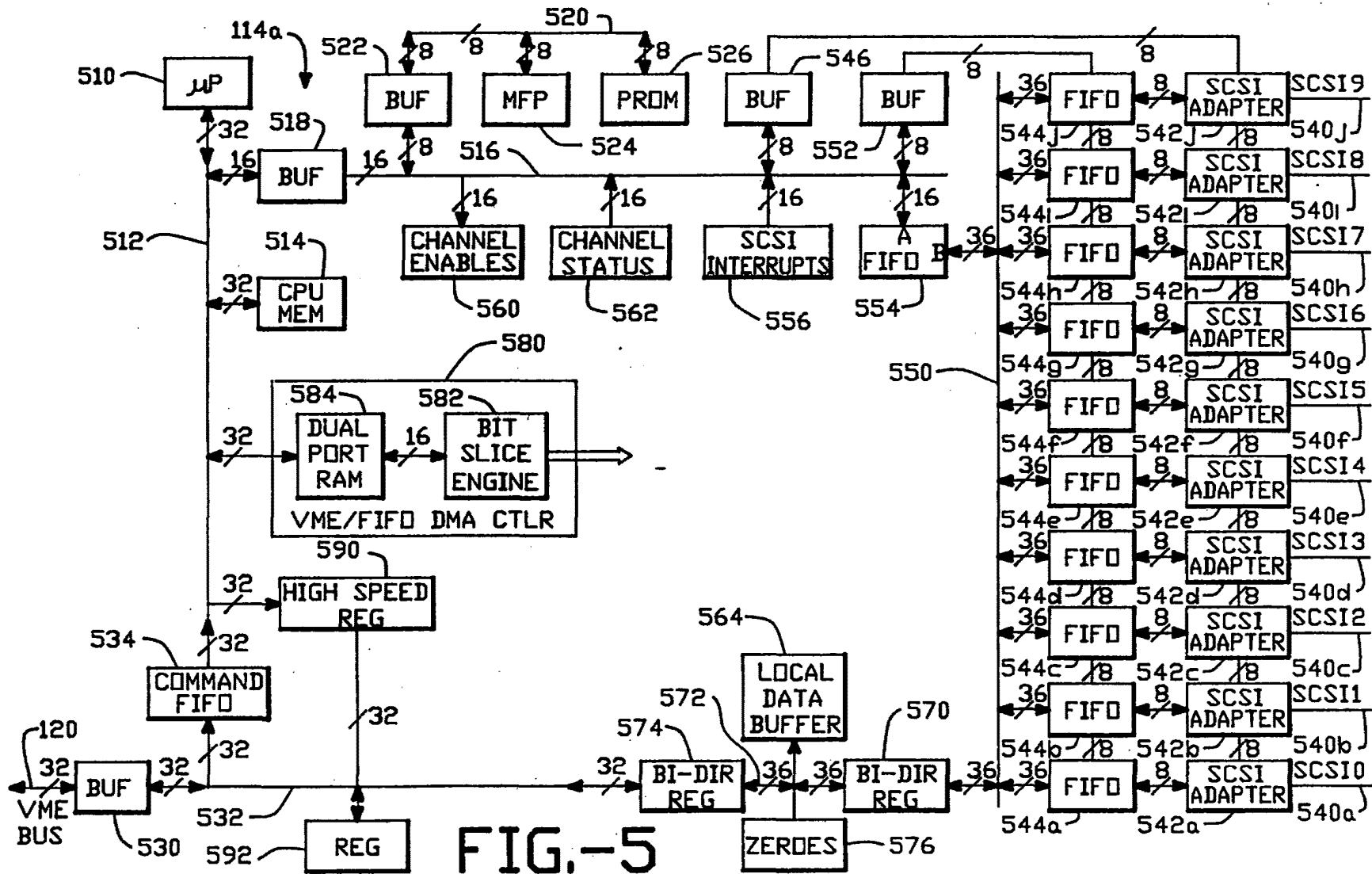


FIG.-5

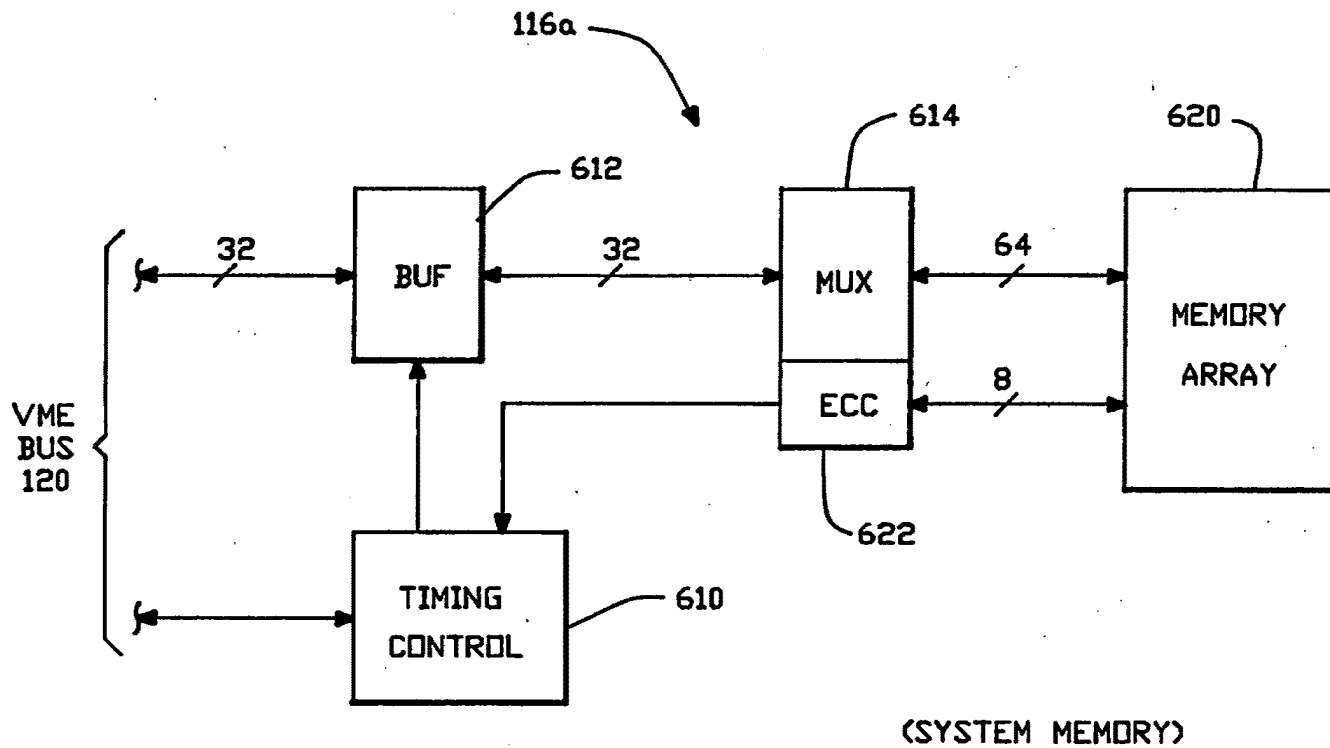


FIG.-6

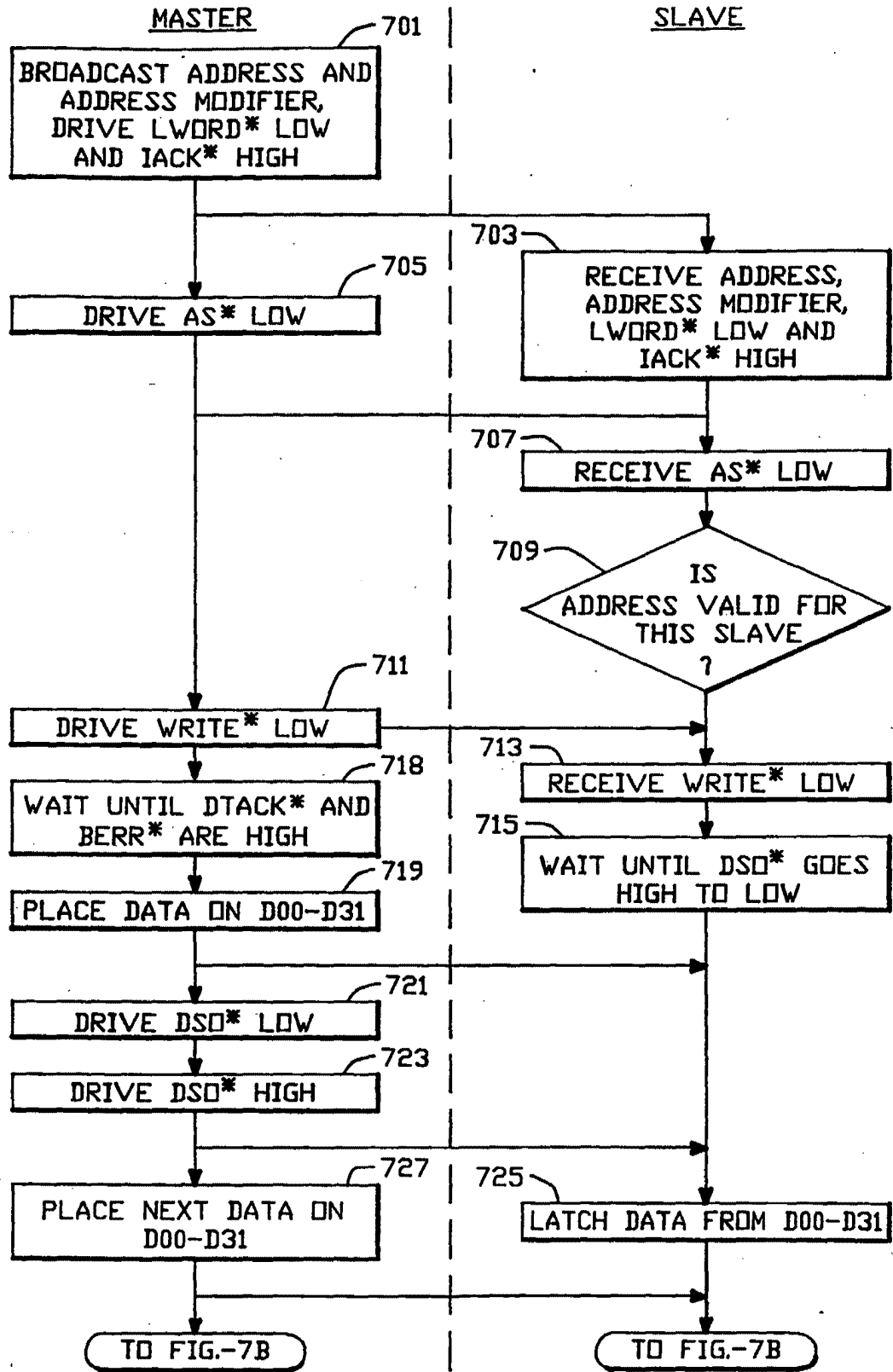


FIG.-7A

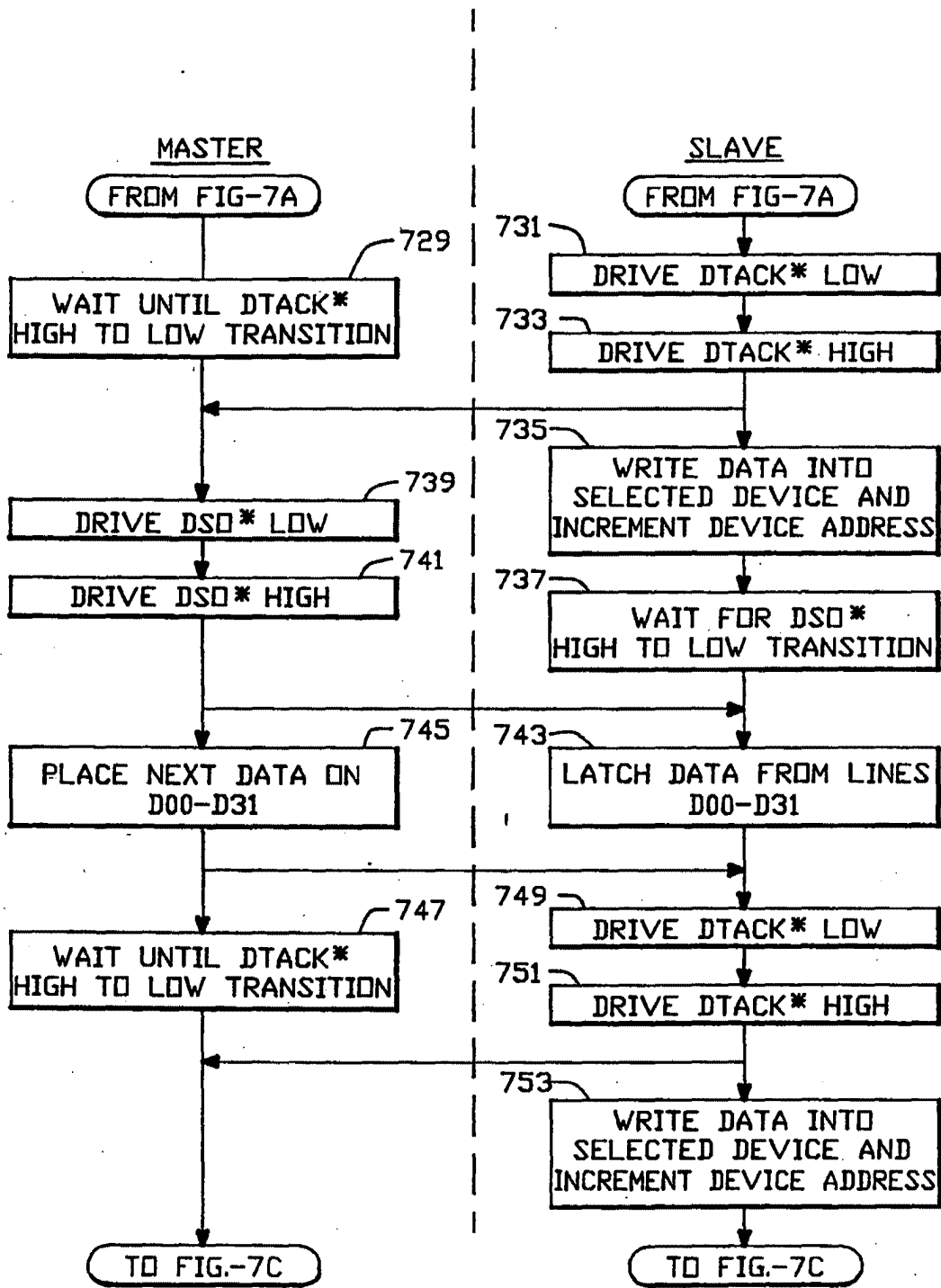


FIG.-7B

SUBSTITUTE SHEET

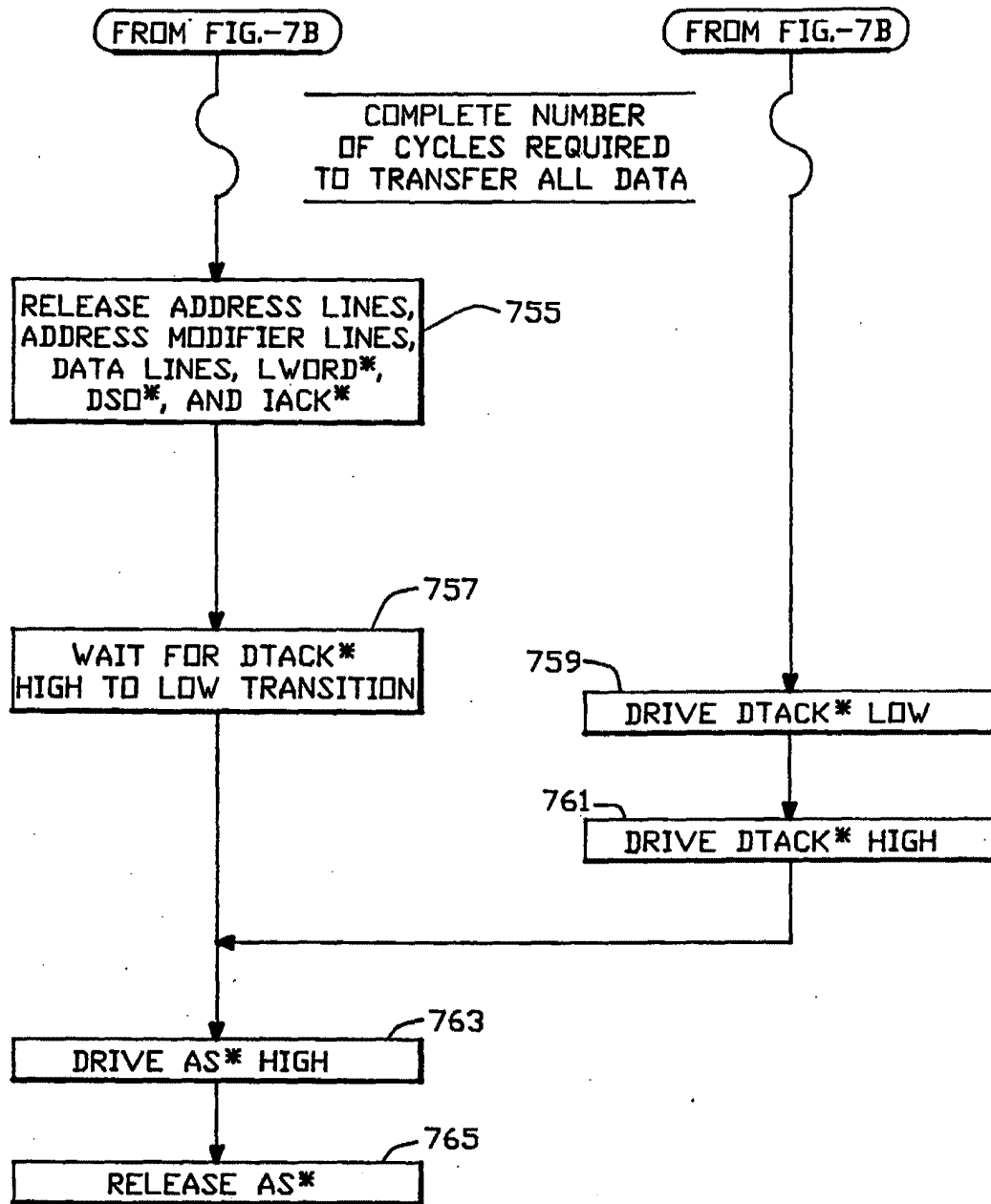


FIG.-7C

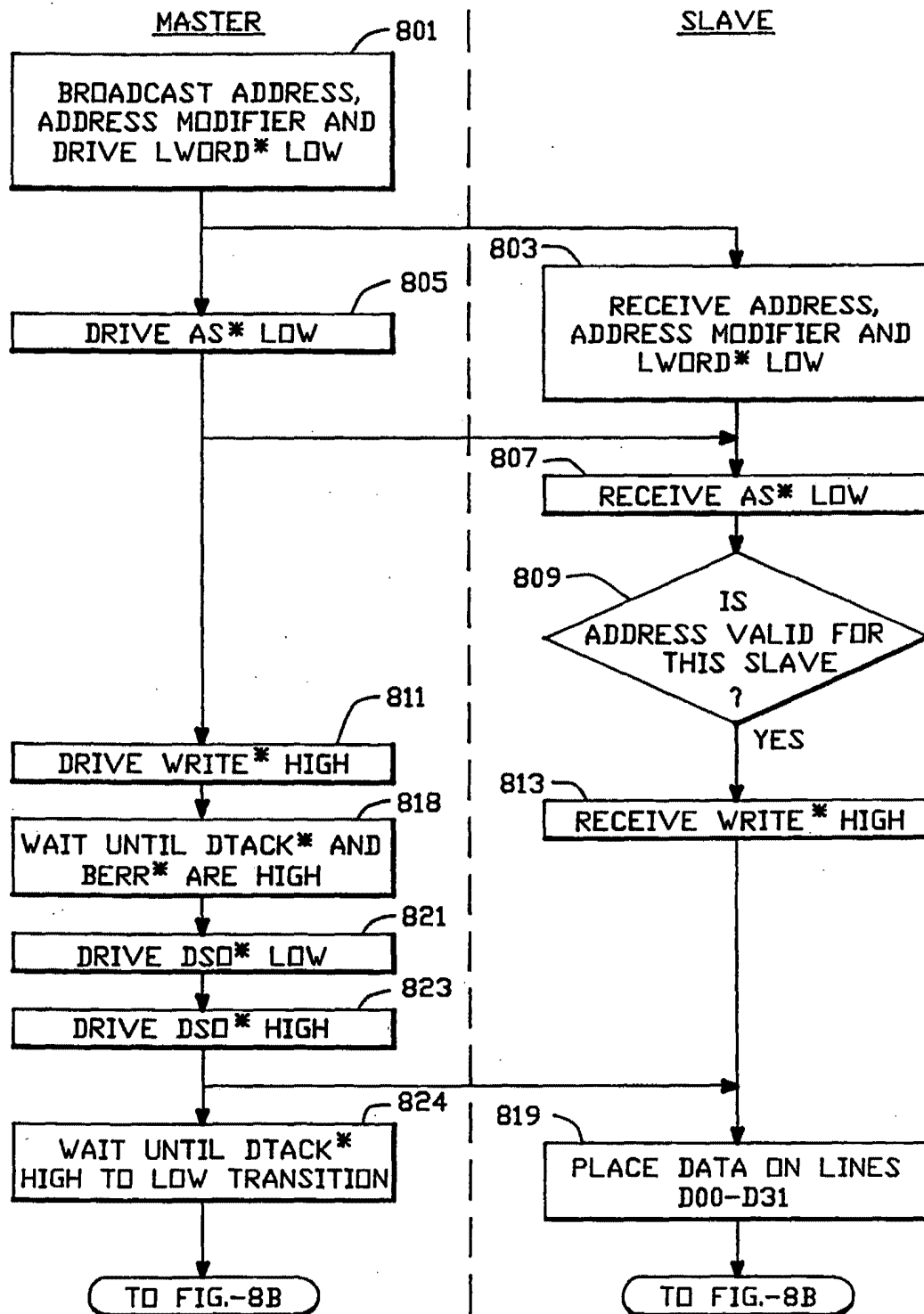


FIG.-8A

SUBSTITUTE SHEET

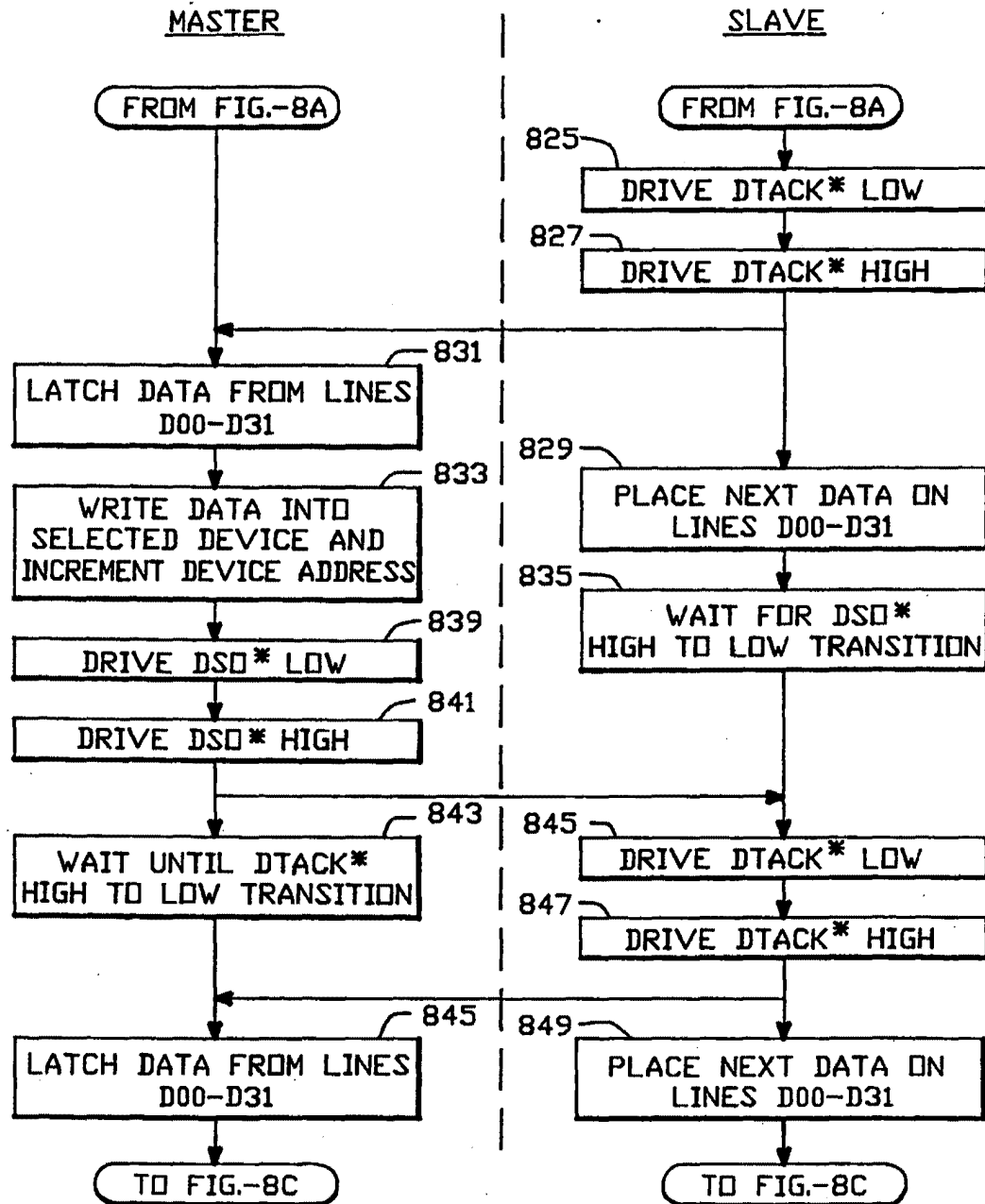


FIG.-8B

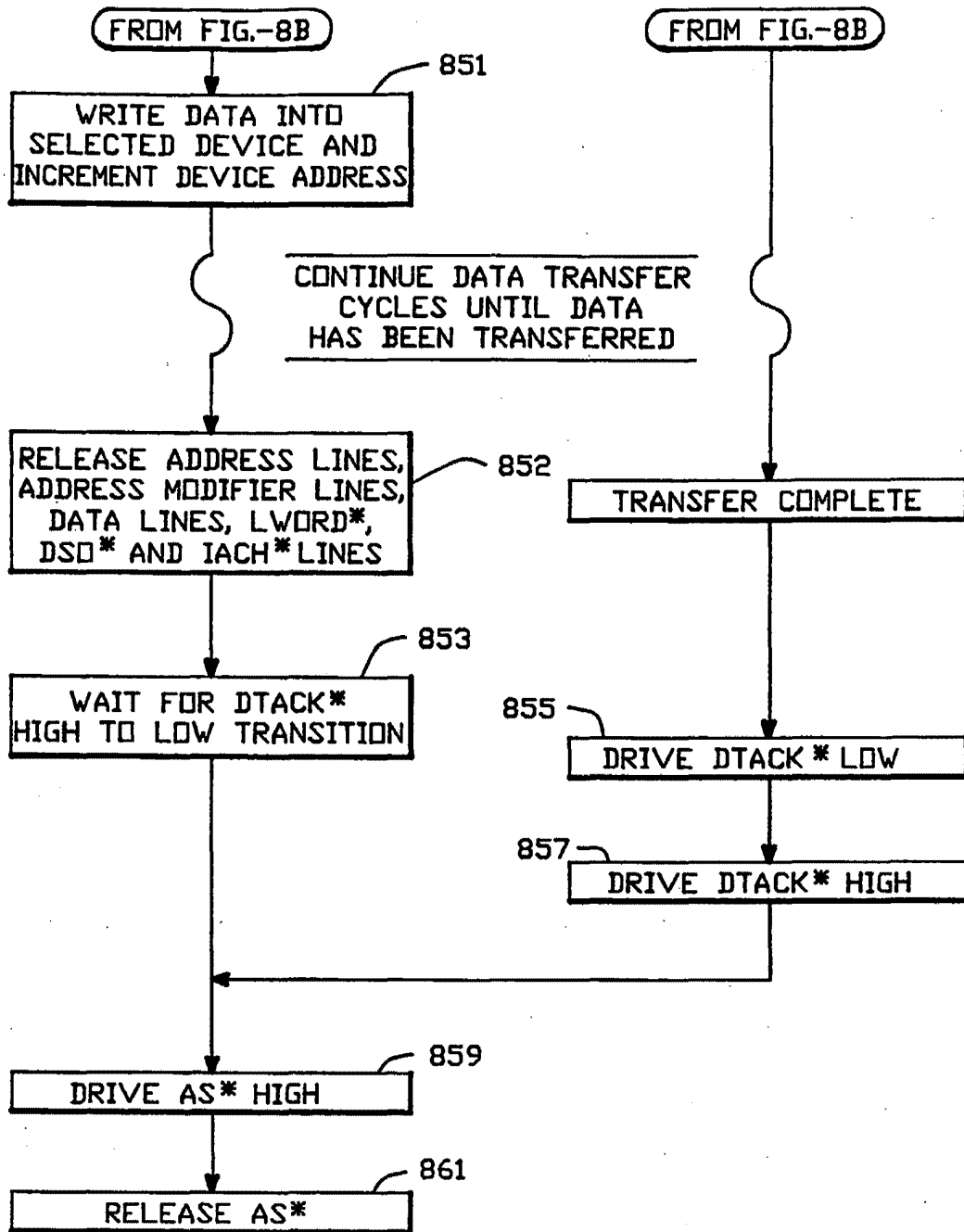


FIG.-8C



# INTERNATIONAL SEARCH REPORT

International Application No. **PCT/US90/04711**

<b>I. CLASSIFICATION OF SUBJECT MATTER</b> (if several classification symbols apply, indicate all) *		
According to International Patent Classification (IPC) or to both National Classification and IPC		
IPC (5) : G06F 15/16		
U.S. Cl : 364/200		
<b>II. FIELDS SEARCHED</b>		
Minimum Documentation Searched *		
Classification System	Classification Symbols	
U.S.	364/200, 900	
Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched *		
<b>III. DOCUMENTS CONSIDERED TO BE RELEVANT</b> 14		
Category *	Citation of Document, 16 with indication, where appropriate, of the relevant passages 17	Relevant to Claim No. 18
Y P	US,A 4,897,781 (CHANG) 30 January 1990 See the entire document.	1-8
Y P	US,A 4,887,204 (JOHNSON) 12 December 1989 See the entire document.	1-8
Y	US,A 4,819,159 (SHIPLEY) 04 April 1989 See the entire document.	1-8
Y	US,A 4,710,868 (COCKE) 01 December 1987 See the entire document.	1-8
<p>* Special categories of cited documents: 15</p> <p>"A" document defining the general state of the art which is not considered to be of particular relevance</p> <p>"E" earlier document but published on or after the international filing date</p> <p>"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)</p> <p>"O" document referring to an oral disclosure, use, exhibition or other means</p> <p>"P" document published prior to the international filing date but later than the priority date claimed</p> <p>"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</p> <p>"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step</p> <p>"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.</p> <p>"&amp;" document member of the same patent family</p>		
<b>IV. CERTIFICATION</b>		
Date of the Actual Completion of the International Search 1		Date of Mailing of this International Search Report 2
30 NOVEMBER 1990		24 JAN 1991
International Searching Authority 3		Signature of Authorized Officer 20
ISA/US		G. D. SHAW

B15

**Bibliographic Fields**

**Document Identity**

(19)【発行国】

日本国特許庁(JP)

【公報種別】

再公表特許(A1)

(11)【国際公開番号】

WO97/33227

【発行日】

平成10年(1998)8月4日

(19) [Publication Office]

Japan Patent Office (JP)

[Kind of Document]

Japanese Republished Patent Publication (A1)

(11) [International Publication Number]

WO 97/33227

[Publication Date]

1998 (1998) August 4\*

**International Filing**

(11)【国際公開番号】

WO97/33227

(21)【国際出願番号】

PCT/JP97/00655

(22)【国際出願日】

平成9年(1997)3月4日

(43)【国際公開日】

平成9年(1997)9月12日

(81)【指定国】

JP US

(11) [International Publication Number]

WO 97/33227

(21) [International Application Number]

PCT /JP97/00655

(22) [International Application Date]

1997 (1997) March 4\*

(43) [International Publication Date]

1997 (1997) September 12\*

(81) [Designated States]

JP US

**Technical**

(54)【発明の名称】

高速一括ファイル転送方法及び装置並びに転送方法を実行するためのプログラムを記憶した記憶媒体

(51)【国際特許分類第6版】

G06F 13/00

【全頁数】

67

(54) [Title of Invention]

STORAGE MEDIA WHICH HIGH SPEED COLLECTIVE FILE TRANSFER METHOD AND THE PROGRAM IN ORDER TO EXECUTE DEVICE AND TRANSFER METHOD STORAGE IS DONE

(51) [International Patent Classification, 6th Edition]

G06F13/00

[Number of Pages in Document]

67

**Filing**

【審査請求】

未請求

【予備審査請求】

未請求

[Request for Examination]

Unrequested

[Provisional Request for Examination]

Unrequested

## 【出願番号】

特願平9-531658

## (22)【国際出願日】

平成9年(1997)3月4日

**Foreign Priority**

## (31)【優先権主張番号】

特願平8-50510

## (32)【優先日】

平8(1996)3月7日

## (33)【優先権主張国】

日本(JP)

## (31)【優先権主張番号】

特願平8-50511

## (32)【優先日】

平8(1996)3月7日

## (33)【優先権主張国】

日本(JP)

## (31)【優先権主張番号】

特願平8-164883

## (32)【優先日】

平8(1996)6月25日

## (33)【優先権主張国】

日本(JP)

**Parties****Applicants**

## (71)【出願人】

## 【氏名又は名称】

日本電信電話株式会社

## 【住所又は居所】

東京都新宿区西新宿3丁目19番2号

**Inventors**

## (72)【発明者】

## 【氏名】

小野田 哲也

## [Domestic Application Number]

Japan Patent Application Hei 9- 531658

## (22) [International Application Date]

1997 (1997) March 4\*

## (31) [Priority Application Number]

Japan Patent Application Hei 8- 50510

## (32) [Priority Date]

1996 (1996) March 7\*

## (33) [Priority Country]

Japan (JP)

## (31) [Priority Application Number]

Japan Patent Application Hei 8- 50511

## (32) [Priority Date]

1996 (1996) March 7\*

## (33) [Priority Country]

Japan (JP)

## (31) [Priority Application Number]

Japan Patent Application Hei 8- 164883

## (32) [Priority Date]

1996 (1996) June 25\*

## (33) [Priority Country]

Japan (JP)

## (71) [Applicant]

## [Name]

NIPPON TELEGRAPH & AMP; TELEPHONE CORP.  
(NTT) (DB 69-062-6718)

## [Address]

Tokyo Shinjuku-ku Nishishinjuku 3-Chome 19\*2\*

## (72) [Inventor]

## [Name]

Onoda Tetsuya

## 【住所又は居所】

神奈川県横須賀市グリーンハイツ2-5-402

## [Address]

Kanagawa Prefecture Yokosuka City Green Heights 2- 5- 402

## (72)【発明者】

## (72) [Inventor]

## 【氏名】

## [Name]

吉川 太郎

Yoshikawa Taro

## 【住所又は居所】

## [Address]

神奈川県横浜市磯子区杉田9-2-4-202

Kanagawa Prefecture Yokohama City Isogo-ku Sugita 9- 2- 4- 202

## (72)【発明者】

## (72) [Inventor]

## 【氏名】

## [Name]

小田部 悟士

Oda \*Satoru \*

## 【住所又は居所】

## [Address]

神奈川県横浜市磯子区杉田9-2-12-C-412

Kanagawa Prefecture Yokohama City Isogo-ku Sugita 9- 2- 12- C- 412

## Agents

## (74)【代理人】

## (74) [Attorney(s) Representing All Applicants]

## 【弁理士】

## [Patent Attorney]

## 【氏名又は名称】

## [Name]

志賀 正武

Shiga Masatake

## Abstract

## (57)【要約】

## (57) [Abstract ]

本発明による高速一括ファイル転送方法は、データ転送を行うために、第1の記憶媒体及びその第1の記憶媒体より入出力が速い第2の記憶媒体を用いて、ファイル転送元で通信リンクを設定する前に第1の記憶媒体内のファイルデータに対して、圧縮等の処理を行いながら、そのファイルデータを第2の記憶媒体へ転送し、ファイルデータに対する処理の完了後、通信リンクを設定し、第2の記憶媒体内のファイルデータを、データに対して処理を施さずにネットワークカードへ一括転送する手順と、ファイル転送先でネットワークカードに伝送されたファイルデータをデータに対して処理を施さずに第2の記憶媒体へ一括転送し、解凍等の処理を行いながら第1の記憶媒体へ転送する手段を有している。

high speed collective file transfer method before setting communication link in file transfer origin making use of second storage media where input-output is faster than first storage media and its first storage media , doing compression or other treatment vis-a-vis file data inside first storage media , while in order to do data transfer ,to transfer file data to second storage media with this invention , after completing treatment for file data , It sets communication link , without administering treatment file data inside second storage media , vis-a-vis data , without administering treatment, with protocol and file forwarding destination which it lumps together transfers to network card file data which transmission is done vis-a-vis data in network card while lumping together transferring to second storage media , treating thawing or other it has possessed means which it transfers to first storage media .

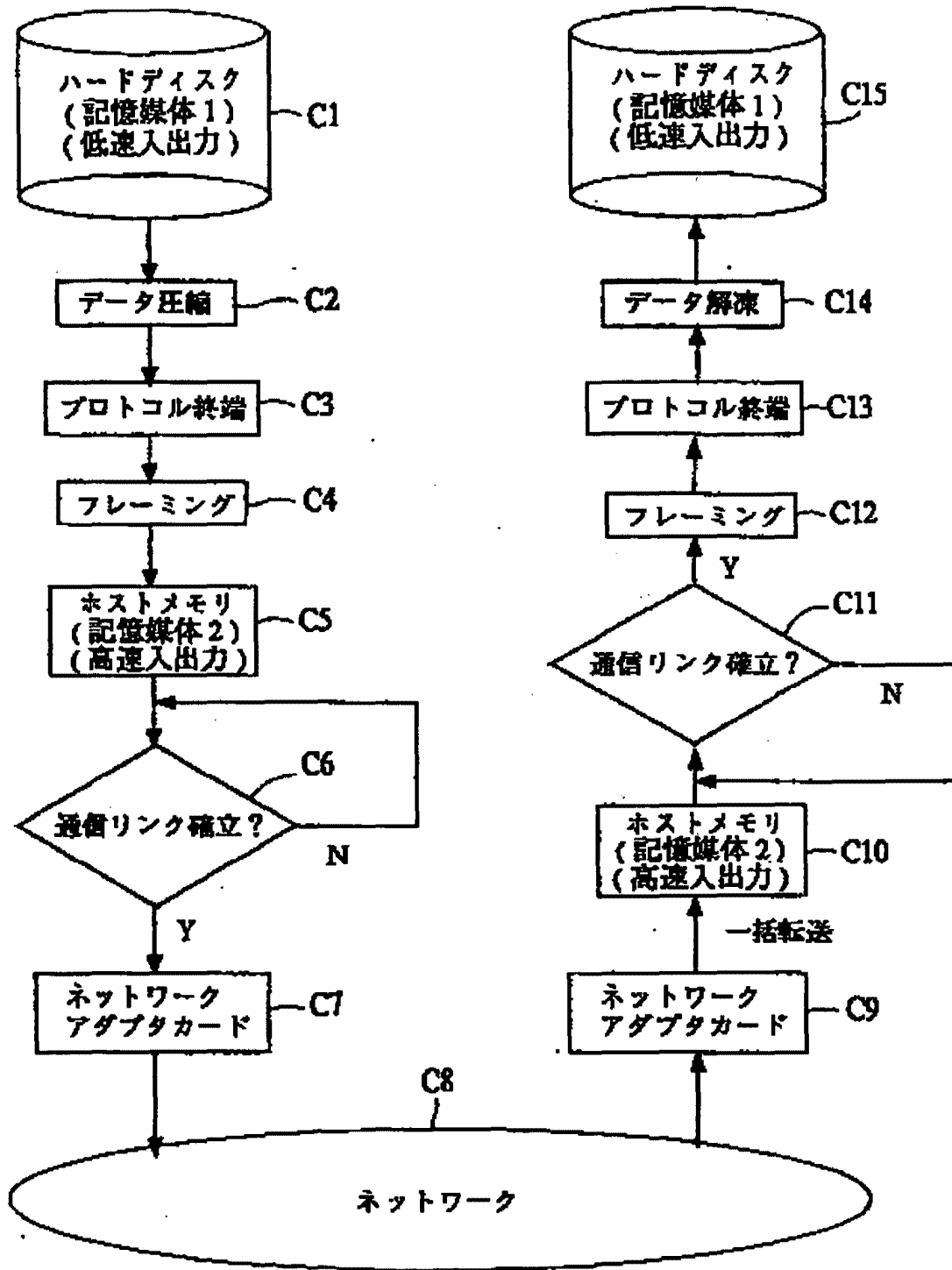


図 6

## Claims

【特許請求の範囲】

[Claim (s)]

1. ファイルの転送元及びファイルの転送先でそれぞれ、データ転送を行うた

Respectively, data transfer is done with transfer origin of 1.file and forwarding destination of file , it is

めの汎用データベース、第1の記憶媒体、及び該第1の記憶媒体より入出力速度が速い第2の記憶媒体から成る汎用計算機アーキテクチャを用いて、ファイルの転送元で、通信リンクを設定する前に、第1の記憶媒体内のファイルデータに対して、少なくとも圧縮、プロトコル終端、及びフレーミングの中のいずれか1つの処理を行いながら、該ファイルデータを汎用データベースを介して第2の記憶媒体へ順次転送する手順と、前記ファイルの転送元で、前記ファイルデータに対する処理の完了後、通信リンクを設定し、前記第2の記憶媒体内のファイルデータを、それに対して処理を施さずに、前記汎用データベースを介して直接、該汎用データベースに接続されている計算機通信用のネットワークアダプタカードへ一括転送し、該ネットワークアダプタカードからネットワークへ伝送する手順と、ファイルの転送先で、前記ネットワークから該転送先の汎用データベースに接続されているネットワークアダプタカードへ伝送された前記ファイルデータを、データの解凍、プロトコル終端、及びフレーミングを含むいずれの処理も施さずに

\*Making use of general purpose computer architecture which consists of second storage media where input-output speed is faster than general purpose data bus , first storage media , and said first storage media , while in transfer origin of file , before setting communication link , at least treating any one in midst of compression, protocol terminal , and flaming vis-a-vis file data inside first storage media , protocol which through general purpose data bus , sequential transfers said file data to second storage media and, With forwarding destination of protocol and file where in transfer origin of aforementioned file , after completing treatment for the aforementioned file data , it sets communication link , without administering treatment vis-a-vis that , through aforementioned general purpose data bus , to the network adapter card for computer communication which directly , is connected to said general purpose data bus it lumpstogether transfer file data inside aforementioned second storage media , from the said network adapter card to network transmission they do , Aforementioned file data which transmission is done , without administering thawing , protocol terminal , of data or each treatment which includes flaming to network adapter card which from aforementioned network is connected to general purpose data bus of said forwarding destination

、該汎用データベースを介して第2の記憶媒体へ一括転送し、前記通信リンクを解放する手順と、前記ファイルの転送先で、前記通信リンクを解放した後、前記第2の記憶媒体

Through said general purpose data bus , with forwarding destination of protocol and aforementioned file which it lumps together transfers to second storage media , release the aforementioned communication link , after releasing aforementioned communication link , the aforementioned second storage media

内のデータに対して、少なくとも解凍処理、通信処理の中のいずれか1つの処理を行いながら、順次処理された前記第2の記憶媒体内のデータを前記汎用データベースを介して第1の記憶媒体へ転送する手順とを有することを特徴とするファイル転送方法。2. 請求項1記載のファイル転送方法において、前記転送元で、前記ファイルデータを前記ネットワークアダプタカードから前記ネットワークへ伝送する場合に、

Vis-a-vis data inside , while treating any one in midst of thawing treatment and communication treatment at least , sequential data inside aforementioned second storage media which was treated through the aforementioned general purpose data bus , regarding to file transfer method which it states in file transfer method . 2.Claim 1 which designates that it possesses protocol which it transfers to first storage media as feature , in aforementioned transfer origin , When aforementioned file data from aforementioned network adapter card transmission it does to aforementioned network ,

該ファイルデータを1個または複数個の packets 単位で転送し、かつ

To transfer said file data with packet unit of one or a plurality , at same time

該パケットの転送先を指定するネットワークアドレスとして、前記転送元の第2の記憶媒体内で該パケットに対応するデータが格納されている場所に対応するメモリアドレスを用いることを特徴とするファイル転送方法。

file transfer method . which designates that it uses memory address which corresponds to site where data which corresponds to the said packet inside aforementioned transfer original second storage media as network address appointing forwarding destination of said packet , is housed as feature

3.

3.

請求項2記載の転送方法において、	
Regarding to transfer method which is stated in Claim 2,	
ネットワーク層のプロトコルとして インターネットプロトコル(IP)を用	
As protocol of network layer Internet protocol (IP ) business	
い	
It is	
IPヘッダのオプション領域に、前記メモリア	ドレスを付与すること
In option region of IP header , description above [ memoria ]	Grant dress
を特徴とするファイル転送方法。	
file transfer method . which is made feature	
4. 請求項1記載のファイル転送方法において、前記転送元で、前記ファイルデータを前記ネットワークアダプタカードから前記ネットワークへ伝送する場合に、該ファイルデータを1個または複数個のパケット単位で転送し、かつ該パケットの転送先を指定するネットワークアドレスとして、ネットワーク層	
Regarding to file transfer method which is stated in 4.Claim 1, whenin aforementioned transfer origin, aforementioned file data from theaforementioned network adapter card transmission it does to aforementioned network , as network address which transfers said file data with packet unit of one or a plurality , atsame time appoints forwarding destination of said packet , network layer	
の論理的アドレス、前記転送元の第2の記憶媒体内で該パケットに対応するデータ	
D which corresponds to said packet inside logical address , aforementioned transfer original second storage media	
タが格納されている場所に対応するメモリアドレス、及び前記転送先の所定のハ	
predetermined of memory address , and aforementioned forwarding destination which correspondto site where [ta ] is housed [ha ]	
ードウェアを識別するためのハードウェアアドレスを統合して定義されたワール	

Integrating hardware address in order to identify [douea ], it was defined [waaru ]	
ドワイドにユニークな一次元のアドレスを用いる ことを特徴とするファイル転送方法。	
file transfer method . which designates that unique one-dimensional address is used for [dowaido ] as feature	
5. 請求項1記載のファイル転送方法において、前記転送元で、前記ファイルデータを前記ネットワークアダプタカードから前	
Regarding to file transfer method which is stated in 5.Claim 1, inaforementioned transfer origin, aforementioned file data from theaforementioned network adapter card before	
記ネットワークへ伝送する場合に、前記データファイルが1個または複数個の packets 単位で転送され、さらに、前記ファイルの転送元で、前記ファイルデータを該ネットワークアダプタカー	
When transmission it does to description network , aforementioned data file to be transferred with packet unit of one or a plurality , furthermore, intransfer origin of aforementioned file , aforementioned file data said network adapter car	
ドから前記ネットワークへ伝送する手順が、	
From [do ] to aforementioned network transmission protocol which isdone,	

送信パケットに前記転送元の第2の記憶媒体における該パケットに対応するデータの先頭アドレスとパケット長を付与して、前記転送先からの肯定応答を待たずに該パケットを順次送信する手順と、前記転送先からパケットの再送要求を受けたときには、該再送要求を受けたパケットのみを選択的に該転送元の第2の記憶媒体から読み出して再送する手順を含み、前記ファイルの転送先で、前記通信リンクを解放する手順が、受信したパケットのエラーチェックを行い、途中でデータの欠落及び誤りが発生しなかった受信パケットを、前記転送元へ肯定応答を返すことなく、該転送先の第2の記憶媒体へ順次蓄積してゆく手順と、データの欠落及びデータの誤りの中の少なくとも一方が検出されたパケットについては、直ちにこれを廃棄し、該パケットの蓄積されるべき記憶領域を該パケット容量分だけ空けておき、以降受信する欠落及び誤りのないパケットを順次蓄積してゆくとともに、前記検出されたパケットの先頭アドレスとパケット長を前記転送元へ通知して再送を要求する手順と、再送パケットを受け取った後、該パケットを前記空けておいた記憶領域に蓄積する手順を含むことを特徴とするファイル転送方法。

Granting start address and packet length of data which corresponds to the said packet in aforementioned transfer original second storage media in transmission packet ,without waiting for affirmative response from aforementioned forwarding destination ,when receiving retransmission demand for packet from protocol and aforementioned forwarding destination which sequential it transmits said packet , Only packet which receives said retransmission request reading \* is resent including protocol which from selectively said transfer original second storage media , protocol where with forwarding destination of aforementioned file , protocol which releases aforementioned communication link , does error check of packet which is received, midway omission of data or reception packet where error does not occur, without returning affirmative response toaforementioned transfer origin, to second storage media of said forwarding destination sequential compilation does and, It abolishes this at once concerning omission of data and the packet where at least one in error of data is detected, compilation of said packet memory region which it should you do, as is less crowded just said packet capacity fraction , later sequential compilation it does packet which does not have the omission or error which are received, Description above notifying start address and packet length of packet which is detected to aforementioned transfer origin, after receiving the protocol and retransmission packet which require retransmission, the said packet description above in memory region which is less crowded compilation file transfer method . which designates that protocol which is



6.

請求項 1 記載のファイル転送方法において、前記ファイルの転送先に、ネットワークインタフェースカードを介して複数の端末を接続し、該複数の端末の中の1つである第1の端末が、前記ファイルの転送元からデータファイルを読み出す際に、前記転送先が、前記転送元になり変わり、前記第1の端末との間で第1の通信リンクを確立し、前記転送先が、前記第1のリンクを介して前記第1の端末から送られる前記ファイルへのランダムアクセス要求をスタックし、前記転送先が、前記第1の端末になり変わり、前記転送元との間で第2のリンクを確立し、前記転送先が、前記第2のリンクを用いて前記データファイルを前記転送元からシーケンシャルに読み出し、該読み出したデータファイルを該転送先の第2の記憶媒体へ一括転送し、前記転送先が、前記一括転送終了後に前記第2のリンクを解放し、前記転送先が、前記第2の記憶媒体内のデータに対して、少なくとも解凍処理、通信処理の中のいずれか1つの処理を行いながら、順次処理された前記第2の記憶媒体内のデータを前記汎用データベースを介して該転送先の第1の記憶媒体へ転送し、前記転送先が、前記ランダムアクセス要求を順次実行し、前記第1のリンクを介して前記ファイルデータを該転送先の第1の記憶媒体から前記第1の端末へランダムに転送し、前記転送先が、前記ランダム転送終了後に前記第1のリンクを解放することを特徴とするファイル転送方法。

7.

請求項6記載の転送方法において、	
Regarding to transfer method which is stated in Claim 6,	
前記転送元に対して同時に複数の端末から複数のランダムアクセス要求があつ	
Vis-a-vis aforementioned transfer origin simultaneously from terminal of plural random access demand for plural oh	
た場合に、	

doneis included as feature

6.

Regarding to file transfer method which is stated in Claim 1,through [nettowaakuintafeesukaado ] to forwarding destination of aforementioned file , you connect terminal of plural , first terminal which is a one in terminal of the said plural , occasion where data file is read out from transfer originof aforementioned file , aforementioned forwarding destination , becomesaforementioned transfer cause and changes, establishes first communication link between aforementioned first terminal , Aforementioned forwarding destination , through aforementioned first link , demand foraforementioned file which is sent from aforementioned first terminal random access stack , aforementioned forwarding destination , to become theaforementioned first terminal and change, to establish second link betweenaforementioned transfer origin, aforementioned forwarding destination , Aforementioned data file from aforementioned transfer origin reading , said reading it is in sequential making use of aforementioned second link to second storage media of said forwarding destination to lump together transfer data file , while theaforementioned forwarding destination , to release aforementioned second link afterdescription above bundle transfer ending, aforementioned forwarding destination ,treating any one in midst of thawing treatment and communication treatmentat least vis-a-vis data inside aforementioned second storage media , sequential through aforementioned general purpose data bus , to transfer data inside aforementioned second storage media which was treated to first storage media of the said forwarding destination , aforementioned forwarding destination , sequential to execute aforementioned random access request, through aforementioned first link , from first storage media of the said forwarding destination to transfer aforementioned file data to random to theaforementioned first terminal , aforementioned forwarding destination , Description above file transfer method . which designates that theaforementioned first link is released after random transfer ending asfeature

7.

It is when,	
前記転送先が、前記同時に要求された複数のランダムアクセス要求を整理して	
Aforementioned forwarding destination , rearranging random access demand for plural which description above simultaneously is required	
、順次、各端末がランダムアクセス要求したデータファイルを前記転送元から一括してシーケンシャルに読み出すことを特徴とするファイル転送方法。8. 請求項6記載のファイル転送方法において、前記転送先をパケットのルータとして機能させ、前記転送先が前記第2のリンクを用いて前記データファイルを前記転送元から	
sequential , each terminal random access lumping together data file which isrequired regarding to file transfer method which is stated in the file transfer method . 8.Claim 6 which designates that it reads out in sequential as feature from aforementioned transfer origin, functioning with theaforementioned forwarding destination as router of packet , aforementioned forwarding destination making use of aforementioned second link aforementioned data file from aforementioned transfer origin	
シーケンシャルに読み出し、該転送先の第2の記憶媒体へ一括転送する際に、前	
When in sequential lumping together transferring to second storage media of the reading , said forwarding destination , before	
記転送元が、前記データファイルを1個または複数個のパケット単位で、かつ該	
Description transfer origin, aforementioned data file with packet unit of one or a plurality ,at same time said	
パケットに前記転送元の第2の記憶媒体内で該パケット	に対応するデータが格納
In packet inside aforementioned transfer original second storage media said packet	data which corresponds houses

されている場所を示すメモリアドレスとサイズの情報を付与して送信し、前記転送先が該転送先の第2の記憶媒体内のデータに対して、少なくとも解凍処理、通信処理の中のいずれか1つの処理を行いながら、順次処理された該第2の記憶媒体内のデータを前記汎用データベースを介して該転送先の第1の記憶媒体へ転送する際に、該転送先が、前記転送元によって付与されたメモリアドレスとサイズの情報に元、該第2の記憶媒体内の受信パケットを該転送先の第1の記憶媒体に蓄積し、前記転送先が前記ランダムアクセス要求を順次実行し、前記第1のリンクを介して前記ファイルデータを該転送先の第1の記憶媒体から前記第1の端末へランダムに転送する際に、前記転送先が、前記データファイルを1個または複数個のパケット単位で、かつ、前記転送元によって付与されたメモリアドレスとサイズの情報を再び該パケットに付与して前記第1の端末へ転送し、さらに、前記第1の端末において、前記パケットに付与されたメモリ

Granting data of memory address and size which show site which is done while transmitting, when aforementioned forwarding destination treating any one in midst of thawing treatment and communication treatment at least vis-a-vis data inside second storage media of said forwarding destination to the first storage media of said forwarding destination , sequential data inside said second storage media which wastreated through aforementioned general purpose data bus , transferring, When said forwarding destination , in data of memory address and size which are granted in aforementioned transfer origin in origin, compilation does reception packet inside said second storage media in first storage media of said forwarding destination , theaforementioned forwarding destination sequential executes aforementioned random access request, through aforementioned first link , aforementioned file data from the first storage media of said forwarding destination transferring to random to aforementioned first terminal , aforementioned forwarding destination , Aforementioned data file with packet unit of one or a plurality , at sametime, granting data of memory address

アドレスとサイズの情報を元に該第 1 の端末の  
 所定の記憶領域にデータを蓄積する ことを特  
 徴とするデータ転送方法。

and size which are granted in aforementioned transfer origin to  
 said packet again, on basis of data of memory address and  
 size which it transfers to the aforementioned first terminal , are  
 granted to aforementioned packet furthermore, in  
 aforementioned first terminal , in predetermined memory  
 region of said first terminal data transfer method . which  
 designates that compilation it does data as feature

9.

9.

請求項1～請求項8記載のいずれか1項記載のデータ転送方法を実行する		
data transfer method which is stated in any one claim which is stated in Claim 1 ~Claim 8 is executed		
ためのプログラムを記憶した記憶媒体。		
storage media . which program of for sake of storage is done		
10. データ転送を行うための第1の汎用データバスと、		
first general purpose data bus in order to do 10.data transfer and,		
第1の記憶媒体と、		
first storage media and,		
前記第1の記憶媒体より入出力速度が速い第2の記憶媒体と、		
second storage medium where input-output speed is faster than aforementioned first storage media and,		
前記第1の汎用データバスに接続されている計算機通信用の第1のネットワーク		
first for computer communication which is connected to aforementioned first general purpose data bus [network]		
クアダプタカードと、		
[kuadaputakaado ] With,		
通信リンクを設定する前に、前記第1の記憶媒体内のファイルデータに対して		
Before setting communication link , in file data inside aforementioned first storage media confronting		
、少なくとも圧縮、プロトコル終端、及びフレーミングの中のいずれか1つの処		
At least place of any one in midst of compression, protocol terminal , and flaming		
理を行いながら、該ファイルデータを前記第1の汎用データバスを介して前記第2の記憶媒体へ順次転送するための第1の転送手段と、		

While doing reason, first forwarding means in order through aforementioned first general purpose data bus , sequential to transfer said file data to aforementioned second storage media and,		
前記ファイルデータに対する処理の完了後、通信リンクを設定し	、前記	第2の
After completing treatment for aforementioned file data , communication link is set	Description above	second

記憶媒体内のファイルデータに対して処理を施さずに、該ファイルデータを、前記第 1 の汎用データバスを介して直接、前記第 1 のネットワークアダプタカードへ一括転送し、該第 1 のネットワークアダプタカードからネットワークへ伝送するための伝送手段と、を有する転送元計算機と、データ転送を行うための第 2 の汎用データバスと、第 3 の記憶媒体と、前記第 3 の記憶媒体より入出力速度が速い第 4 の記憶媒体と、前記第 2 の汎用データバスに接続されている計算機通信の第 2 のネットワークアダプタカードと、前記ネットワークから前記第 2 のネットワークアダプタカードへ伝送された前記ファイルデータを、データの解凍、プロトコル終端、及びフレーミングを含むいずれの処理も施さずに、前記第 2 の汎用データバスを介して前記第 4 の記憶媒体へ一括転送し、前記通信リンクを解放するための解放手段と、前記通信リンクを解放した後、前記第 4 の記憶媒体内のデータに対して、少なくとも解凍処理、通信処理の中のいずれか 1 つの処理を行いながら、順次処理された前記第 4 の記憶媒体内のデータを前記第 2 の汎用データバスを介して前記第 3 の記憶媒体へ転送するための第 2 の転送手段とを有する転送先計算機とを具備することを特徴とするファイル転送装置。

Without administering treatment vis-a-vis file data inside storage media ,said file data , through aforementioned first general purpose data bus , transfer original computer and second general purpose data bus in order to do data transfer and storage media of the third which possess transmission means. in order directly, it lumps together transfers to aforementioned first network adapter card , from said first network adapter card to network the transmission to do and, From second network adapter card and aforementioned network for computer communication which is connected to storage media and aforementioned second general purpose data bus of 4 th where input-output speed is faster than storage media of aforementioned third the aforementioned file data which transmission is done, without administering thawing , protocol terminal , of data or each treatment which includes flaming to aforementioned second network adapter card , through aforementioned second general purpose data bus , it lumpstogether transfers to storage media of aforementioned 4 th , While treating any one in midst of thawing treatment and communication treatment at least releasing means in order to release aforementioned communication link and, after releasing aforementioned communication link , vis-a-vis the data inside storage media of aforementioned 4 th , sequential file transfer device . which designates that forwarding destination computer which possesses second forwarding means in order through aforementioned second general purpose data bus ,to transfer data inside storage media of aforementioned 4 th which were treated to storage media of aforementioned third is possessed as feature

11.

請求項 10 記載のファイル転送装置において、前記転送元計算機が、前記ファイルデータを前記第 1 のネットワークアダプタカードから前記ネットワークへ伝送する場合に、該ファイルデータを 1 個または複数個の packets 単位で転送し、かつ、該 packets の転送先を指定するネットワークアドレスとして、前記第 2 の記憶媒体内で該 packets に対応するデータが格納される場所に対応するメモリアドレスを用いる ことを特徴とするファイル転送装置。

11.

When aforementioned transfer original computer , aforementioned file data from aforementioned first network adapter card transmission it does to the aforementioned network in file transfer device which is stated in Claim 10, as network address which transfers said file data with packet unit of one or a plurality , at same time, appoints forwarding destination of said packet , file transfer device . which designates that memory address which corresponds to site where data which corresponds to the said packet inside aforementioned second storage media is housed is used as feature

12.

12.

請求項 11 記載の転送装置において、ネットワーク層の protocols として インターネットプロトコル(IP)を用い、IP ヘッダのオプション領域に、前記メモリアドレスを付与することを特徴とするファイル転送装置。

13.

請求項 10 記載のファイル転送装置において、前記転送元計算機が、前記ファイルデータを前記第 1 のネットワークアダプタカードから前記ネットワークへ伝送する場合に、該ファイルデータを 1 個または複数個の packet 単位で転送し、かつ、該 packet の転送先を指定するネットワークアドレスとして、ネットワーク層の論理的アドレス、前記第 2 の記憶媒体内で該 packet に対応するデータが格納されている場所に対応するメモリアドレス、及び前記転送先計算機内の所定のハードウェアを識別するためのハードウェアアドレスを統合して定義されたワールドワイドにユニークな一次元のアドレスを用いることを特徴とするファイル転送装置。

14.

請求項 10 記載のファイル転送装置において、前記転送元計算機が、前記ファイルデータを前記第 1 のネットワークアダプタカードから前記ネットワークへ伝送する場合に、前記データファイルを 1 個または複数個の packet 単位で転送し、さらに、前記伝送手段が、送信 packet に前記第 2 の記憶媒体における該 packet に対応するデータの先頭アドレスと packet 長を付与して、前記転送先計算機からの肯定応答を待たずに該 packet を順次送信する手段と、前記転送先計算機から packet の再送要求を受けたときには、該再送要求を受けた packet のみを選択的に前記第 2 の記憶媒体から読み出して再送する手段を有し、前記解放手順が、受信した packet のエラーチェックを行い、途中でデータの欠落及び誤りが発生しなかった受信 packet を、前記転送元計算機へ肯定応答を返すことなく、前記第 4 の記憶媒体へ順次蓄積してゆく手段と、データの欠落及びデータの誤りの中の少なくとも一方が検出された packet については直ちにこれを廃棄し、該 packet の蓄積されるべき記憶領域を該 packet 容量分だけ空けておき、以降受信する欠落及び誤りのない packet を順次蓄積してゆくとともに、前記検出された packet の先頭アドレスと packet 長を前記転送元計算機へ通知して再送を要求する手段と、再送 packet を受け取った後、該 packet を前記空けておいた記憶領域に蓄積する手段を有することを特徴とするファイル転送装置。

file transfer device . which designates that aforementioned memory address is granted to option region of IP header , making use of Internet protocol (IP )in transfer device which is stated in Claim 11, as protocol of network layer , as feature

13.

When aforementioned transfer original computer , aforementioned file data from aforementioned first network adapter card transmission it does to theaforementioned network in file transfer device which is stated in Claim 10, as network address which transfers said file data with packet unit of one or a plurality , at same time, appoints forwarding destination of said packet , Integrating hardware address in order to identify predetermined hardware inside memory address , and aforementioned forwarding destination computer which correspond to site where the data which corresponds to said packet inside logical address , aforementioned second storage media of network layer is housed, file transfer device . which designates that it uses unique one-dimensional address for world wide which is defined asfeature

14.

When aforementioned transfer original computer , aforementioned file data from aforementioned first network adapter card transmission it does to theaforementioned network in file transfer device which is stated in Claim 10, granting start address and packet length of data which corresponds to said packet which transfers aforementioned data file with packet unit of one or a plurality , furthermore, aforementioned transmission means , in theaforementioned second storage media in transmission packet , Without waiting for affirmative response from aforementioned forwarding destination computer ,when receiving retransmission demand for packet from means. aforementioned forwarding destination computer which said packet sequential is transmitted, the error check of packet where only packet which receives said retransmission request reading \* is resent has means which from selectively aforementioned second storage media , aforementioned release protocol ,receives action, Is done concerning omission of means. data which and packet where at least one in error of data is detected to storage media of theaforementioned 4 th midway omission of data and reception packet where error does not occur, to aforementioned transferoriginal computer without returning affirmative response, sequential compilation this isabolished at once, compilation of said packet memory region which it should you do as it is lesscrowded just said packet capacity fraction , later sequential compilation it does packet which doesnot have omission or error which are received, description abovenotifying start address and packet length of packet which is detected toaforementioned transfer

ことを特徴とするファイル転送装置。

15.

請求項 10 記載のファイル転送装置において、さらに、前記転送先計算機にネットワークインタフェースカードを介して接続された複数の端末を具備し、該転送先計算機が、該複数の端末の中の 1 つである第 1 の端末が前記転送元計算機からデータファイルを読み出す際に、前記転送元計算機になり変わり、前記第 1 の端末との間で第 1 の通信リンクを確立し、前記第 1 のリンクを介して前記第 1 の端末から送られる前記ファイルへのランダムアクセス要求をスタックし、前記第 1 の端末になり変わり、前記転送元計算機との間で第 2 のリンクを確立し、前記第 2 のリンクを用いて前記データファイルを前記転送元計算機からシーケンシャルに読み出し、該読み出したデータファイルを前記第 4 の記憶媒体へ一括転送し、前記一括転送終了後に前記第 2 のリンクを解放し、前記第 4 の記憶媒体内のデータに対して、少なくとも解凍処理、通信処理の中のいずれか 1 つの処理を行いながら、順次処理された前記第 4 の記憶媒体内のデータを前記第 2 の汎用データベースを介して前記第 3 の記憶媒体へ転送し、前記ランダムアクセス要求を順次実行し、前記第 1 のリンクを介して前記ファイルデータを前記第 3 の記憶媒体から前記第 1 の端末へランダムに転送し、前記ランダム転送終了後に前記第 1 のリンクを解放する手段を有することを特徴とするファイル転送装置。

16.

請求項 15 記載の転送装置において、

In transfer device which is stated in Claim 15,

前記転送先計算機が、前記転送元計算機に対して同時に複数の端末から複数の

Aforementioned forwarding destination computer , vis-a-vis aforementioned transferoriginal computer simultaneously from terminal of plural plural

original computer , after receiving means. retransmission packet which requires retransmission, said packet description above in memory region which is less crowded the compilation file transfer device . which designates that it possesses the means which is done as feature

15.

Furthermore, through [nettowaakuintafeesukaado ] to aforementioned forwarding destination computer , in file transfer device which is stated in Claim 10, terminal of plural which is connected is possessed, said forwarding destination computer , occasion where the first terminal which is a one in terminal of said plural reads out data file from aforementioned transfer original computer , becomes theaforementioned transfer original computer , change, You establish first communication link between aforementioned first terminal , through theaforementioned first link , demand for aforementioned file which issent from aforementioned first terminal random access stack , becomes theaforementioned first terminal and changes, establishes second link between theaforementioned transfer original computer , Aforementioned data file from aforementioned transfer original computer reading , said reading it is in sequential making use of aforementioned second link to storage media of aforementioned 4 th to lump togethertransfer data file , while releasing aforementioned second link afterdescription above bundle transfer ending, treating any one in themidst of thawing treatment and communication treatment at least vis-a-vis the data inside storage media of aforementioned 4 th , sequential , data inside storage media of aforementioned 4 th which were treated through aforementioned second general purpose data bus , is transferred to storage media of aforementioned third , aforementioned random access request sequential is executed, through aforementioned first link , theaforementioned file data from storage media of aforementioned third istransferred to random to aforementioned first terminal , Description above file transfer device . which designates that itpossesses means which releases aforementioned first link after random transfer ending as feature

16.

ランダムアクセス要求があった場合に、該同時に要求された複数のランダムアク
When there is random access request, said plural which is requiredsimultaneously [randamuaku ]
セス要求を整理して、順次、各端末がランダムアクセス要求したデータファイル
Rearranging [sesu ] request, sequential , each terminal random access data file whichis required
を前記転送元計算機から一括してシーケンシャルに読み出す手段を有する
Lumping together from aforementioned transfer original computer , the means which it reads out in sequential it possesses
ことを特徴とするファイル転送装置。
file transfer device . which designates thing as feature
17. 請求項 15 記載のファイル転送装置において、
In file transfer device which is stated in 17.Claim 15,
前記転送先計算機がパケットのルータとして機能し、
Aforementioned forwarding destination computer it functions as router of packet ,
前記転送元計算機が、前記データファイルを、1個または複数個のパケット単
Aforementioned transfer original computer , aforementioned data file , packet of one or a plurality single
位で、かつ該パケットに前記転送元の第2の記憶媒体内で該パケットに対応する
At rank, at same time in said packet it corresponds to said packet inside aforementioned transfer original second storage media
データが格納されている場所を示すメモリアドレスとサイズの情報を付与して送信する手段を有し、前記転送先計算機が、前記第4の記憶媒体内のデータに対して、少なくとも解凍処理、通信処理の中のいずれか1つの処理を行いながら、順次処理された該第4の記憶媒体内のデー
Granting data of memory address and size which show site where data is housed while to possess means which ittransmits, aforementioned forwarding destination computer , treating any one in midstof thawing treatment and communication treatment at least vis-a-vis data inside storage media of aforementioned 4 th , sequential D inside storage media of said 4th which was treated
々を前記第2の汎用データバスを介して前記第3の記憶媒体へ転送する際に、前
When [ta ] through aforementioned second general purpose data bus , transferring to the storage media of aforementioned third , before
記転送元計算機によって付与されたメモリアドレスとサイズの情報に元、該第4の記憶媒体内の受信パケットを該

<p>第3の記憶媒体に蓄積する手段と、</p>
<p>In data of memory address and size which are granted with the description transfer original computer in origin, reception packet inside storage media of said 4th in storage media of said third compilation the means. which is done</p>
<p>前記ランダムアクセス要求を順次実行し、前記第1のリンクを介して前記ファ</p>
<p>sequential to execute aforementioned random access request, through the aforementioned first link , description above [fa ]</p>

イルデータを前記第3の記憶媒体から前記第1の端末へランダムに転送する際に、前記データファイルを1個または複数個の packets 単位で、かつ、前記転送先計算機によって付与されたメモリアドレスとサイズの情報を再び該 packets に付与して前記第1の端末へ転送する手段を有し、前記第1の端末が、前記 packets に付与されたメモリアドレスとサイズの情報を元に該第1の端末の所定の記憶領域にデータを蓄積する手段を有することを特徴とするデータ転送装置。

When y1 data from storage media of aforementioned third transferring to random to aforementioned first terminal , aforementioned data file with packet unit of one or a plurality , at same time, granting data of memory address and size which are granted with aforementioned forwarding destination computer to said packet again, to possess means which it transfers to aforementioned first terminal , aforementioned first terminal , On basis of data of memory address and size which are granted to aforementioned packet in predetermined memory region of said first terminal data transfer device . which designates that it possesses means which data compilation is done as feature

**Specification**

**【発明の詳細な説明】**

高速一括ファイル転送方法及び装置並びに転送方法を実行するためのプログラムを記憶した記憶媒体 技術分野 本発明はネットワークに接続されたパーソナルコンピュータ、ワークステーション、各種通信端末等の機器の間の通信に用いて好適な高速一括ファイル転送方法及び装置並びに転送方法を実行するためのプログラムを記憶した記憶媒体に関する。

**[Description of the Invention ]**

storage media technical field this invention which high speed collective file transfer method and the program in order to execute device and transfer method storage is done it was connected to network [paasonarukonpyuuta ], using for communication between workstation , various communication terminal or other equipment , regards storage media which preferred high speed collective file transfer method and program in order to execute device and transfer method storage is done.

**背景技術**

マルチメディア時代の本格化とともに、画像等の大容量バルクデータをサーバからユーザ端末に配送するサービスがますます重要となる。

**background technology**

With materialization of multimedia age , image or other large capacity bulk data service which from the server is delivered in user terminal more and more becomes important.

現状の VOD(ビデオオンデマンド)においては、ファイルの転送開始命令だけでなく映画の一時停止や巻き戻しなどの制御命令も、著しく増加し、これがサーバプロセッサの負荷となるため、ネットワークの高速化によるメリットを十分に生かすことができない。

Regarding VOD (video-on-demand ) of present state , also halt and rewinding or other control command of the motion picture , increase considerably not only a transfer start command of file ,because this becomes load of server processor , they are not possible to utilize merit to fully with acceleration of network .

これに対して、CD-ROM 1枚や映画1本に相当する大容量のバルクデータを数秒ないし数10秒でユーザの蓄積メディアに転送し、即座にネットワークを解放することによりユーザの利便性/経済性が考えられる。

Vis-a-vis this, bulk data of large capacity which is suitable to CDROM one layer and motion picture 1 is transferred to compilation media of user with several seconds or several 10 second , convenience /economy of user is thought by releasing network instantaneously.



本発明は、これを実現するための方法及び装置を提供するものであるが、本発明の背景技術となる各要素技術について以下、図面を参照して詳細に説明する。

図 20A は ATM ネットワークアダプタカードを装備した汎用計算機アーキテクチャを示す図であり、図 20B はファイル転送プロトコルを用いて、大容量ファイルを受信する際のデータの流れを示す流れ図である。

ファイル転送プロトコル(ftp)はインターネットプロトコル(TCP/IP)上に載るアプリケーションであり、TCP/IP を含め、計算機のホスト CPU(中央演算装置)でソフトウェアとして処理される。

また、図 21 は ATM(Asynchronous Transfer Mode:非同期転送モード)リンクを用いた場合のファイル転送プロトコル(ftp:file transfer protocol)のプロトコルスタックであり、実行するハードウェアを併記して示している。

TCP、IP 及び SNAP/LLC は、それぞれ転送制御プロトコル(Transmission Control Protocol)、インターネットプロトコル(Internet Protocol)及びサブネットワークアクセスポイント/論理リンク制御(Subnetwork Access Point/Logical Link Control)の略語である。

また、それぞれ、AAL は ATM アダプテーションレイヤ(ATM Adaptation Layer)、SAR はセル分割組立サブレイヤ(Segmentation And Reassembly Sublayer)、PHY は論理プロトコル(Physical Protocol)、S/P 変換はシリアル/パラレル変換を示す略語である。

this invention, it is a method in order to actualize this and something which offers device, but below, referring to drawing concerning each element technology which becomes background technology of this invention, you explain in detail.

As for Figure 20 A in figure which shows general purpose computer architecture which equips the ATM network adapter card, Figure 20 B when receiving large capacity file making use of file transfer protocol, is flowchart which shows flow of data.

file transfer protocol (ftp) with application which is recorded on Internet protocol (TCP/IP), includes TCP/IP, is treated with host CPU (central processing unit) of computer as the software.

In addition, inscribing hardware which with protocol stack of file transfer protocol (ftp:file transfer protocol) when ATM (Asynchronous transfer mode : asynchronous transfer mode) link is used, is executed it has shown Figure 21.

TCP, IP and SNAP/LLC, respective forwarding control protocol (Transmission control protocol), Internet protocol (internet protocol) and are abbreviation of sub network access point /logic link control (Subnetwork access Point/Logical Link control).

In addition, respectively, as for AAL ATM [adapteeeshonreiya] (ATMA daptation Layer), as for SAR cell portion percentage assembly sub layer (Segmentation And Reassembly Sublayer), as for PHY the logic protocol (physical Protocol), as for S/P conversion it is a abbreviation which shows serial /parallel conversion.

ここでファイル転送プロトコルを用いて、大容量ファイルを受信する際の動作
When receiving large capacity file here making use of file transfer protocol, operation
を説明する。なお、送信側については同様の動作の順番が逆になるだけなので、
You explain. Furthermore, because sequence of similar operation just becomes opposite is concerning transmitting side,
説明を省略する。ATM-LAN(ローカルエリアネットワーク:Local Area Network)
<seq>local area network work :local Area Network Explanation is abbreviated. ATM - LAN
etwork)等の計算機ネットワークE10から送られてきたデータは、まず、AT
data which is sent from etwork) or other computer network E10 first, AT

MアダプタカードE5で受信され、光モジュールやS/P変換チップ、セル同期チップによって、物理レイヤを終端され、53バイトのセルデータとして、ATMレイヤチップに渡され、ATMレイヤを終端される。ATMレイヤではVCI/VPI(仮想パス識別子:Virtual Path Identifier/仮想チャネル識別子:Virtual Channel Identifier)による分離、多重処理

It is received with Madapter card E5, with optical module and S/P conversion chip , cell synchronization chip , terminal is done physical layer , it is transferred by ATM layer chip as cell data of 53 byte , terminal is done ATM layer . With ATM layer to separate with VCI /VPI (virtual path identifier :virtual path identifier hIdentifier/virtual channel identifier :virtual channel identifier nnelIdentifier ) , multiple treatment

が行われる。AALレイヤ(通常タイプ5)ではSARチップにより、セルのヘッダを除いた 48 バイト(SAR-PDU(プロトコルデータユニット:Protocol Data Uint))の情報をリンクし、CRCチェック(巡回冗長検査:Cyclic Redundancy Check)やデータ長チェックを行い、CPCS(コンバージェンスサブ

<seq> [konbaajensusabu ] </seq>link to do data of SAR- PDU (protocol data unit :protocol data Uint ) , CRC check (Round redundant inspection:CyclicRedundancyCheck ) and to do the data length check , CPCS Is done. With AALlayer (Usually type 5 ) 48 byte which exclude header of cell due to SARchip

レイヤ共通部:Convergence Sublayer Common Part) - PDUのペイロードを形

payload of layer common section:ConvergenceSublayercommon Part) - PDU shape

成する(図 20B参照)。CPCS-PDUペイロードはユーザデータとして高速の汎用バス(ここではPCIバス(Peripheral Component Interconnect Bus))E3及びPCIブリッジE4を介して、ホストCPU・E1に転送される。

<seq>Here through PCI bus (Peripheralcomponent InterconnectBus ) E3 and PCI bridge E4, it is transferred to host CPU \*E1. It forms, \* (Figure 20 B reference ).As for CPCS- PDU payload as user data general purpose bus of high speed

ホストCPU・E1に送られたデータIPデータグラム化しており、CPUは

To data IP data gram which is sent to host CPU \*E1 we to have converted, as for CPU

IP レイヤを順次終端して、カプセル化されたファイル転送データの中身を取り出す。

sequential terminal doing IP layer , it removes contents of file transfer data which encapsulation is done.

そして、ホスト CPU・E1 は、取り出した転送データの中身を PCI バス E3 を介して、ハードディスク E6 に格納する。

And, host CPU \*E1, through PCI bus E3, houses contents of transfer data which is removed in hard disk E6.

なお、図 20Aにおいて、E30はCRT(ディスプレイ:Cathod-Ray Tube)

Furthermore, in Figure 20 A, as for E30 CRT (display :Cathod-RayTube )

、E31はグラフィックボード、E32はキーボード、E33はキーボードコン

As for E31 as for graphic board , E32 as for keyboard , E33 keyboard Kong

トローラであり、PCIバスE3及びPCIブリッジE4を介してホストCPU

With jp7 roller , through PCI bus E3 and PCI bridge E4, host CPU
・E1と接続されている。
*E1 it is connected.
上記のようなファイル転送プロトコルは多くのプロトコルスタック上で実現さ
As description above as for file transfer protocol on many protocol stack actualization
れるアプリケーションであり、その下位のプロトコルの多くがホストCPUで処理されている。特にTCPレイヤはデータの着信応答を行うため、ホストCPUには大きな負荷がかかっていた。このため、ATM-LANのように高速な計算機ネットワークから高速でネットワークアダプタカードにデータが伝送され、アダプタカードからPCIのような高速転送可能な汎用バスを介してホストCPU側にデータが転送される場合においても、プロトコルの多くがCPUで処理されるため、ファイル転送のスルー
**With application , are treated many of protocol of lower position with host CPU . As for especially TCP layer in order to respond data receive , the large load depended on host CPU . Because of this , like ATM - LAN from high speed computer network with high speed the data transmission is done in network adapter card , through high speed transmission possible general purpose bus like PCI from adapter card when data is transferred to host CPU side, putting, because are treated many of protocol with CPU ,slew of file transfer
ットがCPUの処理能力に制限されてしまい、高速な計算機ネットワークの能力が十分活かせないという問題点があった。次に、ATMネットワーク内で通信速度が異なるデータ転送が介在する場合に
There was a problem that [putto ] is restricted by throughput of CPU ,fully cannot utilize capacity of high speed computer network . When next, communication speed different data transfer lies between inside ATM network
ついて説明する。図 22 は、従来のATMリンクを用いた場合のデータ転送の手順
Being attached, you explain. As for Figure 22 , protocol of data transfer when conventional ATM link is used
と、それを実現するための装置構成の概略を表している。ファイルデータを格納
With, outline of equipment configuration in order to actualize that is displayed. It houses file data
しているコンテンツサーバB101とコンテンツサーバB101内のデータを読み出す端末B102が1台のATMスイッチB103に接続されており、それぞれのインタフェース速度が異なるものとする。すなわち、コンテンツサーバB101とATMスイッチB103とのインタフェース速度は155Mbps(Megabits per second)であり、ATMスイッチB103と端末B102とのインタフェース速度は25Mbpsであるとする。 端末B102がコンテンツサーバB101内のファイルデータにアクセスしてデータを読み出すには、まず、端末B102がシグナリングによりコンテンツ
terminal B102 which reads out data inside contents server B101 and contents server B101 which it has done is connected by ATM switch B103 of 1, respective interface speed makes different ones. As for interface speed of namely, contents server B101 and ATM switch B103 with 155 Mbps (Megabitspersecond ), as for interface speed of ATM switch B103 and terminal B102 we assume that they are 25 Mbps . terminal B102 access doing in file data inside contents server B101, to read out data , first, terminal B102 with Signa ring [kontentsusa ]
サーバB101とのATMリンクを設定要求し、ATMスイッチB103がコンテ

setting request to do ATM link of [ba ] B101, ATM switch B103 [konte ]
--

コンテンツサーバ B101 と端末 B102 間の ATM リンクを設定する。

この処理は図中の C プレーン(呼制御信号転送プレーン)を用いてなされる。

次いで、ATM リンク確立後、コンテンツサーバ B101 から端末 B102 へ ATM セル化されたファイルデータが転送され、これは図中の U プレーン(ユーザ情報転送プレーン)を用いてなされる。

ところが、このとき ATM スイッチ B103 は AAL(ATM アダプテーション・レイヤ)以上の上位レイヤ処理は行わず、セルのヘッダ情報のみ(VCI, VPI)を参照して、セルを一方のポートから他方のポートへスイッチングするだけである。

また、ATM スイッチ B103 の内部には、速度変換に必要とされる大規模な記憶媒体が存在していない。

このため、ATM スイッチ B103 のそれぞれのポートのインタフェース速度が、上述した如く 155 Mbps と 25 Mbps というように異なる場合には、スイッチの転送速度が低速側に場合は 25 Mbps) に律則され、高速インタフェース(コンテンツサーバ B101-ATM スイッチ B103 間)が有効利用できない。

さらに、コンテンツサーバ B101 内のファイルデータに対して、シーケンシャルアクセスするのではなく、映像データの再生時のように巻き戻し、早送り、一時停止といったランダムアクセスを行う場合は、コンテンツサーバ B101 に過大な負荷がかかる上、さらに複数の端末から同時にランダムアクセスされた場合などは、いっそう転送速度が低下してしまう。

以上説明したように、図 22 に示すような従来の ATM リンクによるファイルのデータ転送では、ATM スイッチ B103 が、異なるインタフェース速度のポート間の大容量ファイルのデータ転送時に必要とされる速度変換のための大容量記憶媒体を持っていない。

したがって、転送速度が低速インタフェース速度に律則され、高速インタフェースが有効に利用できないという問題点があった。

さらに、インタフェース速度が低速に制限されることに加え、映像、音声データファイルのアクセスのようにファイルへのランダムアクセス時においてコンテンツサーバ B101 に過大な負荷がかかり、さらにはファイルデータの転送速度を低下させる要因となっていた。

[ntsusaaba ] B101 and ATM link between terminal B102 are set.

You can do this treatment making use of Cplane (Call control signal transfer plane ) of in the diagram .

Next, after ATM link establishing, from contents server B101 to ATM-cell is converted file data which is transferred to terminal B102, can do this making use of Uplane (user data transfer plane ) of in the diagram .

However, at time of this ATM switch B103 upper position layer not to treat above AAL (ATM [adaputeeshon ] \* layer ), only header information of cell referring to (VCI , VPI ), the cell from port of one side switching just is done to port of other .

In addition, large scale storage media which is needed for rate conversion does not exist in interior of ATM switch B103 .

Because of this , as though interface speed of respective port of ATM switch B103 did, description above, way, 155 Mbps and 25 Mbps , incase of different , when forwarding rate of switch is on low speed side, the governing it is done in 25 Mbps ) , high speed interface (Between contents server B101-ATM switch B103 ) effective use is not possible.

Furthermore, it is not [shiikensharuakusesu ] vis-a-vis file data inside contents server B101, liketime of regeneration of image data when random access such as rewinding , rapid feed , haltis done, in addition to fact that excessive load depends on contents server B101,when furthermore simultaneously random access it is done from terminal of plural etc, forwarding rate decreases more.

As above explained, with data transfer of file , ATM switch B103 , does nothave large capacity storage media for rate conversion which is needed at time of data transfer of large capacity file between port of different interface speed with kind of conventional ATM link which is shown in Figure 22 .

Therefore, there was a problem that forwarding rate governing is done in the low speed interface speed , cannot utilize high speed interface effectively.

Furthermore, in addition to interface speed being restricted to low speed ,like access of image , voice data file excessive load depended on contents server B101 in thetime of random access to file , furthermore forwarding rate of file data hadbecome factor which decreases.

次に、ファイルデータ転送のプロトコルの背景技術について説明する。

TCP(Transmission Control Protocol:転送制御プロトコル)は、計算機間通信で現在広く用いられているトランスポート層プロトコルである。

TCP は信頼性の高い通信を実現するため、以下のように送/受信間でハンドシェイクを行い、データにエラーや欠落のあった場合再送を行っている。

なお以下で用いるセグメントとは TCP での転送単位を表し、他のプロトコルにおけるパケットあるいはフレームに対応する。

送信側は、これから送信するセグメントのシーケンス番号(SEQ:Sequence Number)を TCP ヘッダにマッピングして送信する。

シーケンス番号は、全データストリーム中におけるそのセグメントの最初のデータ位置をバイト単位で表したもので、通信確立時に初期化され、以降転送されたデータのバイト数を加算してゆく。

受信側は上記セグメントを正しく受信すると、応答確認番号を TCP ヘッダにマッピングして、これを ACK(Acknowledgement:肯定応答)として送信側に返す。

応答確認番号は、送信側が次に送信すべきシーケンス番号を表し、データを欠落なく正しい順序で受信できたことを送信側に通知する目的で使われる。

送信側はこの ACK を待ち、ACK を受信した後初めて次のセグメントを送信する。

一定のタイムアウト値以内に ACK を受信しなければ、そのセグメントは再送する。

TCP の場合、上記の ACK のタイムアウト値以内の未受信によるものが、再送のための唯一の機構である。

図 23 に TCP による再送処理を示す。

図 23 は、送信側及び受信側計算機間における TCP のフローコントロールを示すタイムチャートである。

図 23 は、送信側から受信側へ 10 バイト×5 セグメントのデータを転送する例を示している。

また、図 23 は、1 回目の転送時におけるシーケンス番号 SEQ=40 のセグメントが、受信側で正しく受信されない場合を示している。

送信側は、5 個のセグメントの TCP ヘッダにそれ

Next, you explain concerning background technology of protocol of file data transfer .

TCP (Transmission control protocol :forwarding control protocol ) is transport layer protocol which presently is widely used with communication between computer .

TCP in order to actualize communication where reliability is high, like below does [handosheiku ] between sending / reception, when there is an error and an omission in data , resends.

Furthermore segment which is used at below you display transfer unit with TCP , you correspond to packet or frame in the other protocol .

mapping doing sequence number (SEQ:SequenceNumber ) of segment which is transmitted from now on in TCP header it transmits transmitting side .

As for sequence number , being something which displays initial data position of the segment in in all data stream with byte unit , at time of communication establishment initialization it is done, later it adds number of bytes of the data which was transferred.

When above-mentioned segment is received correctly, mapping doing response verification number in TCP header , it returns called side to transmitting side ACK (Acknowledgement:affirmative response) as this.

Response verification number displays sequence number which transmitting side should transmit next, without omission is used data with the objective which notifies fact that with correct order it can receive to transmitting side .

transmitting side waits for this ACK, after receiving ACK, transmits following segment for first time.

If ACK is not received within fixed timeout value, it resends the segment .

In case of TCP , thing, is mechanism of only one because of retransmission with not yet reception which is within timeout value of above-mentioned ACK.

In Figure 23 retransmission treatment is shown with TCP .

As for Figure 23 , it is a time chart which shows flow control of the TCP in between transmitting side and called side computer .

Figure 23 has shown example which from transmitting side transfers the data of 10 byte X 5 segment to called side .

In addition, as for Figure 23 , segment of sequence number SEQ=40 at time of transfer of first , has shown case where it is not correctly received with called side .

mapping doing sequence number of respective SEQ=10, 20,

ぞれ SEQ=10,20,30,40,50 のシーケンス番号をマッピングして送信する。

受信側は、SEQ=10,20,30 の各セグメントを正しく受信し、その都度、ACK=20,30,40 を TCP ヘッダにマッピングして送信する。

送信側は、各セグメントの所定のタイムアウト値以内で ACK=20,30,40 を受信する。

この例では 1 回目の転送時に SEQ=40 のセグメントが受信側で正しく受信されないため、送信側では SEQ=40 のセグメントを送信してから所定のタイムアウト値以内で ACK=50 を受信することができない。

送信側は、タイムアウト値経過時点で SEQ=40 のセグメントにエラーが発生したものと判断し、SEQ=40 のセグメントを再送する。

また、SEQ=50 のセグメントに対する ACK も所定のタイムアウト値以内で送られてこないことになるので、送信側は SEQ=50 送信時からタイムアウト値が経過した時点で SEQ=50 のセグメントを再送する。

上記 TCP のように ACK のみで応答確認を行う方法では、再送を受けるためには常に送信側のタイム終了を待たなければならない。

また、エラーの生じたセグメントが 1 個だけだった場合でも、それ以降のセグメントをすべて再送しなければならない。

このため一旦エラーが発生すると、再送セグメントが累積的に増加してしまうという問題がある。

これらは、特に大容量のバルクデータを多くのセグメントに分割して転送するようなアプリケーションの場合、例えば、受信側から NAK (Negative Acknowledgement: 否定応答) を返して特定セグメントの強制的な再送を行う方法に対して、転送効率が著しく低下してしまう原因となる。

さらに TCP 等の応答確認は、IP (Internet Protocol: インターネットプロトコル) レイヤまでの下位レイヤ終端を終えてから行うソフト処理であるため、高速処理が困難である。

本発明は上記の点に鑑みてなされたものであり、本発明は、従来技術において、多くのプロトコル、通信処理を CPU が処理していたことによるファイル転送時のスループット低下を改善し、高速な計算機ネットワークを十分に活かす高スループットな高速一括ファイル転送方法及び装置並びに転送方法を実行するためのプログラム

30, 40, 50 in TCP header of 5 segment, it transmits transmitting side.

called side receives each segment of SEQ=10, 20, 30 correctly, every time, the mapping does ACK=20, 30, 40 in TCP header and transmits.

transmitting side receives ACK=20, 30, 40 within predetermined timeout value of each segment.

With this example segment of SEQ=40 being called side when transferring first, because it is not received correctly, after with transmitting side transmitting segment of SEQ=40, it is not possible to receive ACK=50 within predetermined timeout value.

As for transmitting side, it judges as thing where with timeout value passage time point error occurs in segment of SEQ=40, resends segment of SEQ=40.

In addition, because it means that either ACK for segment of SEQ=50 is not sent within predetermined timeout value, transmitting side from at time of SEQ=50 transmission timeout value resends segment of SEQ=50 with time point which passage is done.

Like above-mentioned TCP with method where does response verification with only ACK, in order to receive retransmission, you must wait for timer end of normally transmitting side.

In addition, segment after that must be resent entirely even with when segment which error occurs is just 1.

Because of this when error occurs once, there is a problem that retransmission segment increases in cumulative.

These, dividing bulk data of especially large capacity into many segment, in case of application which it transfers, returning NAK (Negative Acknowledgement: negative response) from for example called side, become cause where transport efficiency decreases considerably vis-a-vis method which resends specific segment forcible.

Furthermore after as for TCP or other response verification, finishing the lower position layer terminal to IP (internet protocol: Internet protocol) layer, because it is a software treatment which it does, fast processing is difficult.

As for this invention considering to above-mentioned point, being something which it is possible, this invention improves throughput decrease at time of file transfer by fact that CPU treated many protocol, communication treatments in Prior Art, It designates that storage media which high [suru uputsuto] high speed collective file transfer method and program utilizes high speed computer network to fully in

を記憶した記憶媒体を提供することを目的とする。

また、本発明の他の目的は、インタフェース速度の異なるATMリンクの速度変換が可能であり、しかも、大容量ファイルのランダムアクセスなどサーバにかかる大きな負荷をも分散して高い転送スループットを実現することにある。

また、本発明の他の目的は、各パケット毎のソフト処理による応答確認に起因する転送効率の低下を回避し、データを複数パケットに分割して転送する場合でも高いスループットが得られるデータ転送方法を提供することにある。

#### 発明の開示

上記課題を解決するため、本発明の第1の態様は、ファイルの転送元及びファイルの転送先でそれぞれ、データ転送を行うための汎用データベース、第1の記憶媒体、及び該第1の記憶媒体より入出力速度が速い第2の記憶媒体から成る汎用計算機アーキテクチャを用いて、(a)ファイルの転送元で、通信リンクを設定する前に、第1の記憶媒体内のファイルデータに対して、少なくとも圧縮、プロトコル終端、及びフレーミングの中のいずれか1つの処理を行いながら、該ファイルデータを汎用データベースを介して第2の記憶媒体へ順次転送する手順と、(b)前記ファイルの転送元で、前記ファイルデータに対する処理の完了後、通信リンクを設定し、前記第2の記憶媒体内のファイルデータを、それに対して処理を施さずに、前記汎用データベースを介して直接、該汎用データベースに接続されている計算機通信用のネットワークアダプタカードへ一括転送し、該ネットワークアダプタカードからネットワークへ伝送する手順と、(c)ファイルの転送先で、前記ネットワークから該転送先の汎用データベースに接続されているネットワークアダプタカードへ伝送された前記ファイルデータを、データの解凍、プロトコル終端、及びフレーミングを含むいずれの処理も施さずに、該汎用データベースを介して第2の記憶媒体へ一括転送し、前記通信リンクを解放する手順と(d)、前記ファイルの転送先で、前記通信リンクを解放した後、前記第2の記憶媒体内のデータに対して、少なくとも解凍処理、通信処理の中のいずれか1つの処理を行いながら、順次処理された前記第2の記憶媒体内のデータを前記汎用データベースを介して第1の記憶媒体へ転送する手順とを有することを特徴とするファイル転送方法である。

order which to execute device and transfer method storage is done is offered as objective .

In addition, as for other objective of this invention , rate conversion of the different ATM link of interface speed being possible, dispersing also large load where furthermore, it depends on server such as random access of large capacity file , it is to actualize high transfer throughput .

In addition, it is to offer data transfer method where high throughput is acquired even with when other objective of this invention evades decrease of the transport efficiency which originates in response verification in software treatment of each every packet , divides data into plural packet and transfers.

#### Disclosure of Invention

In order to solve above-mentioned problem , as for first embodiment of the this invention , respectively, making use of general purpose computer architecture which consists of the second storage media where input-output speed is faster than general purpose data bus , first storage media , and said first storage media in order to do data transfer with transfer origin of file and forwarding destination of the file , in transfer origin of ( a ) file , before setting communication link , vis-a-vis file data inside first storage media , At least, while treating any one in midst of compression, the protocol terminal , and flaming , in transfer origin of protocol and ( b ) aforementioned file which through general purpose data bus , sequential transfer the said file data to second description 100,000,000 media , after completing the treatment for aforementioned file data , to set communication link , file data inside aforementioned second storage media , Without administering treatment vis-a-vis that, through the aforementioned general purpose data bus , with forwarding destination of protocol and ( c ) file which it lumps together transfers to network adapter card for computer communication , directly, is connected to said general purpose data bus from said network adapter card to network the transmission it does, to network adapter card which from aforementioned network is connected to general purpose data bus of said forwarding destination transmission aforementioned file data which is done, Without administering thawing , protocol terminal , of data or each treatment which includes flaming , through said general purpose data bus , protocol which it lumps together transfers to second storage media , releases aforementioned communication link ( d ), with forwarding destination of aforementioned file , after releasing the aforementioned communication link , at least vis-a-vis data inside the aforementioned second storage media , thawing treatment, While treating any one in midst of communication treatment, sequential it is a file transfer method which designates that it possesses protocol which through aforementioned general

また、本発明のデータ転送方法を実行するためのプログラムは、記憶媒体に記憶させて形態で配布することができる。

また、本発明の他の態様は、(a)データ転送を行うための第1の汎用データバスと、第1の記憶媒体と、前記第1の記憶媒体より入出力速度が速い第2の記憶媒体と、前記第1の汎用データバスに接続されている計算機通信の第1のネットワークアダプタカードと、通信リンクを設定する前に、前記第1の記憶媒体内のファイルデータに対して、少なくとも圧縮、プロトコル終端、及びフレーミングの中のいずれか1つの処理を行いながら、該ファイルデータを前記第1の汎用データバスを介して前記第2の記憶媒体へ順次転送するための第1の転送手段と、前記ファイルデータに対する処理の完了後、通信リンクを設定し、前記第2の記憶媒体内のファイルデータに対して処理を施さずに、該ファイルデータを、前記第1の汎用データバスを介して直接、前記第1のネットワークアダプタカードへ一括転送し、該第1のネットワークアダプタカードからネットワークへ伝送するための伝送手段とを有する転送元計算機と、(b)データ転送を行うための第2の汎用データバスと、第3の記憶媒体と、前記第3の記憶媒体より入出力速度が速い第4の記憶媒体と、前記第2の汎用データバスに接続されている計算機通信の第2のネットワークアダプタカードと、前記ネットワークから前記第2のネットワークアダプタカードへ伝送された前記ファイルデータを、データの解凍、プロトコル終端、及びフレーミングを含むいずれの処理も施さずに、前記第2の汎用データバスを介して前記第4の記憶媒体へ一括転送し、前記通信リンクを解放するための解放手段と、前記通信リンクを解放した後、前記第4の記憶媒体内のデータに対して、少なくとも解凍処理、通信処理の中のいずれか1つの処理を行いながら、順次処理された前記第4の記憶媒体内のデータを前記第2の汎用データバスを介して前記第3の記憶媒体へ転送するための第2の転送手段とを有する転送先計算機とを具備することを特徴とするファイル転送装置である。

本発明のファイル転送方法は、従来技術において

purpose data bus , transfers data inside the aforementioned second storage media which was treated to first storage media as feature.

In addition, program in order to execute data transfer method of this invention , the storage doing in storage media , distribution fabric is possible with the form .

In addition, other embodiment of this invention first network adapter card for computer communication which is connected to second storage media and aforementioned first general purpose data bus where the input-output speed is faster than first general purpose data bus and first storage media and aforementioned first storage media in order to do ( a ) data transfer and, before setting communication link , atleast vis-a-vis file data inside aforementioned first storage media ,compression, While treating any one in protocol terminal , and flaming , first forwarding means in order through aforementioned first general purpose data bus , sequential to transfer the said file data to aforementioned second storage media and after completing treatment for aforementioned file data , it sets communication link , without administering treatment vis-a-vis file data inside aforementioned second storage media , the said file data , through aforementioned first general purpose data bus , direct, second network adapter card for computer communication which is connected to storage media and the aforementioned second general purpose data bus of 4 th where input-output speed is faster than the transfer original computer and second general purpose data bus in order to do ( b ) data transfer and storage media of third and storage media of aforementioned third which possess transmission means in order it lumps together transfers to the aforementioned first network adapter card , from said first network adapter card to network transmission to do and, From aforementioned network aforementioned file data which transmission is done, without administering thawing , protocol terminal , of data or each treatment which includes flaming to aforementioned second network adapter card ,through aforementioned second general purpose data bus , releasing means in order it lumps together transfers to storage media of aforementioned 4 th , to release the aforementioned communication link and, after releasing aforementioned communication link ,vis-a-vis data inside storage media of aforementioned 4 th , At least while treating any one in midst of thawing treatment and communication treatment, sequential it is a file transfer device which designates that forwarding destination computer which possesses second forwarding means in order through aforementioned second general purpose data bus , to transfer data inside storage media of aforementioned 4 th which were treated to storage media of the aforementioned third is possessed as feature.

file transfer method of this invention designates that protocol



て CPU が処理していたプロトコル終端などの多くの処理をデータ伝送時には行わないことを特徴とする。

ファイルデータはホストメモリなどの高速入出力可能なメモリとネットワークアダプタカードとの間で一括して入出力されるため、ファイル転送のスループットが上がる。

これにより、ネットワークは短い時間で解放されるため、高速な計算機ネットワークを有効利用できる。

また、一旦、メモリに蓄えられるファイルデータは送信側ではネットワークの通信リンク設定前に、ハードディスクのような低速の大容量記憶媒体から順次蓄積され、ファイルの受信側では、ネットワークを解放した後、ディスクに順次に蓄積される。

このメモリ、ハードディスク間のデータの転送時に、汎用データバスの転送速度に比べ、ハードディスクの入出力が十分低速であるため、この差を利用して、プロトコル処理等の処理が可能となる。

本発明のファイル転送方法では、高速ネットワークを有効利用するため、ネットワークアダプタカードの転送速度を制限しないよう、ファイルデータを第 1 又は第 3 の記憶媒体より高速の第 2 又は第 4 の記憶媒体との間で一括転送し、この間、中央演算装置による処理を行わない。

第 2 又は第 4 の記憶媒体と第 1 又は第 3 の記憶媒体間のデータ転送はネットワークを解放している間、すなわち、ファイルの転送元ではファイルの伝送前、ファイルの転送先ではファイルの伝送後に行う。

このとき、第 1 又は第 3 の記憶媒体の書き込み速度と汎用データバスとの速度差を利用してデータの圧縮・解凍、プロトコル終端等の処理を行いながら、第 2 又は第 4 の記憶媒体と第 1 又は第 3 の記憶媒体の間で逐次データを転送する。

これにより、ファイル転送の際、ネットワークの高スループットを実現でき、大容量のファイル転送の際にも早期にネットワークを解放することができる。

図面の簡単な説明 図 1 は、この発明の一実施形態による ATM ネットワークに接続される計算機の構成例とファイルデータの流れを示すブロック図である。

図 2 は、この発明の他の一実施形態による ATM ネットワークに接続される計算機の構成例と

terminal or other many treatment where CPU treated in Prior Art is not done at time of data transmission as feature.

As for file data lumping together host memory or other high speed input-output possible memory, and between network adapter card because input-output it is done, throughput of file transfer rises.

Because of this, network because in a short time it is released, the effective use can do high speed computer network.

In addition, file data which once, is stored in memory with the transmitting side before communication link setting of network, sequential compilation is done from the large capacity storage media of low speed, like hard disk with called side of file, after releasing network, in disk compilation is done in sequential.

Because when transferring data between this memory, hard disk, input-output of hard disk is fully low speed in comparison with forwarding rate of general purpose data bus, making use of this difference, protocol treatment or other treatment becomes possible.

With file transfer method of this invention, in order effective use to do high speed network, in order not to restrict forwarding rate of network adapter card, from the storage media of first or third it lumps together transfers file data second of high speed, or between storage media of 4 th at this time, does not treat with central processing unit.

second or storage media and first or third of 4 th data transfer between storage media while releasing network, in transfer origin of the namely, file before transmission of file, with forwarding destination of file does after transmission of file.

Compressing at time of this, making use of speed difference of the writing speed and general purpose data bus of storage media of first or third while treating data thawing, protocol terminal or other, second or storage media and it transfers thesequential data first or third of 4 th between storage media.

Because of this, at time of file transfer, be able to actualize high throughput of network, network can be released to early stage even case of file transfer of large capacity.

simple explanation Figure 1 of drawing is configuration example of computer which is connected to ATM network with one embodiment of this invention and block diagram which shows flow of file data.

Figure 2 is configuration example of computer which is connected to ATM network with other one embodiment of

ファイルデータの流れを示すブロック図である。

図 3 は、この発明の他の一実施形態による ATM ネットワークに接続される計算機の構成例とファイルデータの流れを示すブロック図である。

図 4 は、図 1 に示す構成におけるファイルデータの転送先での処理の手順を示す流れ図である。

図 5 は、図 1 に示す構成におけるファイルデータの転送元での処理の手順を示す流れ図である。

図 6 は、図 1 に示す構成におけるファイルデータの送信から受信までの処理の手順を示す流れ図である。

図 7 は、本発明の一実施形態による ATM ファイル転送方法の手順を説明した図である。

図 8 は、図 7 に示す ATM ファイル転送方法を実現するための装置の構成を示すブロック図である。

図 9 は、ATM スイッチ & 2 次サーバ D100, ユーザ端末 D200, コンテンツサーバ D300, ATM スイッチ D400 を ATM で接続した場合の構成例を示す図である。

図 10 は、コンテンツサーバ D300 から ATM スイッチ & 2 次サーバ D100 に大容量ファイルデータを一括転送する際のプロトコルスタックとデータの流れを示す図である。

図 11 は、ユーザ端末 D200 から ATM スイッチ & 2 次サーバ D100 に対してランダムアクセスする際のプロトコルスタックとデータの流れを示す図である。

図 12 は、本発明を、ハードディスクに蓄積された大容量ファイルに同時にアクセスする場合に適用した実施形態を説明するための図である。

図 13 は、本発明のデータ転送方法によるデータの転送手順を説明するための図である。

図 14 は、UDP パケットのフォーマットを示す図である。

図 15 は、ATM/AAL-5 CPCS-PDU のフォーマットを示す図である。

図 16 は、本発明によるデータ転送シーケンスを示す図である。

図 17 は、本発明におけるデータ転送装置の構

this invention and block diagram which shows the flow of file data .

Figure 3 is configuration example of computer which is connected to ATM network with other one embodiment of this invention and block diagram which shows the flow of file data .

As for Figure 4 , it is a flowchart which shows protocol of treatment with forwarding destination of file data in configuration which is shown in Figure 1 .

As for Figure 5 , it is a flowchart which shows protocol of treatment in transfer origin of file data in configuration which is shown in Figure 1 .

As for Figure 6 , it is a flowchart which shows protocol of treatment from reception from transmission of file data in configuration which is shown in Figure 1 .

Figure 7 is figure which explains protocol of ATM file transfer method with one embodiment of this invention .

Figure 8 is block diagram which shows configuration of device in order to actualize ATM file transfer method which is shown in Figure 7 .

Figure 9 is figure which shows configuration example when ATM switch & secondary server D100, user terminal D200, contents server D300, ATM switch D400 is connected with ATM .

Figure 10 , when from contents server D300 lumping together transferring large capacity file data in ATM switch & secondary server D100 is figure which shows flow of protocol stack and the data .

Figure 11 when random access doing from user terminal D200 vis-a-vis ATM switch & secondary server D100, is figure which shows flow of protocol stack and data .

Figure 12 , when this invention , simultaneously to large capacity file which compilation is done access it does in hard disk , is figure in order to explain embodiment which is applied.

Figure 13 is figure in order to explain transfer protocol of the data with data transfer method of this invention .

Figure 14 is figure which shows format of UDP packet .

Figure 15 is figure which shows format of ATM /AAL-5CPCS\*PDU .

Figure 16 is figure which shows data transfer sequence with this invention .

As for Figure 17 , it is a figure which shows configuration

成例を示す図である。

example of data transfer device in this invention .

図 18 は、本発明によるアドレス形態の一例を示す図である。

Figure 18 is figure which shows one example of address form with the this invention .

図 19 は、本発明によるアドレス形態の一例を示す図である。

Figure 19 is figure which shows one example of address form with the this invention .

図 20A は ATM ネットワークアダプタカードを装備した汎用計算機アーキテクチャを示すブロック図であり、図 20B は従来のファイル転送プロトコルを用いて大容量ファイルを受信する際のデータの流れを示す流れ図である。

As for Figure 20 A with block diagram which shows general purpose computer architecture which equips the ATM network adapter card , Figure 20 B when receiving large capacity file making use of conventional file transfer protocol , is flowchart which shows flow of data .

図 21 は、ATM リンクを用いたファイル転送プロトコルのプロトコルスタックとそれを実行するハードウェアをまとめた図である。

Figure 21 protocol stack of file transfer protocol which uses ATM link is figure which collected hardware which executes that.

図 22 は、従来の ATM リンクを用いた場合のデータ転送の手順と、それを実現

Figure 22 , actualizes protocol of data transfer when conventional ATM link is used and, that

するための装置構成の概略図である。

It is a conceptual diagram of equipment configuration in order to do.

図 23 は、従来の送信側及び受信側計算機間における TCP のフローコントロール

As for Figure 23 , conventional transmitting side and TCP in between called side computer [furookontoroo ]

ルを示すタイムチャートを示す図である。

It is a figure which shows time chart which shows jp11 .

発明を実施するための最良の形態

preferred embodiment in order to execute invention

図 1 は、第 1 又は第 3 の記憶媒体としてのハードディスク、第 2 又は第 4 の記

As for Figure 1 , hard disk , second as storage media of first or third or description of 4 th

憶媒体としての半導体メモリ (DRAM: ダイナミックランダムアクセスメモリ

<seq>DRAM :dynamic random access memory semiconductor memory as \* media

; Dynamic Random Access Memory) から成るホストメモリを用いた装置構成を有し、本発明によるファイル転送方法を実施する ATM ネットワークに接続される計算機の構成例とファイルデータの流れを示す図である。図 1 に示す各構成は、ホスト CPU・F1、DRAM 等で構成されるホストメモリ F2、高速の汎用バスである PCI バス F3、CPU と PCI バスを結ぶホスト-PCI ブリッジ F4、物理レイヤから AAL レイヤまでを終端する ATM ネットワークアダプタカード (あるいはネットワークインターフェースカード) F5、ハードディスク F6、及び ATM ネットワーク F10 である。次に上記の構成において、ATM-LAN 等の ATM ネットワーク F10 から

It is a configuration example of computer which is connected to ATM network which possesses equipment configuration which uses host memory which consists of Dynamic random access memory ) executes file transfer method with this invention and a figure which shows flow of file data . Each configuration which is shown in Figure 1 is ATM network adapter card (Or network interface face card ) F5, hard disk F6, and ATM network F10 which to AAL layer terminal are done with such as host CPU \*F1, DRAM from the host - PCI bridge F4, physical layer which ties PCI bus F3, CPU and PCI bus which are a general purpose bus of the host memory F2, high speed which configuration is done. Next in above-mentioned configuration , from ATM - LAN or other ATM network F10

大容量ファイルが伝送される場合の受信側の手順について、図1及び図4を参照

You refer to Figure 1 and Figure 4 large capacity file concerning protocol of called side when transmission it is done

して説明する。なお、図4は、図1に示すファイルの受信手順の流れを示す流れ図である。まず転送されて来たファイルは、ATMネットワークF10からATMネットワークアダプタカードF5に伝送される(図4のステップB1~B2)。ATMネットワークアダプタカードF5内ではまず光信号の終端やS/P変換、セル同

Doing, you explain. Furthermore, Figure 4 reception protocol of file which is shown in Figure 1 is flowchart which shows flow. First file which is transferred from ATM network F10 transmission is done in ATM network adapter card F5, (step B1~B2 of Figure 4 ). Inside ATM network adapter card F5 first terminal and S/P conversion and cell of light signal same

期といった物理レイヤ(PHY)の終端を行い、ATMレイヤ、AALへと順次に上位レイヤ側にデータが転送される。ATMレイヤでは主にVCI/VPIによるセルの多重分離が行われ、AALレイヤではセルの48バイトのペイロードを結合してCPCS-PDUを構成し、CRCや長さのチェックを行ってからC

terminal of physical layer (PHY ) such as period is done, to ATM layer , AAL the data is transferred to upper position layer side in sequential . With ATM layer demultiplexing of cell to be done mainly with the VCI /VPI , with AAL layer connecting payload of 48 byte of cell , configuration to do CPCS- PDU , after doing check of CRC and the length , C

PCS-PDUのペイロードをユーザデータとして取り外す。

You remove payload of PCS - PDU as user data .

ATM、AAL レイヤを終端して取り出されたファイルのデータは PCI バスのバス I/F チップ(Bus I/F)から PCI バス F3 へ転送される。

terminal doing ATM , AAL layer , data of file which is removed is transferred from bus I/F chip (Bus I/F ) of PCI bus to PCI bus F3.

このとき、ATM アダプタカード F5 のバス I/F チップはファイルデータの転送先としてホスト PCI ブリッジ F4 を選択する。

At time of this , bus I/F chip of ATM adapter card F5 selects host PCI bridge F4 as forwarding destination of file data .

ネットワークアダプタカード F5 から転送されたデータはホスト-PCI ブリッジ F4 を介してホストメモリ F2 へ高速に転送される(ステップ B2~B3)。

data which was transferred from network adapter card F5 through host -PCI bridge F4, is transferred to high speed to host memory F2 (step B2~B3) .

ホストメモリ F2 に一括で蓄えられたデータは CP CS-PDU ペイロードのままである。

data which in host memory F2 is stored with bundle it continues to be a CPCS-PDU payload .

CPCS まではアダプタカード F5 内のハードウェアで終端されるため、高速処理が可能であり、PCI バス F3、ホスト-PCI ブリッジ F4、DRAM-F2 のデータ転送はすべてハードウェアであるた

Because to CPCS terminal it is done with hardware inside the adapter card F5, fast processing being possible, as for data transfer of PCI bus F3, host -PCI bridge F4, DRAM -F2 because it is a hardware entirely, there is not restriction of

め、ソフトウェア処理による転送速度の制限がない。

ファイルデータの転送が終了した後、ATM は ATM リンクを終了し、ネットワーク F10 を解放する。

ホストメモリ F2 に一括で蓄えられたデータは CP CS-PDU ペイロードの状態であり、圧縮ファイルである場合の解凍処理等は施されていない。

これらの処理は CPU によるソフトウェア処理であるため、従来の方によってデータ転送中に同時に処理するとファイル転送のスループットを著しく低下させてしまう。

そこで、本発明では、一旦高速でホストメモリ F2 に圧縮された状態のままファイル転送してしまい、ネットワーク F10 を解放した後で CPU・F1 が解凍などの処理を行う。

ネットワーク F10 を解放した後、CPU・F1 は、ホストメモリ F2 に一括で蓄えられた CPCS-PDU ペイロードの状態データに対して通信リンクを確立する(ステップ B4)。

そして、CPU・F1 は、CPCS-PDU ペイロードの状態データに対して、フレーミング(ステップ B5)、プロトコル終端(ステップ B6)、及び必要に応じてデータ解凍(ステップ B7)の処理を行い、転送すべきデータの中身を取り出し、取り出したデータをハードディスク F6 へ格納する(ステップ B8)。

このように、最終的に、送られてきたファイルは、ハードディスク、光磁気ディスク、磁気テープ等の大容量記憶媒体(この例ではハードディスク F6)に蓄積されるが、ハードディスク F6 等の連続書き込み、読み出しの最大速度は汎用バス F3 の最大転送速度に比べ、十分低速である。

したがって、ホストメモリ F2 からハードディスク F6 への転送は、PCI バス F3 の最大転送速度を連続して使用した転送とはならず、ハードディスク F6 の磁気記憶媒体への低速の書き込み速度に合わせた、ハードディスクが備えるバスインターフェイス内のバッファメモリへの断続的な転送として実行されることになる。

ホストメモリ F2 からハードディスク F6 へのデータの書き込みの休止期間に CPU・F1 はホストメモリ F2 内のファイルデータに対する解凍などの処理を行うことができるので、これらの処理によるハードディスク F6 の書き込み速度の低下は発生しない。

forwarding rate in software treatment.

After transfer of file data ends, ATM ends ATM link , releases network F10.

As for data which in host memory F2 is stored with bundle with the state of CPCS-PDU payload , as for thawing treatment etc when it is a compressed file it is not administered.

These treatments decrease because it is a software treatment with the CPU , when it treats simultaneously in data transfer with conventional method , the throughput of file transfer considerably.

Then, with this invention , while it is a state which once was compressed to host memory F2 with high speed file it transfers, after releasing the network F10, CPU \*F1 treats thawing or other .

After releasing network F10, as for CPU \*F1, communication link is established vis-a-vis state data of CPCS-PDU payload which in host memory F2 is stored with bundle (step B4 ).

And, CPU \*F1, flaming (step B5 ), protocol terminal (step B6 ), and treats according to need data thawing (step B7 ) vis-a-vis state data of CPCS-PDU payload , removes contents of data which it should transfer, houses data which is removed to hard disk F6 (step B8 ).

this way, finally , file which is sent compilation is done in hard disk , magneto-optical disk , magnetic tape or other large capacity storage media (With this example hard disk F6 ), but maximum speed of hard disk F6 or other continuous writing , reading is fully low speed in comparison with maximum forwarding rate of general purpose bus F3.

Therefore, from host memory F2 as for transfer to hard disk F6, continuing the maximum forwarding rate of PCI bus F3, it did not become with transfer which you use, adjusted to writing speed of low speed to magnetic memory media of hard disk F6, it means to be executed as discontinuous transfer to buffer memory inside bus interface which hard disk has.

Because to do thawing or other treatment for file data inside host memory F2 it is possible CPU \*F1 to respiratory pause of writing of data to hard disk F6 from host memory F2, decrease of writing speed of hard disk F6 does not occur in these treatments.

次に図1に示す構成において、ホストCPU1からATMネットワークF10へ大容量ファイルを伝送する送信側の手順について図1及び図5を参照して説明する。

なお、送信側の手順は、受信側の手順と逆向きの流れとなるものであり、この場合、図1に示す矢印を付けた受信側の手順と逆向きの流れとなる。

まず、転送すべきファイルをハードディスクF6に格納する(ステップA1)。

CPU\*F1は、ハードディスクF6に格納されているデータを読み出しながら、データ圧縮(ステップA2)、プロトコル終端(ステップA3)及びフレーミング(ステップA4)を行って、CPCS-PDUペイロードの状態データをホストメモリF2に記憶する(ステップA5)。

次に、CPU\*F1はATMリンクを確立し(ステップA6)、ネットワークアダプタカードF5に対してファイルの転送を指令する(ステップA7)。

ネットワークアダプタカードF5は、PCIバスF3を介してホストメモリF2に記憶されているCPCS-PDUペイロードの状態データを直接読み出して、AALレイヤでSAR-PDUに分割し、ATMレイヤでATMセルとして、物理レイヤによってATMネットワークF10へ伝送する(ステップA8)。

そして、ファイルの伝送が終了したところでCPU\*F1は通信リンクを解放する。

このようにファイルの転送時には、データ圧縮、プロトコル終端及びフレーミングの処理を行った後のデータをホストメモリF2に記憶し、ホストメモリF2に記憶されたデータをネットワークアダプタF5から直接読み出して、ネットワークF10へ伝送するようにしたので、ハードディスクF6からの読み出し処理やホストCPU\*F1によるデータ処理が、ネットワークアダプタF5の伝送速度を低下させることはない。

図6は、図4に示す本発明のファイル転送方法による転送先での手順と、図5に示す本発明のファイル転送方法による転送元での手順を、一連の処理として行う場合の処理の流れを示したものである。

図6において、ステップC1~C8は図5に示すステップA1~A8の処理に、ステップC8~C15は図4に示すステップB1~B8の処理に、それぞれ対応している。

Referring to Figure 1 and Figure 5 is done concerning protocol of transmitting side which to ATM network F10 in configuration which is shown next in Figure 1, from the host CPU 1 large capacity file transmission, you explain.

Furthermore, protocol of transmitting side being something which becomes the protocol of called side and flow of reverse direction, in case of this, becomes protocol of called side which attaches arrow which it shows in Figure 1 and flow of reverse direction.

First, file which it should transfer is housed in hard disk F6 (step A1).

CPU \*F1 data which is housed in hard disk F6 reading, data compression (step A2), protocol terminal (step A3) and doing flaming (step A4), with state of CPCS-PDU payload remembers data in host memory F2 (step A5).

Next, CPU \*F1 establishes ATM link and (step A6), transfers file vis-a-vis network adapter card F5 command, (step A7).

network adapter card F5, through PCI bus F3, reading \*, divides data of the state of CPCS-PDU payload which is remembered in host memory F2 into SAR-PDU directly with AAL layer, with ATM layer with physical layer transmission doesto ATM network F10 as ATM-cell, (step A8).

And, being at point where transmission of file ends, CPU \*F1 releases communication link.

this way when transferring file, after treating data compression, protocol terminal and flaming, storage to do data in host memory F2, data which is remembered in host memory F2 from network adapter F5 reading \*, transmission is done directly to network F10, because it required, with reading treatment and host CPU \*F1 from hard disk F6 data processing, transmission speed of [network adapter] F5 there are not times when it decreases.

Figure 6 with file transfer method of this invention which is shown in Figure 4 in transfer origin is something which shows flow of treatment when it does protocol, as consecutive treatment with the protocol with forwarding destination and file transfer method of this invention which is shown in Figure 5.

In Figure 6, as for step C1~C8 in treatment of step A1~A8 which is shown in Figure 5, step C8~C15 corresponds to treatment of the step B1~B8 which is shown in Figure 4, respectively.

図 6 に示すようにファイルの転送元及び転送先の両方で、ホスト CPU・F1 を介さないホストメモリ F2 とネットワークアダプタカード F5 間のデータの転送を行うことで、ファイルの転送のスループットをより大きくすることができる。

次に、図 2 を参照して、図 1 を参照して説明した実施形態の変形例について説明する。

図 2 に示す本発明の実施形態は、第 1 又は第 3 の記憶媒体としてハードディスク、第 2 又は第 4 の記憶媒体としてホストメモリの代わりにネットワークアダプタカードと同じ汎用データバス上に高速入出力可能な半導体メモリによるメモリボードを配備した実ものである。

図 2 に示す各構成は、ホスト CPU・G1、DRAM 等で構成されるホストメモリ G2、高速の汎用バスである PCI バス G3、CPU と PCI バスを結ぶホスト-PCI ブリッジ G4、物理レイヤから AAL レイヤまで終端する ATM ネットワークアダプタカード G5、ハードディスク G6、DRAM 等で構成される PCI バス上のメモリボード G7、ATM ネットワーク G10 である。

なお、図 1 に示すものと同一の数字を有する符号を持つ構成は、図 1 に示す対応する構成と同様のものである。

次に図 2 に示す構成において、ATM-LAN 等の ATM ネットワークから大容量ファイルが転送される場合の手順を説明する。

ファイルは ATM ネットワーク G10 から ATM ネットワークアダプタカード G5 に伝送される。

ATM ネットワークアダプタカード G5 内では物理レイヤ、ATM レイヤ、AAL へと順次に終端され、上位レイヤ側にデータを転送する。

AAL レイヤを終端して取り出されたファイルのデータは PCI バスのバス I/F チップ (PCI バスコントローラ) から PCI バス G3 へ転送される。

このとき、ATM アダプタカード G5 のバス I/F チップは図 1 の場合とは異なり、同一の PCI バス G3 上のメモリボード G7 をファイルデータの転送先ターゲットとして指定する。

ネットワークアダプタカード G5 とメモリボード G7 は同-PCI バス G3 のエージェントであり、ATM ネットワーク G10 から高速伝送されたファイルデータをスループット落とさずにメモリボード G7 に書き込むことができる。

また、図 1 の構成に比べ、ホストメモリ G2 を利用

As shown in Figure-6 , with transfer origin of file and both of forwarding destination , by fact that it transfers data between the host memory F2 and network adapter card F5 which do not mind host CPU \*F1, throughput of transfer of file can be made larger.

Next, referring to Figure 2 , referring to Figure 1 , you explain concerning modified example of embodiment which you explain.

embodiment of this invention which is shown in Figure 2 as network adapter card is actual ones which on same general purpose data bus dispose memory board with high speed input-output possible semiconductor memory in place of host memory hard disk , second or as storage media of 4 th as storage media of first or third .

Each configuration showing in Figure 2 is with such as host CPU \*G1, DRAM memory board G7, ATM network G10 on the PCI bus which configuration is done with such as PCI bus G3, CPU which is a general purpose bus of host memory G2, high speed which configuration is done and from host -PCI bridge G4, physical layer which ties the PCI bus to AAL layer ATM network adapter card G5, hard disk G6, DRAM which terminal is done.

Furthermore, it is something which is similar to configuration which as those which are shown in Figure 1 shows configuration which has code which possesses same numeral , in Figure 1 and corresponds.

protocol when large capacity file is transferred from ATM -LAN or other ATM network in configuration which is shown next in Figure 2 , is explained.

file from ATM network G10 transmission is done in ATM network adapter card G5.

Inside ATM network adapter card G5 to physical layer , ATM layer , AAL terminal it is done in sequential , transfers data to upper position layer side.

terminal doing AAL layer , data of file which is removed is transferred from bus I/F chip (PCI bus controller ) of PCI bus to PCI bus G3.

Appoints memory board G7 on same PCI bus G3 at time of this , the bus I/F chip of ATM adapter card G5 unlike case of Figure 1 , as forwarding destination target of the file data .

With same -PCI bus G3 agent , throughput without dropping file data which high speed transmission is done from ATM network G10, can network adapter card G5 and memory board G7 write to memory board G7.

In addition, you can call configuration which faces to file

しない分、大容量のファイル転送に向けた構成と言える。

この構成において、ファイルデータの転送が終了した後、ATM は ATM リンクを終了し、ネットワークを解放する。

PCI バス G3 上のメモリボード G7 に蓄えられたデータは CPCS-PDU ペイロードの状態であり、圧縮ファイルである場合の解凍処理等は施されていない。

図 1 の場合と同様に、これらのファイルデータに対する処理は、ハードディスクへの書き込み速度とバスの転送速度に差を利用して、ハードディスクに転送する前に逐次 CPU・G1 で処理を行う。

図 3 は、図 2 の汎用データバス上にブリッジチップを設けることで、汎用データバスを拡張して汎用データバスをもう一つ新設し、ネットワークアダプタカードとメモリボードをこの新設された汎用データバスに接続した他の実施形態を示す図である。

図 3 に示す各構成は、ホスト CPU・H1、DRAM 等で構成されるホストメモリ H2、高速の汎用バスである PCI バス H3a、H3b、CPU と PCI バスを結ぶホスト-PCI ブリッジ H4、物理レイヤから AAL レイヤまで終端する ATM ネットワークアダプタカード H5、バスインターフェイスを内蔵したハードディスク H6、DRAM 等で構成される PCI バス上のメモリボード H7、2つの PCI バスを接続する PCI-PCI ブリッジ H8、ブリッジにより新たに拡張された PCI バス H9、そして ATM ネットワーク H10 である。

次に、図 3 を参照して、この図に示す構成において ATM-LAN 等の ATM ネットワーク H10 から大容量ファイルが転送される場合の手順について説明する。

ファイルは ATM ネットワーク H10 から ATM ネットワークアダプタカード H5 に伝送される。

ATM ネットワークアダプタカード H5 内では物理レイヤ、ATM レイヤ、AAL へと順次に終端され、上位レイヤ側にデータを転送する。

AAL レイヤを終端して取り出されたファイルのデータは PCI バスのバス I/F チップ (PCI バスコントローラ) から PCI バス H3b へ転送される。

このとき、この PCI バス H3b は図 1、2 の場合とは異なり、ホストの PCI ブリッジで CPU やホストメモリに接続される元から存在している PCI バス H3a ではなく、この図にあるように PCI-PCI ブリッ

transfer of amount and large capacity which do not utilize host memory G2 in comparison with configuration of Figure 1 .

In this configuration , after transfer of file data ends, ATM ends the ATM link , releases network .

As for data which is stored in memory board G7 on PCI bus G3 with the state of CPCS-PDU payload , as for thawing treatment etc when it is a compressed file it is not administered.

In same way as case of Figure 1 , treatment for these file data in writing speed to hard disk and forwarding rate of bus making use of difference, before transferring to hard disk , treats with these sequential CPU \*G1.

By fact that bridge chip is provided on general purpose data bus of Figure 2 , expanding general purpose data bus , general purpose data bus another new construction it does Figure 3 , it is a figure which shows other embodiment which is connected to general purpose data bus which network adapter card and memory board this new construction is done.

Each configuration which is shown in Figure 3 is PCI bus H9, and ATM network H10 which are expanded anew by PCI -PCI bridge H8, bridge which memory board H7, 2 PCI bus on PCI bus which configuration is done with such as PCI bus H3a, H3b, CPU which is a general purpose bus of the host memory H2, high speed which configuration is done and from host -PCI bridge H4, physical layer which ties PCI bus to AAL layer hard disk H6, DRAM which builds in ATM network adapter card H5, bus interface which terminal is done connects with such as host CPU \*H1, DRAM .

Next, referring to Figure 3 , you explain concerning protocol when large capacity file is transferred from ATM -LAN or other ATM network H10 in configuration which it shows in this figure.

file from ATM network H10 transmission is done in ATM network adapter card H5.

Inside ATM network adapter card H5 to physical layer , ATM layer , AAL terminal it is done in sequential , transfers data to upper position layer side.

terminal doing AAL layer , data of file which is removed the PCI bus H3b \* is transferred from bus I/F chip (PCI bus controller ) of PCI bus .

In order at time of this , this PCI bus H3b is not PCI bus H3a which exists from cause of being connected to CPU and host memory with PCI bridge of host unlike case of Figure 1 , 2, for there to be a this figure, anew it is a PCI bus H3b which



ジH8で新たに構成されたPCIバスH3bである。

この新設されたPCIバスH3bは元からあるPCIバスH3aとは独立である。

新設されたPCIバスH3bにはATMアダプタカードH5と図2に示すメモリボードG7と同様なメモリボードH7が接続され、アダプタカードH5のバスI/Fチップは図2の場合と同様、同一のPCIバスH3b上のメモリボードH7をファイルデータの転送先ターゲットとして指定する。

ネットワークアダプタカードH5とメモリボードH7は新設された同PCIバスH3bのエージェントであり、ATMネットワークH5から高速伝送されたファイルデータをスループットを落とさずにメモリボードH7に書き込むことができる。

また、図2の場合と比べるとアダプタカードH3、メモリボードH7が搭載されるPCIバスH3bには他にPCIボード(エージェント)が接続されないため、バスを独占できる。

通常、元のPCIバスH3aにはグラフィックスボードやキーボード、その他の周辺機器のインタフェースボード等が接続されることがあり、ATMネットワークアダプタカードとメモリボードがバスを完全に占有できない場合がある(図20A参照)。

この場合、他のPCIエージェントによるバスリクエストにより、PCIバスのデータ転送のスループットが低下するときがあるが、図3のようにアダプタカードH5とメモリボードH7が搭載されるバスを他のPCIエージェントとは別のバスにすることにより、アダプタカードH5とメモリボードH7との間の高いスループットを保証することができる。

特にバスを占有することができるので大容量ファイルの転送においても高スループットが実現できる。

上記の構成において、ファイルデータの転送が終了した後、ATMはATMリンクを終了し、ネットワークを解放する。

PCI上のメモリボードH7に蓄えられたデータはCPCS-PDUペイロードの状態であり、圧縮ファイルである場合の解凍処理等は施されていない。

図1の場合と同様に、これらのファイルデータに対する処理は、ハードディスクへの書き込み速度とバスの転送速度に差を利用して、ハードディスクH6に転送する前に逐次CPU・H1で処理を行う。

このとき、メモリボードH7内のデータはPCI-PCIブリッジH8を介してCPU・H1が搭載される側の

configuration is done with the PCI-PCI bridge H8.

As for PCI bus H3b which this new construction is done PCI bus H3a which is from origin is independence.

memory board H7 which is similar to memory board G7 which is shown in ATM adapter card H5 and Figure 2 is connected by PCI bus H3b which new construction is done, the bus I/F chip of adapter card H5 similarity to case where it is a Figure 2, appoints memory board H7 on same PCI bus H3b as forwarding destination target of file data.

network adapter card H5 and memory board H7 with same -PCI bus H3b agent which new construction is done, throughput without dropping, it can write file data which the high speed transmission is done to memory board H7 from ATM network H5.

In addition, when you compare with case of Figure 2, because the PCI board (agent) is not connected to other things in PCI bus H3b where adapter card H3, memory board H7 is installed, bus can be monopolized.

Usually, there are times when graphics board and interface board etc of keyboard, other peripheral equipment are connected in original PCI bus H3a, there are times when ATM network adapter card and memory board cannot possess bus completely, (Figure 20 A reference).

In case of this, when throughput of data transfer of PCI bus decreases with other PCI agent with bus request, it is, but like the Figure 3 adapter card H5 and throughput whose between of memory board H7 is high can be guaranteed from adapter card H5 and bus where memory board H7 is installed other PCI agent by making another bus.

Especially, because it is possible to possess bus, it can actualize high throughput at time of transferring large capacity file.

In above-mentioned configuration, after transfer of file data ends, the ATM ends ATM link, releases network.

As for data which is stored in memory board H7 on PCI with the state of CPCS-PDU payload, as for thawing treatment etc when it is a compressed file it is not administered.

In same way as case of Figure 1, treatment for these file data in writing speed to hard disk and forwarding rate of bus making use of difference, before transferring to hard disk H6, treats with these sequential CPU \*H1.

At time of this, data inside memory board H7 through the PCI-PCI bridge H8, is transferred to PCI bus H3a side where

PCIバス H3aに転送される。

PCIバス H3aに転送されたデータはCPU・H1に渡って処理された後、CPU・H1が搭載される側のPCIバス H3aに接続されるハードディスク H6に蓄積される。

このように図 1~3 のいずれの構成においても高速通信の ATM ネットワークアダプタカードからの高い転送速度を制限しないように、CPU による処理を施さずに DRAM 等で構成されるメモリに転送する。

これにより、ネットワークを早期に解放でき、高速な ATM ネットワークを有効利用できる。

最終的な蓄積先であるハードディスク等の磁気記憶媒体は連続書き込み速度が遅いため、高速の汎用バスを断続的に使って転送せざるを得ない。

そこでハードディスクへのデータ転送が途切れている時間を利用して解凍等の処理を行う。

次に、上記の図 1~図 6を参照して説明した本発明による各実施形態を、ATM ネットワーク内で通信速度が異なるデータ転送が介在する場合に応用するときの実施形態について説明する。

図 7において、汎用計算機 GPC は、CPU(中央演算装置)、ハードディスクやホストメモリ等の大容量記憶媒体 MSD、複数の ATM アダプタ、これらの間のデータ転送を行う汎用データバス GPD B を有するものであり、例えば図 1 に示す実施形態では受信側で用いるホスト CPU・F1 とその周辺装置に対応するものである。

この場合、汎用計算機 GPC は ATM スイッチ&2 次サーバ SVR として機能し、ATM のアダプタ A DPI,ADP2 を介してそれぞれ端末 T1,T2 が接続される。

この端末 T2 が、図 1 に示す実施形態では、送信側で用いるホスト CPU・F1 とその周辺装置に対応するものである。

したがって、図 7 に示す実施形態は、図 1 に示す受信側で用いるホスト CPU・F1 とその周辺装置に、さらにネットワークアダプタカードを介して他の端末(ここでは端末 T1)を接続した構成に対応している。

図 7 に示す実施形態では、例えば以下に示す手順 1~9 によって通信を実行することができる。

CPU \*H1 is installed.

data which was transferred to PCI bus H3a after being treated over CPU \*H1, compilation is done in hard disk H6 which is connected to the PCI bus H3a side where CPU \*H1 is installed.

this way in order not to restrict forwarding rate where is high from ATM network adapter card of high speed communication regarding whichever configuration of Figure 1 ~3, without administering treatment with CPU , it transfers to memory which with such as DRAM configuration is done.

Because of this, be able to release network to early stage , effective use is possible high speed ATM network .

Because continuous writing speed is slow, using general purpose bus of high speed in discontinuous ,you must transfer hard disk or other magnetic memory media which is ahead final compilation .

It treats thawing or other then making use of time when data transfer to the hard disk has broken off.

Next, referring to above-mentioned Figure 1 ~Figure 6 , when with this invention which you explain each embodiment , communication speed different data transfer lies between inside ATM network when applying, you explain concerning embodiment .

In Figure 7 , general purpose computer GPC , CPU (central processing unit ), being something which possesses general purpose data bus GPDB which does data transfer ATM adapter , at these of hard disk and the host memory or other large capacity storage media MSD , plural time, with embodiment which it shows in for example Figure 1 is something which corresponds to host CPU \*F1 and peripheral which are used with the called side .

In case of this , general purpose computer GPC functions as ATM switch & secondary server SVR , through the adapter ADP1, ADP2 of ATM , terminal T1, T2 is connected respectively.

this terminal T2, with embodiment which is shown in Figure 1 , is something which corresponds to host CPU \*F1 and peripheral which are used with the transmitting side .

Therefore, as for embodiment which is shown in Figure 7 , furthermore through network adapter card to host CPU \*F1 and peripheral which are used with the called side which is shown in Figure 1 , it corresponds to configuration which connects other terminal (Here terminal T1 ) .

With embodiment which is shown in Figure 7 , communication can be executed with protocol 1~9 which is shown below for example .

〈手順1〉 ATMセルの VCI,VPI と汎用データバス GPDB 内のアドレスを対応させることによって、汎用データバス GPDB を介して複数の ATM アダプタ間で ATM セルを転送するとともに、アダプタ ADP1,ADP2 からのデータを大容量記憶媒体 MSD に蓄積させることにより、汎用計算機 GPC を ATM スイッチ&2 次サーバ SVR として機能させる。

〈手順2〉 端末 T1 が端末 T2 から大容量ファイルを読み出すためのファイル転送を行う際、端末 T1 から端末 T2 に対するリンク設定のためのシグナリングセルに含まれる情報要素内容により、ATM スイッチ&2 次サーバ SVR が大容量ファイル転送のための ATM リンクであることを認識する。

〈手順3〉 ATM スイッチ&2 次サーバ SVR が端末 T2 になりかわり、端末 T1 との間でシグナリングすることにより、端末 T1 と大容量記憶媒体 MSD との間でリンク 1 を設定する。

〈手順4〉 リンク 1 により、端末 T1 から大容量ファイルに対するランダムアクセスコマンドが、端末 T1 から ATM スイッチ&2 次サーバ SVR へ送られて、大容量記憶媒体 MSD の I/F コントローラ内の FIFO にスタックされる。

〈手順5〉 ATM スイッチ&2 次サーバ SVR が端末 T1 になりかわり、端末 T2 との間でシグナリングすることにより、大容量記憶媒体 MSD と端末 T2 との間にリンク 2 を設定する。

〈手順6〉 リンク 2 の確立後、端末 T1 がアクセスする大容量ファイルをリンク 2 を使って端末 T2 からシケンシャルに読み出し、端末 T1 に転送する代わりに、汎用データバス GPDB を介して大容量記憶媒体 MSD に一括転送する。

〈手順7〉 大容量記憶媒体 MSD へのファイル転送が完了した後、シグナリングによって端末 T2 と ATM スイッチ&2 次サーバ SVR との間のリンク 2 を解放する。

〈手順8〉 リンク 2 の解放後、大容量記憶媒体 MSD の I/F コントローラの FIFO に蓄積されている大容量ファイルへのランダムアクセスコマンドを順次実行し、リンク 1 を介して、大容量ファイルデータを ATM スイッチ&2 次サーバ SVR から端末 T1 へランダムに転送する。

〈手順9〉 端末 T1 からファイルへのランダムアクセスが終了後に、ATM スイッチ&2 次サーバ SVR と端末 T1 との間でリンク解放のためのシグナリングを行い、大容量ファイルのランダムなデータ転送を終了する。

VCI, VPI of {protocol 1 } ATM-cell and address inside general purpose data bus GPDB by factthat it corresponds, through general purpose data bus GPDB, as ATM-cell is transferredbetween ATM adapter of plural , it functions data from adapter ADP1, ADP2 by compilation doing, with general purpose computer GPC as ATM switch &secondary server SVR in large capacity storage media MSD .

When transferring because {protocol 2 } terminal T1 reads out large capacity file from the terminal T2 file , fact that ATM switch &secondary server SVR is ATM link for large capacity file transferwith data element content which is included in Signa ring cell for link settingfrom terminal T1 for terminal T2, is recognized..

{protocol 3 } ATM switch &secondary server SVR becomes terminal T2 and changes, sets link 1 between the terminal T1 and large capacity storage media MSD by Signa ring doing between terminal T1.

With {protocol 4 } link 1, random access command from terminal T1 for large capacity file , being sentto ATM switch &secondary server SVR from terminal T1, stack it is done in FIFO inside I/Fcontroller of large capacity storage media MSD .

{protocol 5 } ATM switch &secondary server SVR becomes terminal T1 and changes, sets link 2 between the large capacity storage media MSD and terminal T2 by Signa ring doing between terminal T2.

After establishing {protocol 6 } link 2, terminal T1 using link 2, through general purpose data bus GPDB instead of from terminal T2 transferring to reading , terminal T1 in the sequential , lumps together transfers large capacity file which access is done in large capacity storage media MSD .

After file transfer to {protocol 7 } large capacity storage media MSD completes, link 2 between the terminal T2 and ATM switch &secondary server SVR is released with Signa ring .

After releasing {protocol 8 } link 2, random access command to large capacity file which compilation is done sequential is executed in FIFO of I/Fcontroller of large capacity storage media MSD ,through link 1, large capacity file data from ATM switch &secondary server SVR is transferred to the random to terminal T1.

From {protocol 9 } terminal T1 random access to file after ending, does Signa ring for link release between ATM switch &secondary server SVR and terminal T1, ends random data transfer of large capacity file .

次に、上記手順を実現するための具体例について説明する。

図 8 は同実施形態による ATM ファイル転送方法を実現するための装置の構成を示すブロック図であり、同装置は汎用計算機による ATM スイッチ&2 次サーバ D100 である。

同図に示す各構成は、ホスト CPU・C101、DRAM(ダイナミック RAM)等で構成されるホストメモリ C102、高速の汎用バスである PCI バス C103、ホスト CPU・C101 と PCI バス C103 を結ぶホスト-PCI ブリッジ C104、コンテンツサーバ(後述)側に接続する 155Mbps の ATM アダプタ C105、ハードディスク C106、ユーザ端末(後述)側に接続する 25Mbps の ATM アダプタ C107 である。

一方、図 9 は、図 8 に示す ATM スイッチ&2 次サーバ D100 とユーザ端末、コンテンツサーバ、ATM スイッチを ATM で接続した場合の構成例を示している。

図 9 に示す各構成は、図 8 に示した汎用計算機による ATM スイッチ&2 次サーバ D100、PC(パーソナルコンピュータ)などのユーザ端末 D200、コンテンツサーバ D300、コンテンツサーバ D300 と ATM スイッチ&2 次サーバ D100 とを結ぶ ATM スイッチ D400 である。

なお、ユーザ端末 D200 は CPU・D221 と ATM アダプタ D222 が PCI バスで接続されて構成されている。

また、コンテンツサーバ D300 は CPU・D331、ハードディスク D332、ATM アダプタ D333 がそれぞれ PCI バスで接続されて構成されている。

次に、上記構成を用いた ATM ファイル転送方法を説明する。

ここで、図 10 には図 9 のネットワークを用いてコンテンツサーバ D300 から ATM スイッチ&2 次サーバ D100 に大容量ファイルデータを一括転送する際のプロトコルスタックとデータの流れを示してある。

なお、同図において、SSCF は CO(コネクション)型サービスに固有な機能であるサービス依存コーディネーション機能、SSCOP は全ての CO サービスに共通な機能を規定するサービス依存コネクション型プロトコル、SAR はセル分割・組立サブレイヤである。

また、図 11 には、ユーザ端末 D200 から ATM スイッチ&2 次サーバ D100 に対してランダムアクセスする際のプロトコルスタックとデータの流

Next, you explain concerning embodiment in order to actualize the above-mentioned protocol .

As for Figure 8 with block diagram which shows configuration of device in order to actualize ATM file transfer method with same embodiment , the same equipment is ATM switch & secondary server D100 with general purpose computer .

Each configuration which is shown in same Figure is ATM adapter C107 of 25 Mbps which connect PCI bus C103 , host CPU \*C101 and PCI bus C103 which are a general purpose bus of host memory C102, high speed which configuration is done joining \* on ATM adapter C105, hard disk C106, user terminal (Later description) side of 155 Mbps which with such as host CPU \*C101, DRAM (dynamic RAM ) are connected on host -PCI bridge C104, contents server (Later description) side.

configuration example when ATM switch & secondary server D100 and user terminal , contents server , ATM switch which on one hand, as for the Figure 9, are shown in Figure 8 are connected with ATM has been shown.

Each configuration which is shown in Figure 9 is ATM switch & secondary server D100, PC ( [paasonarukonpyuuta ] ) or other user terminal D200, contents server D300, contents server D300 and the ATM switch D400 which ties ATM switch & secondary server D100 with general purpose computer which is shown in Figure 8 .

Furthermore, user terminal D200 is done CPU \*D221 and ATM adapter D222 being connected with PCI bus , configuration .

In addition, contents server D300 is done CPU \*D331, hard disk D332, ATM adapter D333 being connected respectively with PCI bus , configuration .

Next, ATM file transfer method which uses above-mentioned configuration is explained.

When here, in Figure 10 from contents server D300 lumping together transferring large capacity file data in ATM switch & secondary server D100 making use of network of Figure 9 flow of protocol stack and data is shown.

Furthermore, as for SSCF peculiar service dependence coordination function which is a function, as for SSCOP as for service dependence connection type protocol , SAR which stipulates common function in the all CO service it is a cell portion percentage & an assembly sub layer in CO (connection) type service in same Figure .

In addition, when random access doing from user terminal D200 vis-a-vis ATM switch & secondary server D100, protocol stack and data flow is shown to Figure 11 .

れを示してある。

まず、ユーザ端末 D200 はシグナリング(例えば Q.2931 といったプロトコル)によってコンテンツサーバ D300 の ATM アドレスへ ATM リンクを設定しようとする。

このとき、ATM スイッチ&2 次サーバ D100 は特定の VCI,VPI(VCI=5,VPI=0)をシグナリングセルと認識し、さらに Q.2931 のシグナリングセルで運ばれる情報要素内容により、ユーザ端末 D200 がコンテンツサーバ D300 の大容量ファイル(ハードディスク D332)へアクセスしようとしていることを知る。

そこで、ATM スイッチ&2 次サーバ D100 はコンテンツサーバ D300 を疑似し、ユーザ端末 D200 との間でシグナリングを行い、ユーザ端末 D200 と ATM スイッチ&2 次サーバ D100 のハードディスク C106 との間でリンク(25Mbps)を設定する。

すなわち本発明では、ユーザ端末 D200 とコンテンツサーバ D300 間に ATM リンクを設定するのではない。

そしてこのリンクによって、ユーザ端末 D200 から見ると、ATM スイッチ&2 次サーバ D100 が仮想的にコンテンツサーバとなる。

また、ユーザ端末 D200 からコンテンツサーバ D300 に向けて送られた大容量ファイルのランダムアクセスのためのコマンドは、ハードディスク C106 の I/F コントローラ(PCI バスコントローラ)内の FIFO(First In First Out memory:最初に入力された情報を最初に出力する先入れ先出し型のメモリで図示を省略してある)にスタックされる。

次に図 10 に示すように、ATM スイッチ&2 次サーバ D100 はユーザ端末 D200 になりかわり、シグナリング(Q.2931)によって、ATM スイッチ D400 を介してコンテンツサーバ D300 と 155Mbps の ATM リンクを設定する(図 10 の Cプレーンのデータの流れを参照)。

リンク確立後は、ATM スイッチ D400 を介して、ユーザ端末 D200 がアクセスするコンテンツサーバ D300 内の大容量ファイルをシーケンシャルに読み出し、ATM アダプタ C105 内で PHY(物理)、ATM,AAL5 の各レイヤを終端されたデータは CPCS-PDU(CS 共通部-プロトコルデータユニット)の形で PCI バス C103 に転送され、ホスト PCIブリッジ C104 を介して、一旦、ホストメモリ C102 に蓄積される。

このとき、ATM アダプタ C105 内の AAL5 以下のレイヤの処理は総てハードウェアで実現可能である。

D100,protocol stack and data flow is shown to Figure 11 .

First, it tries user terminal D200 to set ATM link to ATM address of contents server D300 with Signa ring (protocol such as for example Q.2931 ).

At time of this , ATM switch &secondary server D100 Signa ring cell will recognize thespecific VCI , VPI (VCI =5, VPI =0 ), you inform that it has been about that user terminal D200 access will do to large capacity file (hard disk D332 ) of contents server D300, furthermore by the data element content which is carried with Signa ring cell of Q.2931.

Then, contents server D300 imitation it does ATM switch &secondary server D100, does Signa ring between user terminal D200, sets link (25 Mbps ) between hard disk C106 of user terminal D200 and ATM switch &secondary server D100.

Namely with this invention , it is not to set ATM link between user terminal D200 and contents server D300.

When and with this link , you see from user terminal D200, ATM switch &secondary server D100 hypothetically becomes contents server .

In addition, command for random access of large capacity file which is sent from user terminal D200 destined for contents server D300 stack is done in FIFO (Illustration is abbreviated with memory of first-in, first-out type which outputs data which FirstIn FirstOutmemory: first is inputted first. )inside I/Fcontroller (PCI bus controller ) of hard disk C106.

As shown next in Figure 10 , ATM switch &secondary server D100 becomes user terminal D200 and changes,with Signa ring (Q.2931 ), through ATM switch D400, sets ATM link of contents server D300 and155 Mbps (You refer to flow of data of Cplane of Figure 10 ).

After link establishment, through ATM switch D400, user terminal D200 large capacity file inside contents server D300 which access is done in sequential PHY (physical ), asfor data which each layer of ATM , AAL5 terminal is done istransferred by PCI bus C103 in form of CPCS-PDU (cs common section -protocol data unit ) inside reading , ATM adapter C105,through host -PCI bridge C104, once, compilation is done in host memory C102.

At time of this , treatment of layer of AAL5 or less inside ATM adapter C105 is realizable with all hardware .

である。

さらに、ホストメモリ C102,PCI バス C103,ホスト-PCI ブリッジ C104 の転送速度は ATM リンクの転送速度(155Mbps)に比べて十分高速であるため、155Mbps の ATM リンクのスループットを制限することなく、大容量ファイルをホストメモリ C102 に一括転送することが可能である。

ホストメモリ C102 へのファイル転送が完了した後、シグナリングによってコンテンツサーバ D300 と ATM スイッチ&2 次サーバ D100 との ATM リンクを解放する。

ATM リンクの解放後、ホストメモリ C102 内のデータは、再び、ホスト-PCI ブリッジ C104,PCI バス C103 を介してハードディスク C106 へ転送される。

ここで、前述したようにハードディスク C106 内の I/F コントローラの FIFO には、ユーザ端末 D200 からの大容量ファイルへのランダムアクセスコマンドがスタックされている。

これを順に実行することで、ATM リンクによって大容量ファイルデータが ATM スイッチ&2 次サーバ D100 から ATM アダプタ C107(25Mbps)を介してユーザ端末 D200 へ逐次転送される。

ユーザ端末 D200 からファイルに対するランダムアクセスが終了後、ユーザ端末 D200 との間で ATM リンク解放のためのシグナリングが行われて、大容量ファイルのランダムなデータ転送が終了する。

以上説明したように、本実施形態によれば、ホストメモリやハードディスク等の大容量記憶媒体とデータバスを有する計算機を端末間に組み込んで ATM スイッチ及びサーバとして機能させ、第 1 の端末から第 2 の端末へのアクセスの際に、第 2 の端末から大容量ファイルを大容量記憶媒体に一括転送し、その後、第 1 の端末から計算機の大容量記憶媒体へランダムアクセス可能としたもので、第 1 の端末側からすると計算機が仮想的に第 2 の端末となり、第 2 の端末からは計算機が第 1 の端末に見える。

このように、大容量記憶媒体と ATM スイッチを組み合わせることにより、インタフェース速度の異なる ATM リンクの速度変換が行えるだけでなく、大容量記憶媒体が 2 次サーバとして利用できるため、大容量ファイルのランダムアクセスといった大きな負荷を分散でき、高い転送スループットが実現できるという効果が得られる。

また、装置を PC などの汎用計算機により構成

Furthermore, as for forwarding rate of host memory C102, PCI bus C103, host -PCI bridge C104 because it is a fully high speed incomparison with forwarding rate (155 Mbps ) of ATM link , large capacity file it is possible without restricting throughput of ATM link of 155 Mbps , to lump together to transfer in host memory C102.

After file transfer to host memory C102 completes, ATM link of contents server D300 and ATM switch & secondary server D100 is released with Signa ring .

data after releasing ATM link and inside host memory C102, again, through host -PCI bridge C104, PCI bus C103 , is transferred to hard disk C106.

As here, mentioned earlier, random access command to large capacity file from user terminal D200 the stack is done, to FIFO of I/F controller inside hard disk C106.

By fact that this is executed in order, with ATM link large capacity file data through ATM adapter C107 (25 Mbps ) from ATM switch & secondary server D100, sequential it is transferred to user terminal D200.

random access from user terminal D200 for file Signa ring for ATM link release being done after ending and between user terminal D200, random data transfer of large capacity file ends.

As above explained, according to this embodiment , installing computer which possesses host memory and hard disk or other large capacity storage media and data bus between terminal , functioning as ATM switch and server , from first terminal case of the access to second terminal , from second terminal to lump together transfer large capacity file in large capacity storage media , after that, from first terminal being something which is made random accessible to large capacity storage media of computer , From point of view of first terminal side computer becomes second terminal hypothetically, from second terminal computer is visible in first terminal .

this way, not only being able to do rate conversion of different ATM link of the interface speed , due to especially combining large capacity storage media and ATM switch , because large capacity storage media it can utilize, as secondary server be able to disperse large load such as random access of large capacity file , effect that is acquired it can be actualized high transfer throughput .

In addition, as device is designated as configurable with PC or

可能とするとともに高速な汎用データバスを採用したので、上記効果に加え、上述した本発明の機能を汎用計算機上で安価に実現できるという効果が得られる。

なお、上記実施形態においては、ユーザ端末 D200 を 1 台のみ ATM スイッチ & 2 次サーバ D100 に接続する例を示しているが、ユーザ端末 D200 の台数は、2 台以上の複数であってもよい。

図 12 は、本発明を、ハードディスクに蓄積された大容量ファイルに同時にアクセスする場合に適用した実施形態を説明するための図である。

図 12 は本発明の一実施形態におけるファイル転送装置の構成図で、この転送システムは、例えばパーソナルコンピュータから成る送信計算機 1 と、送信計算機 1 のハードディスク 2 と、例えばパーソナルコンピュータから成る受信計算機群 31~3n と、受信計算機群 31~3n のハードディスク群 41~4n と、送信計算機 1 と受信計算機群 31~3n とを接続するネットワーク 5 とを備えている。

次に上記ファイル転送システムの転送動作について説明する。

受信計算機群 31~3n から送信計算機 1 に対して、ファイルへのランダムアクセスの要求があったとする。

これは、例えば上記系を VOD システムに適用した場合、送信側計算機はビデオサーバに、受信側計算機群はクライアントに、ファイルは映画などのビデオソフトに、ランダムアクセスはこのビデオソフトへの一時停止や巻き戻しや早送りに相当する。

送信計算機 1 は複数の受信計算機群 31~3n から同時に要求される複数のランダムアクセスにリアルタイムに応答する代わりに、それぞれの受信計算機に要求されたデータを含むファイルをファイルごと一括して転送し、受信計算機のハードディスク群 41~4n へコピーする。

この際、例えば受信計算機 31 に転送が行われている間は、他の受信計算機群 32~3n からのアクセス要求は一旦送信計算機 1 でスタックしておき、受信計算機 31 への転送が終了するまではこれらの要求には応答しない。

また、一つの受信計算機に対する応答は、一つのファイルの中のあちこちの部分へのランダム

other general purpose computer ,because high speed general purpose data bus was adopted, effect that is acquired on the general purpose computer it can actualize function of this invention which description above is done in inexpensive in addition to above-mentioned effect.

Furthermore, user terminal D200 only 1 example which connects to ATM switch & secondary server D100 has been shown regarding above-mentioned embodiment , , but number of devices of user terminal D200 is good even with plural of two or more .

Figure 12 , when this invention , simultaneously to large capacity file which compilation is done access it does in hard disk , is figure in order to explain embodiment which is applied.

As for Figure 12 with configuration diagram of file transfer device in the one embodiment of this invention , as for this transfer system , hard disk group 41 - 4 n of hard disk 2 of transmission computer 1 and the transmission computer 1 which consist of for example [paasonarukonpyuuta ] and reception computer group 31 - 3 n and reception computer group 31 - 3 n which consist of for example [paasonarukonpyuuta ] and , It has transmission computer 1 and network 5 which connects thereception computer group 31 - 3 n.

Next you explain concerning transfer operation of the above-mentioned file transfer system .

From reception computer group 31 - 3 n vis-a-vis transmission computer 1 , we assume that there was demand for random access to file .

As for this , when for example above-mentioned system is applied to the VOD system , as for transmitting side computer in video server , as for called side computer group in the client , as for file in motion picture or other video software , as for random access it is suitable to halt and rewind and rapid feed to this video software .

Transmission computer 1 transfers , instead of responding to real time in random access of plural which is required simultaneously from thereception computer group 31 - 3 n of plural , every file lumping together file which includes data which is required to the respective reception computer , copy does to hard disk group 41 - 4 n of reception computer .

Until at time of this , while transfer is done in for example reception computer 31 , access request from other reception computer group 32 - 3 n stack does once with transmission computer 1 , transfer to reception computer 31 ends , you do not respond to these requests.

In addition , response for reception computer of one is not to respond to random access to portion of here and there in file

なアクセスに直接応答するのではなく、ファイルの先頭から末尾までのシーケンシャルな読み出しのみを行う。

ファイルへのランダムアクセスは、各受信計算機のハードディスク上にコピーされたファイルに、各受信計算機がローカルで行う。

一般に複数クライアントを收容するサーバシステムにおいては、クライアントからのランダムアクセスに直接応答しようとする、ソフトの割り込み時間とサーバプロセッサの負荷が著しく増加し、サーバに計算機を高性能化しても收容するクライアントの数を増やすことは困難である。

また、ハードディスクへのアクセスは、読み書きヘッドが移動してディスクのシリンダに位置付けするためのシーク動作速度が機械的に最も遅い。

このシーク時間がネックとなって、ランダムアクセスはシーケンシャルな読み書きに比べて著しくスループットが低下する。

これらは結局、データの読み出しという実際の処理が細切れに行われ、処理と処理の間の移行時間が全体の処理に必要な時間を大部分を占めることに起因する。

上記実施形態で述べたように本方法では、送信計算機は出来る限り割り込み時間やソフト処理を減らし、またハードディスクはシーケンシャルな読み出しに専念することにより、処理と処理との間の移行時間を最小限にすることにより、トータルのスループットを向上させている。

次に、本発明のデータ転送方法によるデータの転送手順の他の実施形態について説明する。

図 13 は、送信側及び受信側計算機がそれぞれ接続されたネットワークにおけるデータ転送手順を示すタイムチャートである。

送信側及び受信側計算機は、それぞれ送信すべきデータを蓄積するための送信側メモリと、受信データを蓄積するための受信側メモリと受信時刻を計時するためのタイマを備えている。

図 1 に示す実施形態と比較した場合、図 13 に示す送信側計算機は、図 1 に示す送信側で用いるホスト CPU・F1 とその周辺装置に対応し、図 13 に示す受信側計算機は、図 1 に示す受信側で用いるホスト CPU・F1 とその周辺装置に対応している。

なお、図 13 は、一例として、SEQ=40 のパケット

of one directly, sequential to tail only reading is done from head of file .

As for random access to file , on hard disk of each reception computer in file which copy is done, each reception computer does with the local .

When directly it tries to respond to random access from client regarding server system which accommodates plural client generally,, it is difficult to increase quantity of client where interrupt time of the software and load of server processor increase considerably, high efficiency do computer in server and accommodate.

In addition, as for access to hard disk , read-write head moving, the seek operation speed in order position to attach in cylinder of disk is slowest in mechanical .

this seek time becoming neck , as for random access throughput decreases considerably sequential in comparison with read-write .

These are done after all true case, reading of data treatment chopping \*, transfer time during treatment and treatment required time originates in occupying major portion in treatment of entirety .

As expressed with above-mentioned embodiment , with this method , as for transmission computer as much as possible interrupt time and software treatment is decreased, in addition hard disk throughput of total has improved by designating transfer time during treatment and treatment as minimum sequential by devoting to reading .

Next, you explain with data transfer method of this invention concerning other embodiment of transfer protocol of data .

As for Figure 13 , it is a time chart which shows data transfer protocol in network where transmitting side and called side computer are respectively connected.

transmitting side and called side computer , have timer in order timer to do the called side memory and reception time in order compilation to do transmitting side memory and received information in order compilation to do data which it should transmit respectively.

When it compares with embodiment which is shown in Figure 1 , transmitting side computer which is shown in Figure 13 corresponds to host CPU \*F1 and peripheral which are used with transmitting side which is shown in Figure 1 , called side computer which is shown in Figure 13 corresponds to host CPU \*F1 and peripheral which are used with called side which is shown in Figure 1 .

Furthermore, packet of SEQ=40 omission does Figure 13 , as



が欠落し、SEQ=60 のデータにエラーが発生する場合を示している。

送信側計算機は送信側メモリからデータを読み出し、その時の通信プロトコル、例えば、UDP/IP (User Datagram Protocol:ユーザーデータグラムプロトコル/Internet Protocol:インターネットプロトコル)や ATM/AAL-5 に応じて、読み出したデータをパケットに分割してネットワークに送信する。

送信するパケットには、そのパケットの送信側メモリにおける先頭アドレスとパケット長を記録しておく。

送信側は受信側からの応答確認(ACK)を待つことなく、通信路の帯域に応じた転送速度でデータ(SEQ=10,20,...,110)を送る。

パケットを受信した受信側計算機は、パケットの欠落やエラーがなければ送信側に ACK を返すことなく受信データをメモリに順次蓄積してゆく。

受信側計算機は、パケットの欠落が発生した場合には(SEQ=40 のパケット)、所定の時間が経過した後、そのパケットの送信側先頭アドレスとパケット長を付与して送信側に再送要求する(NAK=40)。

また、受信したパケットにエラーがあった場合には(SEQ=60 のパケット)、そのパケットを直ちに廃棄するとともに、そのパケットの送信側先頭アドレスとパケット長を付与して送信側に再送要求する(NAK=60)。

受信側計算機は、欠落やエラーがあった場合には、欠落あるいは廃棄したパケットが格納されるべき受信側メモリ領域をそのまま空白にしておき、以降受信するエラーのないパケットをこの空白領域以降の領域に蓄積して行く。

なお、ここでは、受信側でパケットの欠落が発生した場合には、所定時間経過後に送信側に再送要求することとしているが、欠落が発生した場合に直ちに送信側に再送要求することも可能である。

また、欠落あるいはエラーによってパケット長を得ることができなかったパケットが格納されるべき受信側メモリ領域の容量は、次に正しく受信したパケット(この場合、SEQ=50 のパケット)に付与されている送信側メモリにおける先頭アドレスの値に従って決定することができる。

再送要求を受けた送信側は、通知(NAK=60 及び NAK=40)された先頭アドレスとパケット長を

the one example , case where error occurs in data of SEQ=60 hasbeen shown.

transmitting side computer reading it is data according to communication protocol , for example UDP /IP of thattime of reading , (UserDatagramprotocol :user data gram protocol /internet protocol :Internet protocol ) and ATM /AAL-5 from transmitting side memory , dividing data to packet , transmits to network .

In packet which it transmits, start address and packet length in transmitting side memory of packet are recorded.

transmitting side data (SEQ=10, 20, \*, 110 ) with forwarding rate which responds to domain of communications line without waiting for response verification (ACK ) from the called side , sends.

called side computer which receives packet , if there is not a omission and a error of packet , sequential compilation does received information in memory withoutreturning ACK to transmitting side .

called side computer , when omission of packet occurs, (packet of SEQ=40 ), predetermined time passage after doing, granting transmitting side start address and packet length of packet , resendsrequires to transmitting side (NAK=40 ) .

In addition, when there is a error in packet which is received (packet of SEQ=60 ),as packet is abolished at once, granting transmitting side start address and packet length of packet , it resends requires to transmitting side (NAK=60 ) .

receiver side computer , when there is a omission and a error , omission ordesignates called side memory area which packet which is abolished should housethat way as white space , later compilation does packet which does nothave error which is received in region after this white space region .

Furthermore, here, when omission of packet occurs with called side ,it resends requires to transmitting side after specified time , but when omission occurs, also it is possible at once to resend to require to transmitting side .

In addition, as for capacity of called side memory area which packet whichcannot acquire packet length with omission or error should house, following to value of start address in transmitting side memory which is granted to packet (In case of this , packet of SEQ=50 ) which is received next correctly it can decide.

transmitting side which receives retransmission request reading \* reconstruction does retransmission packet from

用いて再送パケットを選択的に送信側メモリから読み出して再構成し、受信側へ再送する。

そして、再送パケットを受け取った受信側は、その再送パケットを必要に応じて再構築し、空白のまま残しておいた対応する受信側メモリ領域に蓄積する。

この場合、受信側からの再送要求には、再送すべき送信側先頭アドレスとパケット長が付与されているので、送信側では、容易に再送パケットを再構成することができる。

また、再送要求のあったパケットの先頭アドレスとパケット長の値をスタックしておけば、途中で送信を止めずに全データの送信してしまい、複数のパケットの再送を最後にまとめて行うこともできる。

次に図 14 を参照して、本発明のデータ転送方法を UDP/IP を用いた計算機間データ転送に適用する場合について説明する。

この実施形態では、UDP パケットがデータ転送におけるパケットに相当するものである。

従来、IP はコネクションレス通信のネットワーク層プロトコルとして広く普及しているが、データ通信等高い信頼性が要求される場合、上位のトランスポート層プロトコルには前述の TCP を用いるのが普通である。

しかし、通信路に著しい輻輳がなく受信側バッファ量が十分であれば、本発明によるデータ転送方法を用いた上でトランスポート層に UDP を使えば、十分な信頼性と効率的なデータ転送が両立できる。

selectively transmitting side memory making use of start address and packet length it is notified (NAK=60 and NAK=40 ), resends to called side .

compilation it does in called side memory area where and, called side which receives theretransmission packet according to need reconstruction did retransmission packet , while it was a white space left and corresponded.

In case of this , because transmitting side start address and packet length which it shouldresend are granted to retransmission request from called side , with transmitting side , reconstruction is possible retransmission packet easily.

In addition, if start address of packet which has retransmissionrequest and value of packet length stack are done, midway transmissionnot stopping, all data transmitting it finishes, collecting theretransmission of packet of plural lastly, it is possible alsoto do.

Referring to Figure 14 next, when data transfer method of this invention it applies to data transfer between computer which uses UDP /IP , being attached, youexplain.

With this embodiment , it is something which is suitable to packet the UDP packet in data transfer .

Until recently, IP has spread widely as network layer protocol of [konekushonresu ] communication , but data communication isometry it is and when reliability is required, it isnormal in transport layer protocol of upper position to use aforementioned TCP .

But, if not to be considerable congestion called side buffer quantity is the fully in communications line , after if using data transfer method with this invention , UDP is used in transport layer, sufficient reliability and efficient data transfer both achievements is possible.

<p>図 14 はUDPパケットのフォーマットを示す。UDPパケットのサイズは最大</p>
<p>Figure 14 shows format of UDP packet . As for size of UDP packet maximum</p>
<p>が64kBの可変長である。この場合、送信パケットに付与するパケット長はU</p>
<p>Is variable length of 64 kB. In case of this , as for packet length which is granted to thetransmission packet U</p>
<p>DPヘッダの Length フィールドにマッピングされている。送信側先頭アドレスは</p>
<p>mapping it is done in Lengthfield of DPheader . As for transmitting side start address</p>
<p>IPヘッダの未使用のフィールド(例えば最大32ビットのIP Option フィー</p>
<p>&lt;seq&gt;IP Optionfee of for example maximum 32bit unused field of IP header</p>

ルド)にマッピングしておく。UDPでは、送信元及び宛先のIPアドレスをマ

[rudo ]) mapping it does. With UDP , transmission origin and IP address of addressee [ma ]

ッピングした疑似ヘッダを付加した上で、ヘッダと全データのチェックサムを行

After adding false header [ppingu ], header and checksum of all data line

う。

このチェックサムを用いることで通信中のパケットの損傷を検出することができる。

またパケットの欠落は、受信側のタイムアウト値以内に特定の先頭アドレスのパケットの未到達で判断する。

これらを用いて、前述の方法でデータの転送及び再送要求を行う。

次に、本発明のデータ転送方法を ATM/AAL-5を用いた計算機間データ転送に適用する場合について説明する。

この実施形態では CPCS-PDU(Common Part Convergence Sublayer-Protocol Data Unit:CS 共通部プロトコルデータユニット)がデータ転送におけるパケットに相当し、ATM-セルがデータ転送におけるセルに相当する。

AAL-5 はデータ転送の効率化を計る目的で、SAR-PDU(Segmentation And Reassembly-PDU:セル分割・組立サブレイヤ)にヘッダやトレイラを設けず、CPCS-PDU 毎にのみエラーチェックを行う。

図 15 は AAL-5 CPCS-PDU のフォーマットを示したものである。

CPCS-PDU のサイズは最大が 64kB の可変長である。

この場合、送信パケットに付与するパケット長は CPCS-PDU トレイラの LI(Length Indicator)フィールドにマッピングされている。

送信側先頭アドレスは CPCS-PDU トレイラの未使用のフィールド(例えば 8ビットの CPCS-UU(CPCS ユーザー・ユーザ間情報)フィールドや CPI(Common Part Identifier:共通部識別子)フィールド)にマッピングしておく。

CPCS では複数の SAR-PDU をまとめて CRC-32(Cyclic Redundancy Check-32:32ビットの巡回冗長符号)のエラーチェックを行う。

転送単位はこの SAR-PDU に ATM ヘッダを付けた ATM セルであるから、セルの欠落と損傷

\*

injury of packet in communication can be detected by fact that this checksum is used.

In addition within timeout value of called side it judges omission of the packet , with not yet arrival of packet of specific start address .

Making use of these, transfer and retransmission request of data are done with aforementioned method .

When next, data transfer method of this invention it applies to data transfer between the computer which uses ATM /AAL-5, being attached, you explain.

With this embodiment it is suitable to packet CPCS-PDU (common PartConvergenceSublayer-protocol data Unit:cs common section protocol data unit ) in data transfer ,it is suitable to cell ATM -cell in data transfer .

AAL-5 with objective which measures making efficient of data transfer , doesnot provide header and trailer in SAR-PDU (SegmentationAndReassembly-PDU :cell portion data Unit:cs & assembly sub layer ), does error check in every only CPCS\*PDU .

Figure 15 is something which shows format of AAL-5CPCS\*PDU .

size of CPCS-PDU maximum is variable length of 64 kB.

In case of this , packet length which is granted to transmission packet mapping is done in LI (Lengthindicator ) field of CPCS\*PDU trailer .

transmitting side start address mapping does in unused field (CPCS\*UU (data between CPCSuser \*user ) field and CPI (common PartIdentifier:common section identifier ) field of for example 8bit ) of CPCS-PDU trailer .

With CPCS collecting SAR\*PDU of plural , it does error check of CRC \*32 (Round redundant code of CyclicRedundancyCheck-32:32bit ) .

Because transfer unit is ATM-cell which attaches ATM header to this SAR\*PDU , omission and injury of cell being

がこの CRC-32 チェックで同時に検出できる。

これらを用いて、前述の方法でデータの転送及び再送要求を行う。

本実施形態によれば、従来のように応答確認をパケット毎に行うことをせずに、送信パケットの先頭アドレスと長さを送受信間で通信することで、転送の高速化とエラー検出されたパケットの再送の効率化を同時に実現できる。

以上説明したように本実施形態のデータ転送方法は、データを複数パケットに分割して転送する場合に、パケット毎に応答確認を行うことなく一括して送信するため、従来より著しく転送効率を向上させることができる。

また、通信中のエラーが原因で再送が必要になった場合、パケットの先頭アドレスとパケット長を送受信間で通信した上で、受信側から特定パケットを選択的に再送要求する方法をとる。

これにより、再送開始のために送信側タイマの終了を待つ必要がなく、かつ余分なパケットを再送することもなくなる。

さらに再送パケットを送信側で再構成する時のソフト処理とメモリ管理を最小限にすることができるため、TCP 等と比較してはるかに簡単で効率的な再送処理が実現できる。

また、特に、本発明のデータ転送方法はバルクデータの転送に用いるのに好適である。

次に、図 13~図 15 を参照して説明した実施形態を、送受信計算機間にルータとしての機能を提供する中継計算機を追加した場合の構成に適用する場合の実施形態について説明する。

この実施形態では、中継計算機が、図 1 に示す図 1 に示すホスト CPU・FI とその周辺装置の送信側と受信側の双方の機能を提供するものに対応する。

まず、AAL-5/ATM を用いた計算機間データ転送に適用する場合について以下に説明する。

図 16 に本発明における転送シーケンス例を示す。

この実施形態では、CPCS-PDU がデータ転送におけるパケットに相当する。

CPCS-PDU のサイズは最大が 64kByte の可変長である。

this CRC \*32check , it can detect simultaneously.

Making use of these, transfer and retransmission request of data are done with aforementioned method .

According to this embodiment , conventional way doing response verification in every packet do, start address and length of transmission packet by fact that communication it does, acceleration and error of transfer making efficient of retransmission of packet which is detected can be actualized simultaneously between transmission and reception .

As above explained, dividing data into plural packet , when it transfers, lumping together without doing response verification in every packet , in order to transmit, transport efficiency it can improve data transfer method of this embodiment , from until recently considerably.

In addition, when error in communication with cause retransmission becomes necessary, start address and packet length of packet between the transmission and reception communication after doing, method which selectively it resends requires specific packet is taken from called side .

Because of this, it is not necessary to wait for end of transmitting side timer for starting retransmission, it becomes without either at same time resending excess packet .

Furthermore when reconstruction doing retransmission packet with the transmitting side , software treatment and because memory managing can be designated as the minimum , being much simple by comparison with TCP etc, it can actualize efficient retransmission treatment.

In addition, especially, data transfer method of this invention is ideal in order to use for transfer of bulk data .

Next, referring to Figure 13 ~Figure 15 , you explain concerning embodiment when it applies to configuration when it adds computer for relay which offers function embodiment which you explain, as router between transmission and reception computer .

With this embodiment , relay computer , corresponds to transmitting side of host CPU \*FI and peripheral which are shown in Figure 1 which is shown in Figure 1 and those which offer both functions of called side .

First, when it applies to data transfer between computer which uses the AAL\*5/ATM being attached, you explain below.

Transfer sequence example in this invention in Figure 16 is shown.

With this embodiment , it is suitable to packet CPCS\*PDU in data transfer .

size of CPCS-PDU maximum is variable length of 64 kByte.

送信側計算機は、メモリからデータをブロック単位で読み出し、CPCS-P
As for transmitting side computer , from memory data with block unit reading , CPCS*P
DUペイロードにカプセル化する。その際に、該メモリブロックの転送を行う
encapsulation it does in DUpayload . At that occasion, it transfers said memory block , [fu ]
ファイルの先頭からのオフセットアドレスをCPCS-PDUトレイラ部のCPC
offset address from head of [airu ] CPC of CPCS- PDU trailer section
S-UUとCPIの 16bit 部分にマッピングする(図 15 参照)。そして、ATM
mapping it does in 16 bitportion of S- UU and CPI, (Figure 15 reference ).And, ATM
セル化、物理レイヤ終端後、宛先に向けてデータを送信する(SEQ=10、2
<seq>SEQ=10, 2 After cell converting and physical layer terminal , data is transmitteddestined for addressee
0、…100)。
0...100) .
データを中継する計算機では、物理レイヤ、ATMレイヤ、AALレイヤ終端
With computer which relays data , physical layer , ATM layer , AALlayer terminal
を行う。AALレイヤ終端時にCPCS-UUとCPIにマッピングされたアド
It does. At time of AALlayer terminal mapping it was done in CPCS*UU and the CPI, [ado ]
レス情報を取り出し、該アドレスとPDUのサイズであるLIの情報をもとに該
[resu ] data is removed, on basis of data of LI whichis a size of said address and PDU said
計算機に於てデータ転送用に確保したメモリ領域にデータをCPCS-PDU単位でDMA(Direct Memory Access:ダイレクトメモリアクセス)転送し、データを蓄積する。データの中継する計算機の大容量メモリに一時的に蓄積すること
Regarding to computer , in memory area which it guarantees in onefor data transfer DMA (direct memory access :direct memory access ) it transfers data with CPCS- PDU unit , the compilation does data . In large capacity memory of computer which relays data compilation do in the temporary
によってリンク間の転送速度に応じて速度変換を行うことが可能となる。
It becomes possible to do rate conversion , so according to forwarding rate between link .

データを宛先となる計算機に向けて転送する場合、受信したアドレスと長さの情報をもとに該メモリ領域から DMA 転送により、データを読み出す。

これにより、装置内のデータ転送においてこれまで転送処理やメモリ管理を集中的に処理していたホスト CPU の負荷が軽減されると共に、転送処理にソフト処理が関与しないため高速化することができる。

メモリ領域からデータを読み出した後、再び CP CS-PDU ペイロードにデータをカプセル化し、CP CS-UU、CPI にアドレスを格納することによって送信側計算機によって付与されたアドレス情報の保存を行う。

これにより、送信側計算機から受信計算機まで、エンド・エンドで計算機間のメモリ間をアドレスとサイズ情報を活用した高速蓄積転送が可能になる。

次にエラーデータ再送手順の実施形態について説明する。

パケットを中継する計算機と最終的にパケットを受信する計算機では、データを受信し物理レイヤ、ATM レイヤ、AAL レイヤ終端を行う。

AAL レイヤ終端時に CRC-32 計算によりエラーを検出したデータ(SEQ=40,60)については、CPC S-PDU 単位で直ちにデータを廃棄し、該計算機にデータを送信した計算機に対してアドレスと PDU 長を含んだ再送要求用セル(NAK=4 0,60)を送る。

再送要求を受けた計算機は、アドレスと PDU 長を元にメモリ領域から再送すべきデータを読み出し、再び CPCS-PDU にカプセル化して再送する。

このように、アドレスとサイズの情報を付与してデータを転送すると共にデータの中継を行う計算機間で蓄積しながら転送することによって送信側計算機と受信側計算機間だけでなく、中継計算機間においても下位レイヤによるハードウェア処理を主体として効率良い再送が可能となり、ネットワーク全体からみても再送にかかわる負荷が大幅に軽減される。

次に、汎用データバス、大容量メモリ、ネットワークプロトコルを終端するネットワークインタフェースアダプタによって構成される汎用計算機アーキテクチャから成るデータ転送装置の構成例について説明する。

When it transfers data destined for computer which becomes the addressee, data is read out on basis of data of address and length which are received from said memory area with DMA transfer.

Because of this, as load of host CPU which so far treated transfer treatment and memory managing intensively in data transfer inside the device is lightened, because software treatment does not participate in transfer treatment, acceleration it is possible.

From memory area data reading it is after, again in CPCS-PDU payload the encapsulation to do data, it retains address data which by fact that the address is housed in CPCS-UU, CPI is granted with transmitting side computer.

Because of this, from transmitting side computer to reception computer, with endo \*endo between memory between computer high speed compilation transfer which utilizes address and size data becomes possible.

Next you explain concerning embodiment of error data retransmission protocol.

With computer which receives computer and finally packet which relay the packet, data is received and physical layer, ATM layer, AAL layer terminal is done.

At time of AAL layer terminal data is abolished at once with CPCS-PDU unit concerning data (SEQ=40, 60) which detects error with CRC -32 calculation, cell (NAK=40, 60) for retransmission request which includes address and PDU length vis-a-vis computer which transmitted the data to said computer is sent.

reading, encapsulation doing data which it should resend from the memory area on basis of address and PDU length again in CPCS-PDU it resends computer which receives retransmission request.

this way, granting data of address and size, as it transfers data, while compilation doing between computer which relays data efficient retransmission becomes possible with the lower position layer with hardware treatment as main component by fact that it transfers not only between transmitting side computer and called side computer, in between relay computer, load which relates to retransmission considered as network entirety is lightened greatly.

Next, you explain concerning configuration example of data transfer device which consists of general purpose computer architecture which configuration is done with [netto waaku intafees uadaputa] which general purpose data bus, large capacity memory, network protocol the terminal is done.

図 17 に装置構成例を示す。

この図に示す各構成は、パケットを受信し、物理レイヤ、ATM レイヤ、AAL レイヤ処理を行うネットワークインタフェースアダプタ(あるいはネットワークアダプタカード)C508、C509、C510、及び C511、高速汎用データバスである PCI バス C506 及び C507、PCI バスとホストバスをインタフェースするホスト-PCI バスブリッジ回路 C504 及び C505、ホストバス C503、ホスト CPU(中央演算装置)C502、DRAM で構成されるホストメモリ C501 である。

ネットワークインタフェースアダプタ C508 によってデータを受信し、物理レイヤ、ATM レイヤ終端後、セルヘッダの VPI/VCI 値により、転送先の計算機と接続されているネットワークインタフェースアダプタを識別する。

そして、同-VPI/VCI 値のセルをバッファリングして AAL レイヤまでカード内で終端する。

AAL レイヤ終端時に CRC-32 計算によりデータエラーのチェックを行うと共に CPCS-PDU トレイル部にマッピングされているアドレス情報とサイズを取り出し、ローカル CPU がホストメモリ C501 に対して CPCS-PDU 単位で DMA 転送をセットする。

ホストメモリにおいて転送に使用可能な領域はあらかじめホスト CPU からローカル CPU に対してそのベースアドレスとサイズを通知しておく。

CPCS-PDU トレイル部より取り出したアドレス情報、LI に格納されている PDU 長、ホスト CPU により通知を受けたデータ蓄積用ホストメモリ領域のベースアドレス、サイズの情報をもとにローカル CPU はホストメモリに格納するアドレスの再計算を行い、新たなアドレスを求めることも可能である。

これによりホストメモリをリングバッファとして使用することも可能になり、転送するファイルサイズ分のメモリ領域をホストメモリ中に確保できない場合においても効率を損なうことなくデータ転送を行うことができる。

そしてネットワークアダプタカード C508 のローカル CPU は、VPI/VCI 値から決定された転送先のネットワークインタフェースアダプタ C511 のローカル CPU に対して前記アドレスとサイズ情報を通知する。

通知を受けたネットワークインタフェースアダプタ C511 のローカル CPU は、DMA 転送をセットしホストメモリ C501 からデータを読み出す。

equipment configuration example is shown in Figure 17 .

It is a host memory C501 which configuration is done with host -PCI bus bridge circuit C504 and C505, host bus C503, host CPU (central processing unit ) C502, DRAM which PCI bus C506 and C507, PCI bus and host bus which are a network interface face adapter (Or network adapter card ) C508, C509, C510, and a C511, high speed general purpose data bus where each configuration which is shown in this figure receives packet , treats physical layer , ATM layer , AAL layer interface are done.

data is received with [nettowaakuintafeesuadaputa ] C508, [nettowaakuintafeesuadaputa ] which computer of the forwarding destination is connected is identified after physical layer , ATM layer terminal , due to VPI /VCI value of cell header .

And, [buffuaringu ], to AAL layer terminal it does cell of same -VPI /VCI value inside card .

As check of data error is done at time of AAL layer terminal with CRC -32 calculation, it removes address data and size which mapping are done to CPCS-PDU trailer section, local CPU it transfers DMA with CPCS-PDU unit vis-a-vis host memory C501 set .

In host memory useable region notifies base address and size to transfer beforehand from host CPU vis-a-vis local CPU .

Also it is possible local CPU re-to calculate address which is housed in host memory on basis of data of base address , size of the host memory area for data storage , which receives notification PDU length which is housed in address data , LI which is removed from CPCS-PDU trailer section, with host CPU to seek new address .

Because of this as ring buffer , it becomes possible, to use host memory when memory area of file size amount which is transferred cannot be guaranteed in host memory putting, efficiency without impairing also it is possible to do data transfer .

And local CPU of network adapter card C508 notifies aforementioned address and the size data vis-a-vis local CPU of [nettowaakuintafeesuadaputa ] C511 of forwarding destination which is decided from VPI /VCI value.

local CPU of [nettowaakuintafeesuadaputa ] C511 which receives notification transfers DMA and set reads out data from host memory C501.

データを読み出した後、AAL レイヤ処理時に C PCS-PDU トレイラ部(CPCS-UU, CPI)に前記通知を受けたアドレスをマッピングする。

これによりアドレス情報の保存を行う。

その後、ATMレイヤ処理部でセル化され宛先となる計算機に向けて送信される。

この際、PCI バス C506 と C507 は独立に動作することが可能であり、ホストバスのデータ転送速度は PCI バス(32bit/33MHz=132MB/s)より高速(64bit/66MHz=528MB/s)であるため PCI バス側からはホストメモリに対してデータ書込と読み出しが同時に行えるように見える。

例えば、ネットワークインタフェースアダプタカード C508 で受信したパケットは PCI バス C507 のトラヒック状態に関係なくホストメモリに書き込むことが可能である。

そしてホストメモリに蓄積したデータは PCI バス C507 のトラヒック状態に応じて読み出すことにより、輻輳時にはデータを蓄積し、トラヒック量が少ないときは直ちに読み出すといった高速かつ柔軟なデータ転送が可能となる。

バストラヒックのコントロールはホスト PCI バスブリッジのバスアービトラージ機能によってハードウェアにより高速に処理される。

data reading is after, at time of AAL layer treatment address which receives aforementioned notification to CPCS-PDU trailer section (CPCS-UU, CPI ) the mapping is done.

Because of this it retains address data .

After that, to cell it is converted with ATM layer processing unit and it is transmitted destined for computer which becomes addressee .

At time of this , as for PCI bus C506 and C507 being possible to operate in independence, as for data transfer speed of host bus because PCI bus (32 bit/33MHz =132MB /s ) from it is a high speed (64 bit/66MHz =528MB /s ), in order to be able to do data writing and reading simultaneously, vis-a-vis host memory from PCI bus side it is visible.

As for packet which is received with for example [nettowaakuinfatesuadaputakaado ] C508 it is possible to write to host memory , regardless of traffic state of PCI bus C507.

high speed and flexible data transfer that become possible and in host memory the data compilation it does data which compilation is done to when congesting by reading out according to traffic state of PCI bus C507, when amount of traffic is little, it reads out at once.

Control of bus traffic is treated to high speed with bus arbitration function of host PCI bus bridge by hardware .

<p>以上説明したように、本実施形態によれば、パケットに付与されているアドレ</p>
<p>As above explained, according to this embodiment , it is granted to the packet , [adore ]</p>
<p>ス情報とサイズ情報を活用することにより、ソフトウェアによるメモリ管理が簡素化され計算機間の通信において蓄積転送処理を高速化できる。さらに、エラーデータの再送を送信側計算機と受信側計算機間だけでなく中継を行う計算機間で</p>
<p>memory managing simplification is done with software by [su ] data and utilizing size data , acceleration is possible compilation transfer treatment in communication between computer . Furthermore, between computer which relays retransmission of the error data not only between transmitting side computer and called side computer</p>
<p>選択的に行うことができ、TCPと比較して効率的な再送処理が実現できる。特に、本発明のデータ転送方法はバルクデータの転送に好適である。次に、図 13～図 17 を参照して説明した実施形態においてパケットに付与するアドレスの他の形態について説明する。図 18 は、本発明によるアドレス付与の実施形態である 64 ビットのアドレス</p>
<p>selectively it is possible, to do, it can actualize efficient retransmission treatment by comparison with TCP . Especially, data transfer method of this invention is ideal in transfer of bulk data . Next, referring to Figure 13 ~Figure 17 , you explain concerning other form of address which it grants to packet in embodiment which you explain. As for Figure 18 , address of 64 bit which are a embodiment of address grant with this invention</p>
<p>体系の一例を示す図である。先頭 32 ビットは IP アドレスで、TCP/IP 通信</p>



It is a figure which shows one example of system . As for head 32bit with IP address , TCP/IP communication
で通信相手を特走するための論理アドレスである。2番目の 24 ビットは、計算機内に蓄積されたデータの格納場所を示すメモリアドレスである。ただし、これは
So it is a logic address in order special running to do communication counterpart . 24 bit of 2 nd are memory address which shows storage site of data which compilation is done inside computer . However, as for this
メモリの0番地を起点にした絶対アドレスではなく、転送するファイルの先頭位置を起点とするオフセットアドレスである。3番目の3ビットは計算機のSCS
It is not an absolute address which designates 0 of memory as the source , it is a offset address which designates head position of file which istransferred as source . As for 3 bit of 3 rd SCS of computer
I (Small Computer System Inerface)バスに接続されるSCSIデバイスを認識するSCSI-IDである。最後の5ビットで通信プロトコルを指定する。通信に際しては、ユーザは従来通りIPアドレスを使うことが出来る。通信に使われるプロトコルが従来のIP通信であれば、メモリアドレス以下を読まずにIPパケットにカプセル化すればIPルータを介した通信がそのまま可能である
It is a SCSI - ID which recognizes SCSI device which is connected to the I (Smallcomputer system Inerface ) bus . communication protocol is appointed with last 5 bit . In case of communication , user until recently sort can use IP address . If protocol which is used in communication is conventional IP communication , without readingbelow memory address , encapsulation does in IP packet , is through IP router the communication that way being possible, it is
。

転送メモリブロックにメモリアドレスを付与して通信するプロトコルでは、2番目のメモリアドレス(オフセットアドレス)を読み込むようにする。

また3番目の SCSI-IDを使用することにより、計算機内の SCSI デバイスを直接指定することができる。

上記のような統合アドレス体系をとることにより、通信プロトコルや計算機内外を問わないデータ通信機構を確立することができる。

また、ネットワーク層のプロトコルとして IP を用いる場合、図 19 に示すようにパケットにアドレスを付与することができる。

図に示すように、通常、IP 通信に使われる IP の情報は IP ヘッダの中に収容される。

IP 処理されるデータには IP ヘッダが付けられ、上位層から渡されたデータを IP ヘッダによってカプセル化し、下位層に渡す。

Granting memory address to transfer memory block , with protocol which the communication is done, it reads memory address (offset address ) of 2 nd , it requires.

In addition SCSI device inside computer can be appointed directly byusing SCSI -ID of 3 rd .

data communication mechanism which does not question communication protocol and computer inside and outside asdescription above by taking integrated address system , can be established.

In addition, when IP is used as protocol of network layer , asshown in Figure 19 , address can be granted to packet .

As shown in figure, data of IP which usually, is usedin IP communication is accommodated in IP header .

encapsulation it does data where IP IP header is attached by data which is treated, is transferred from upper position layer with the IP header , transfers to lower position layer.

逆に下位層から来たデータはIPヘッダの内容を解析し、必要に応じて上位層に渡す。

IPヘッダの中のオプションフィールドは可変長の長さを持ち、セキュリティレベルやソースルートなど特別の目的を除いて通常は使用されない。

このオプションフィールドに転送メモリブロックのメモリアドレスをマッピングすることにより、通常のIPルータを介した本発明によるデータ転送が可能となる。

なお、上記各実施形態を参照して説明した本発明によるファイル転送方法を実行するためのプログラムは、フロッピーディスク、光ディスク、磁気光ディスク等の記憶媒体に格納して配布させることが可能である。

また、インターネットなどのネットワークを使用して配布することも可能である。

以上図面を参照して本発明の実施形態例を説明したが、本発明は、その精神又は主要な特徴から逸脱することなく、他のいろいろな形で実施することができる。

そのため、前述の実施形態例はあらゆる点で単なる例示にすぎず、限定的に解釈してはならない。

本発明の範囲は、特許請求の範囲に示すものであって、明細書本文には何ら拘束されない。

さらに、特許請求の範囲の均等論に属する変形や変更は、すべて本発明の範囲内となる。

## Drawings

【図1】

data which comes from lower position layer conversely analyzes content of IP header , transfers to according to need upper position layer.

Usually option field in IP header is not used with length of the variable length , excluding special objective such as security level and source root .

Is through conventional IP router data transfer becomes possible with this invention by the mapping doing memory address of transfer memory block in this option field .

Furthermore, referring to above-mentioned each embodiment , in order to execute file transfer method with this invention which you explain as for program , housing in floppy disk , optical disk , magnetic optical disk or other storage media , distribution fabric it is possible to do.

In addition, using Internet or other network , distribution fabric also it is possible to do.

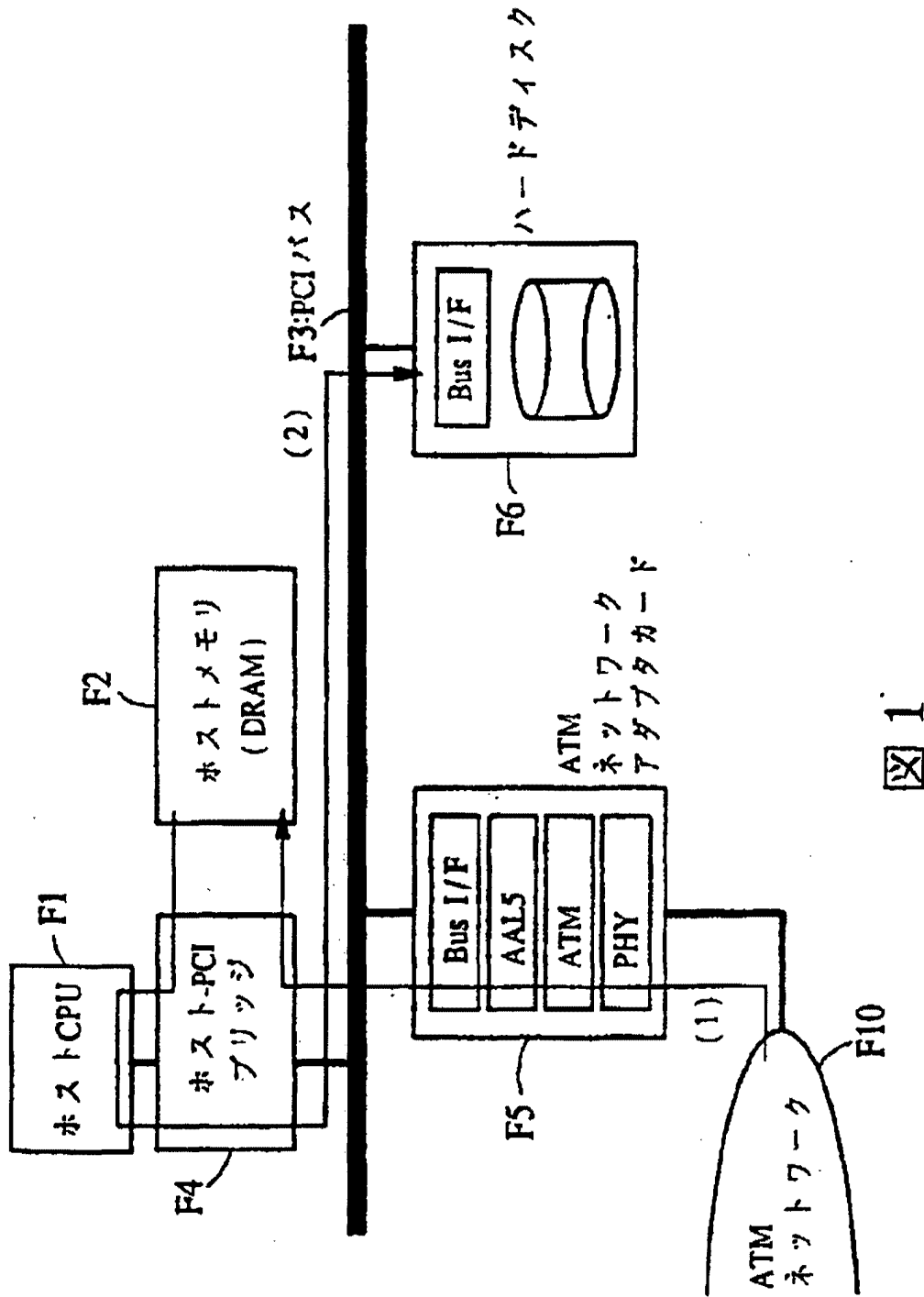
From here, referring to upper drawing aspect, you explained embodiment of this invention , but it can execute this invention, in other various shape without deviating from emotion or principal feature.

Because of that, in all respects interpretation you do not have to do the aforementioned embodiment in limited to be no more than a mere illustration.

Range of this invention being something which is shown in Claims , the what restraint is not done to specification main text .

Furthermore, deformation and modification which belong to the equivalent theory of Claims are inside range of all this invention .

[Figure 1 ]



【図2】

[Figure 2]

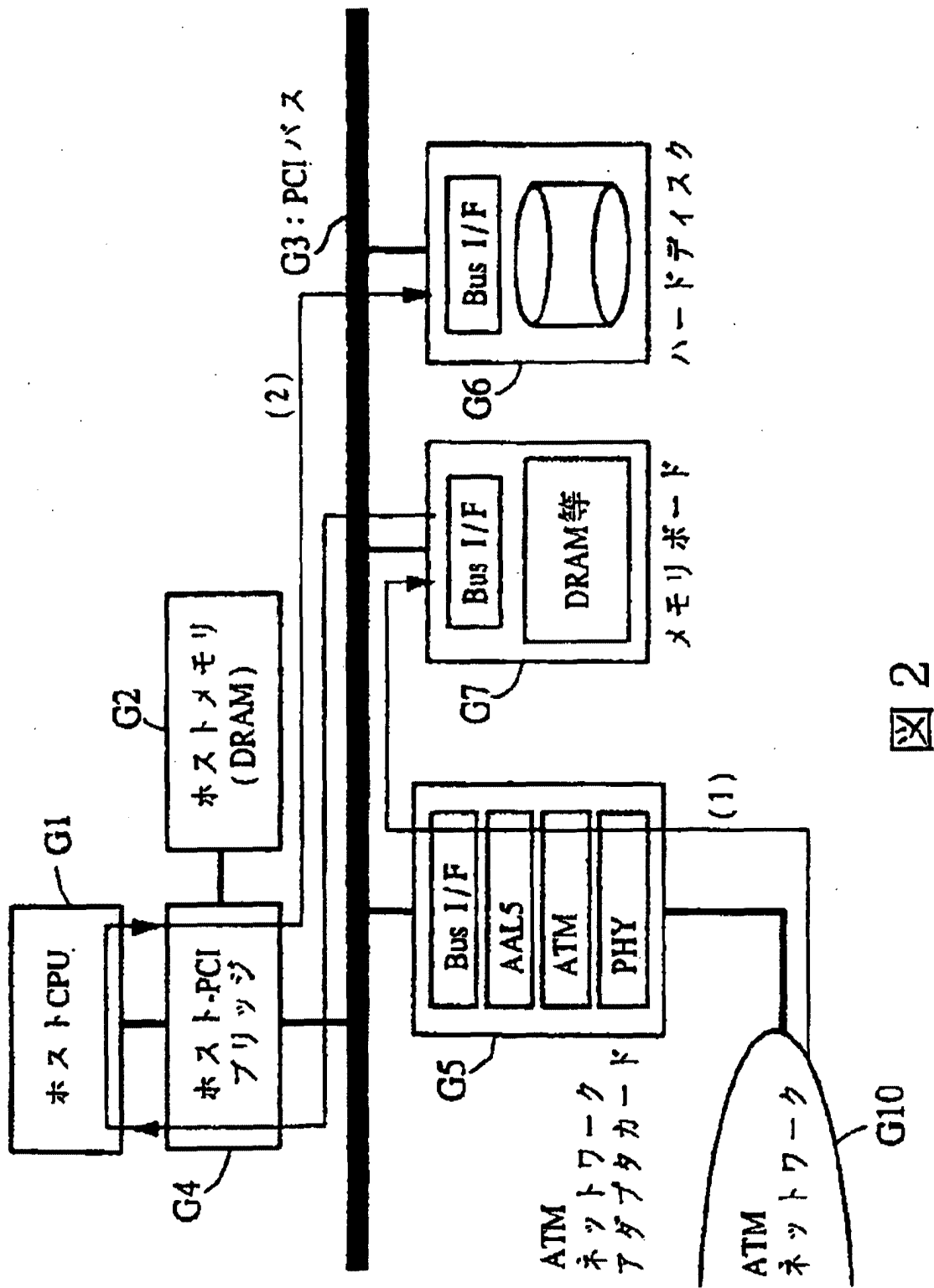
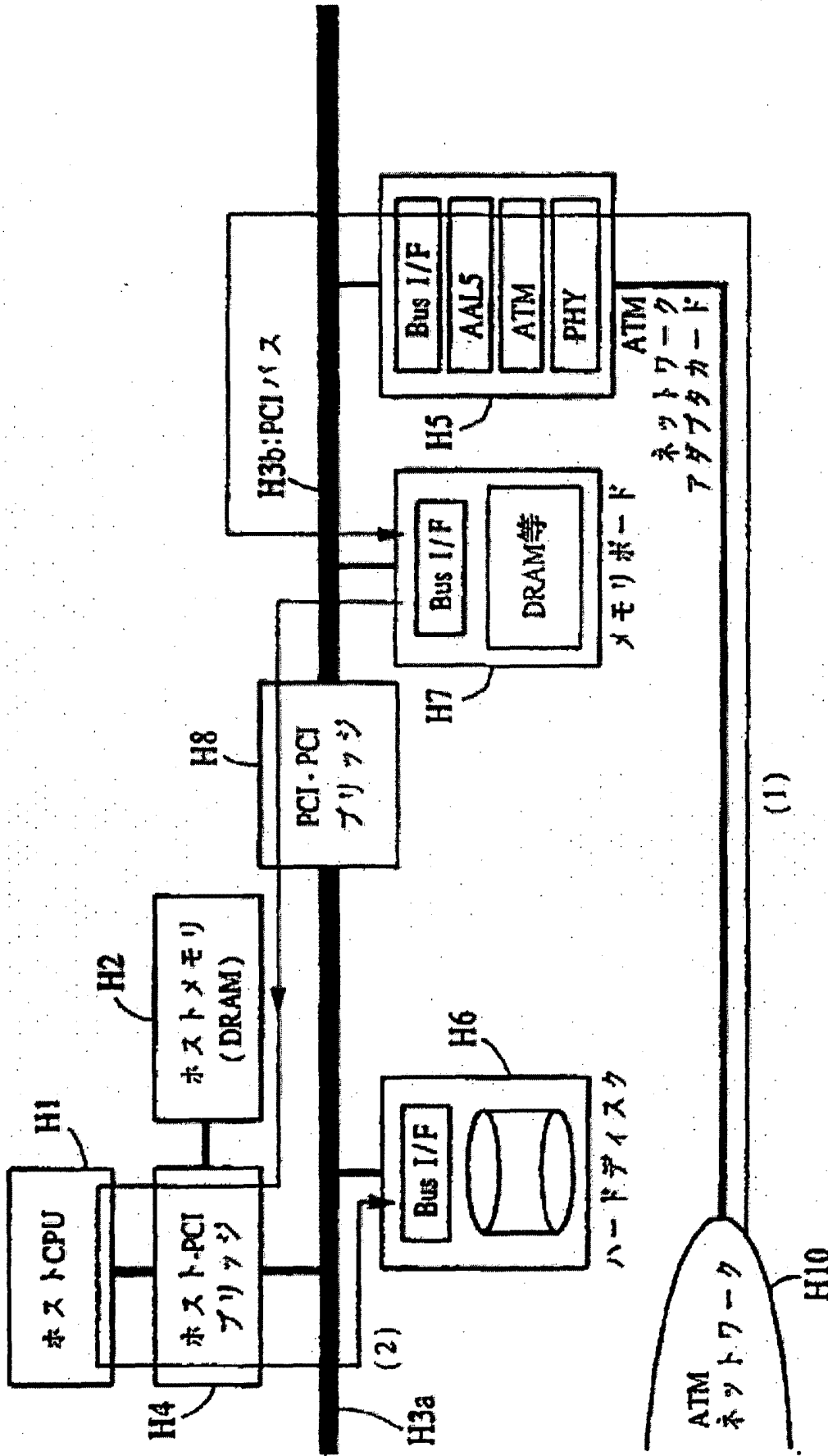


図 2

【図3】

[Figure 3]



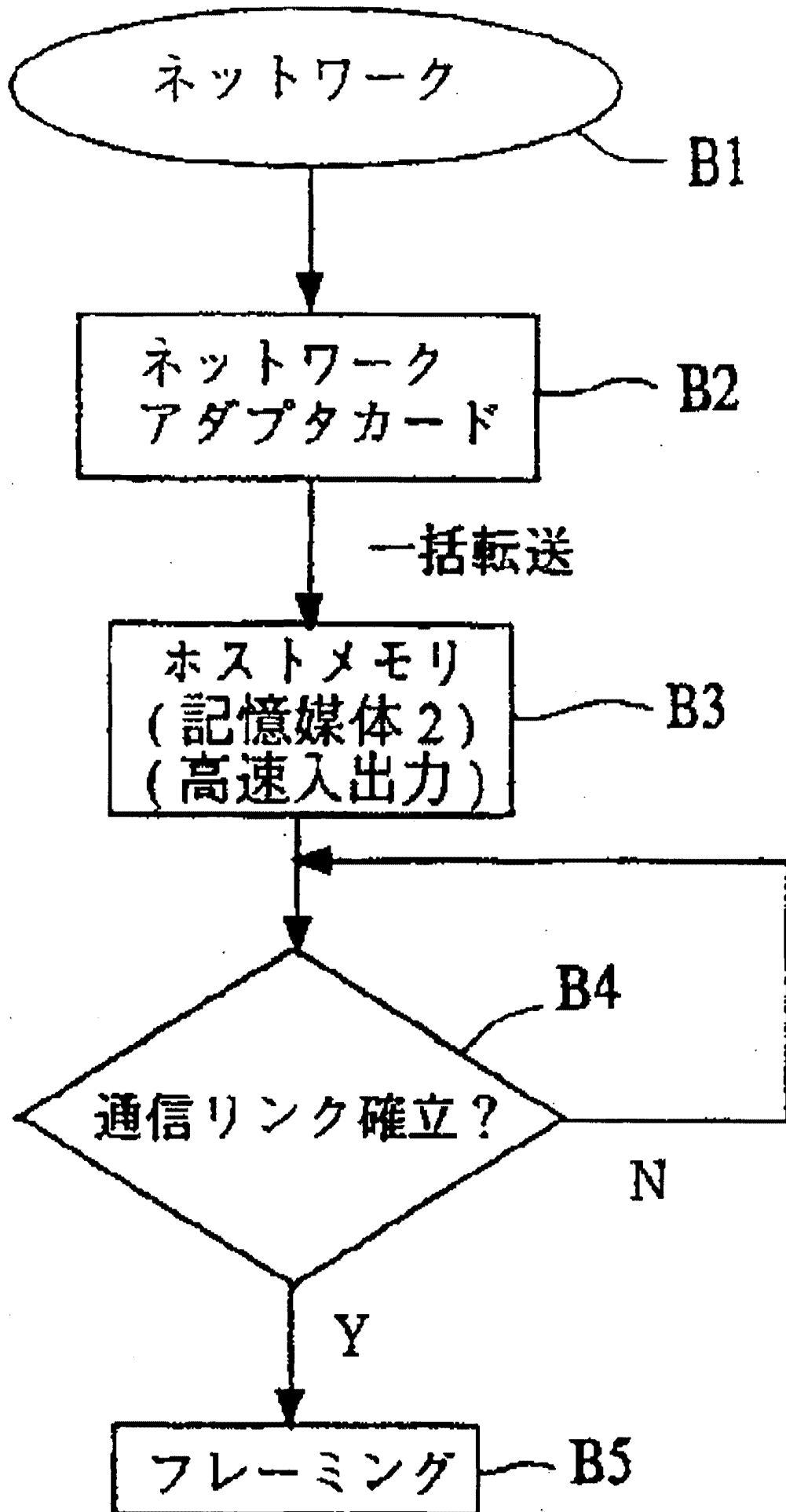
(1)

図3

(7,296)

【図4】

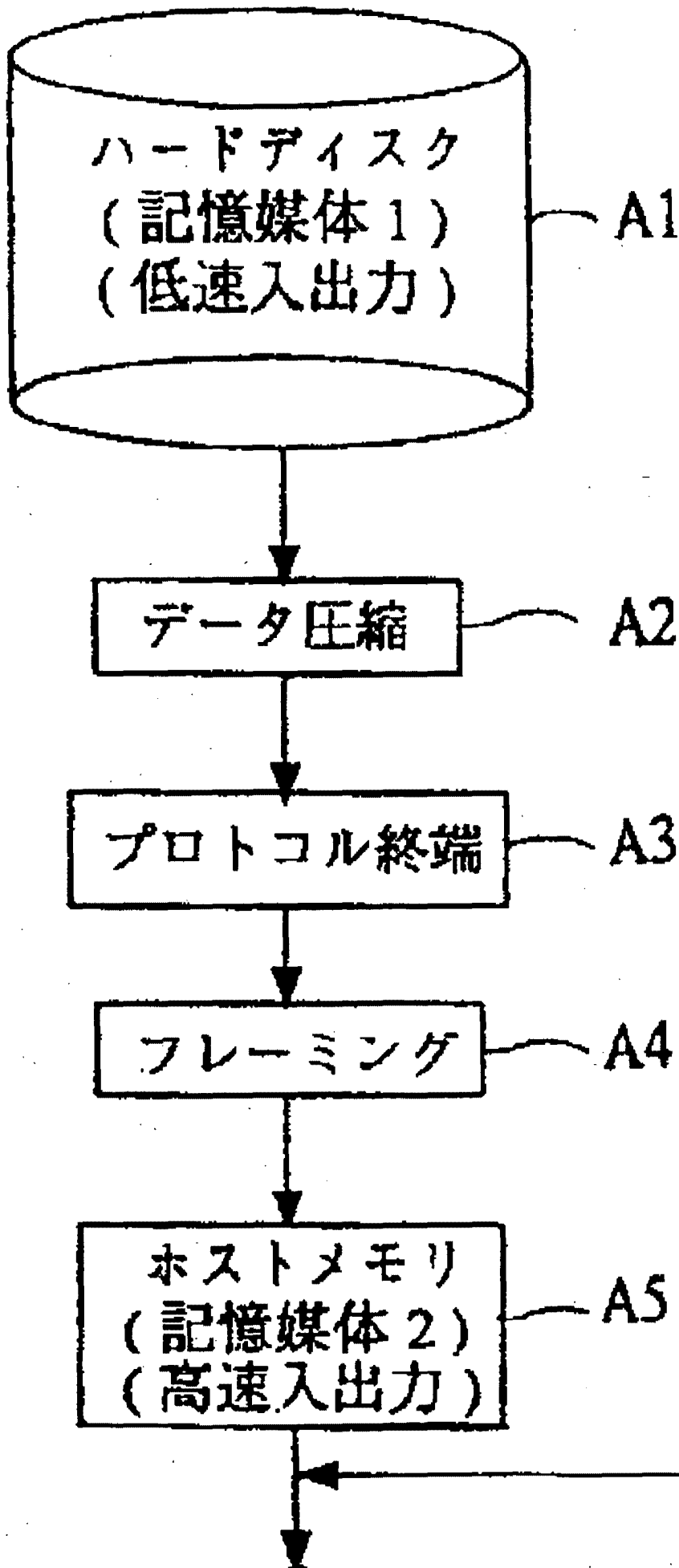
[Figure 4 ]





【図5】

[Figure 5]



(7,296)

【図6】

[Figure 6]

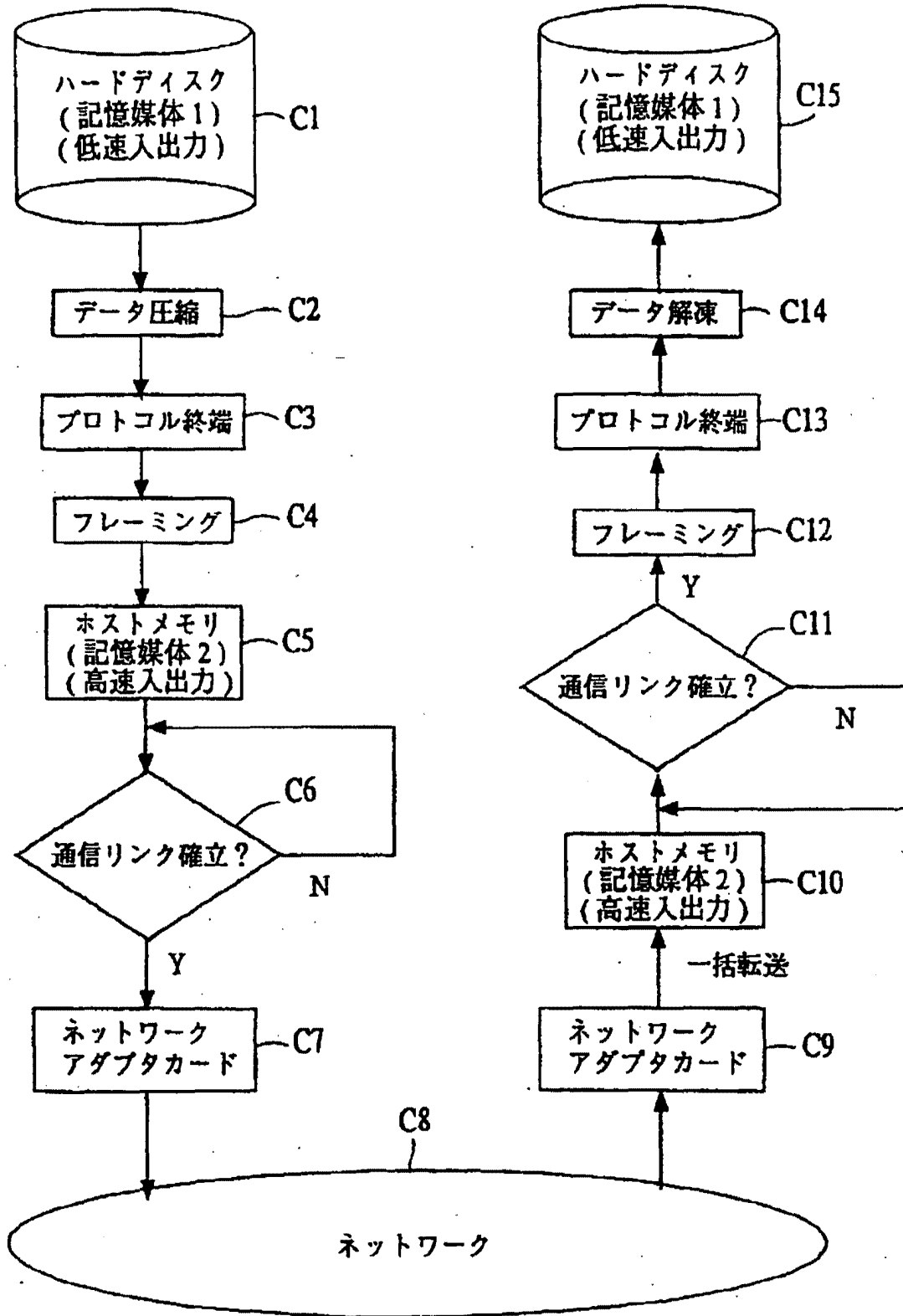


図 6

(7,296)

【図7】

[Figure 7]

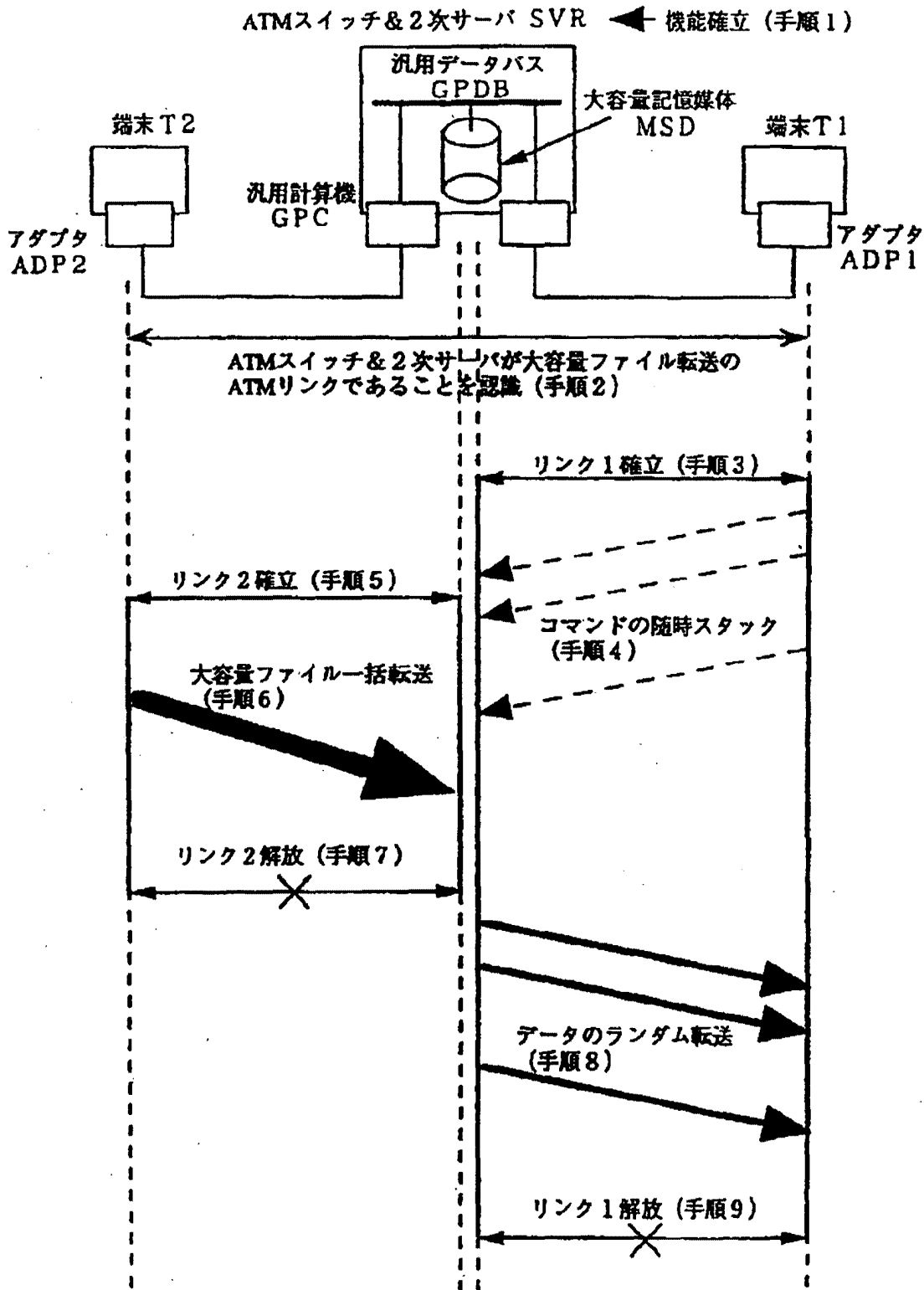


図 7

【図8】

[Figure 8]

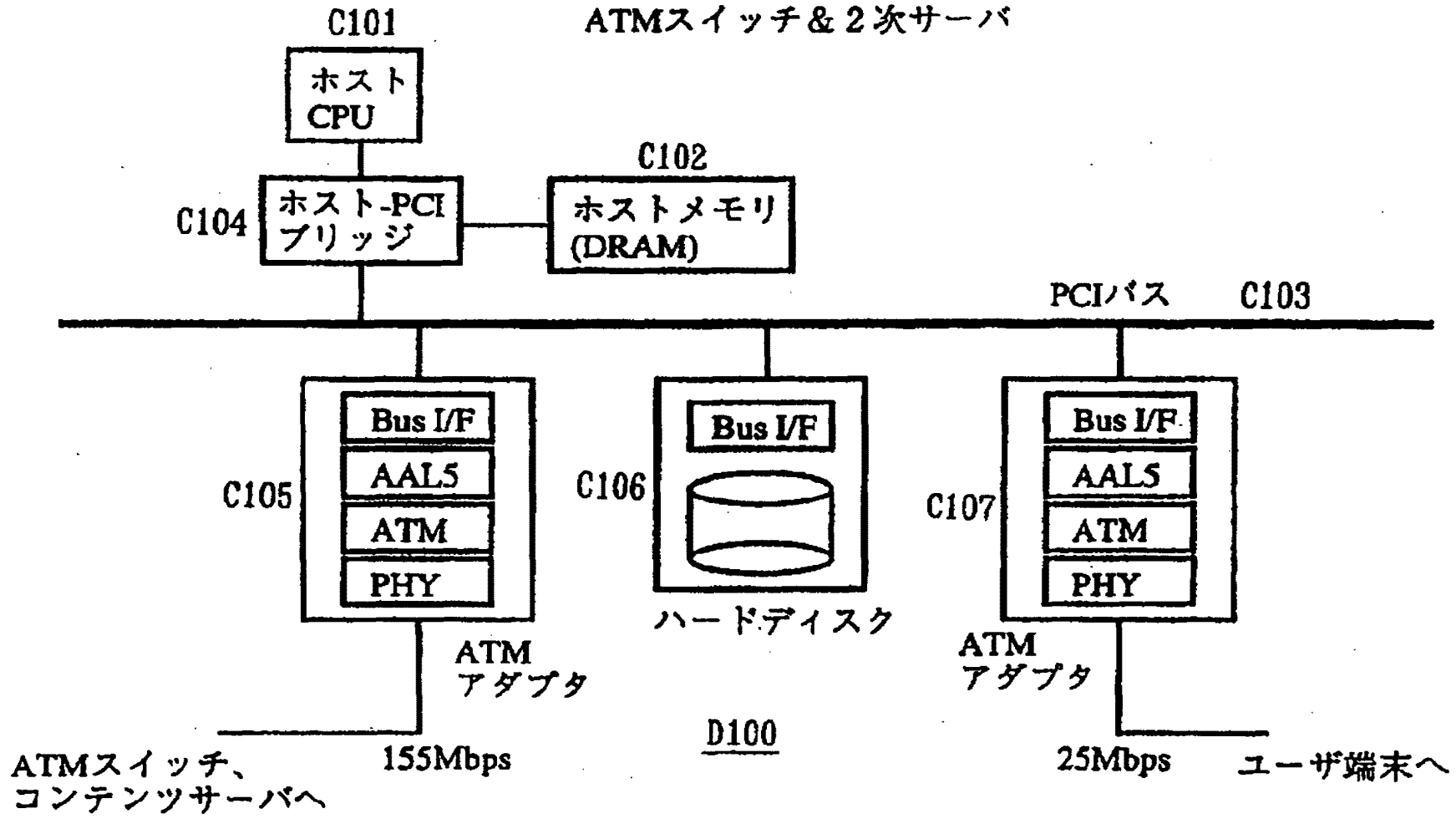


図 8



【図9】

[Figure 9]

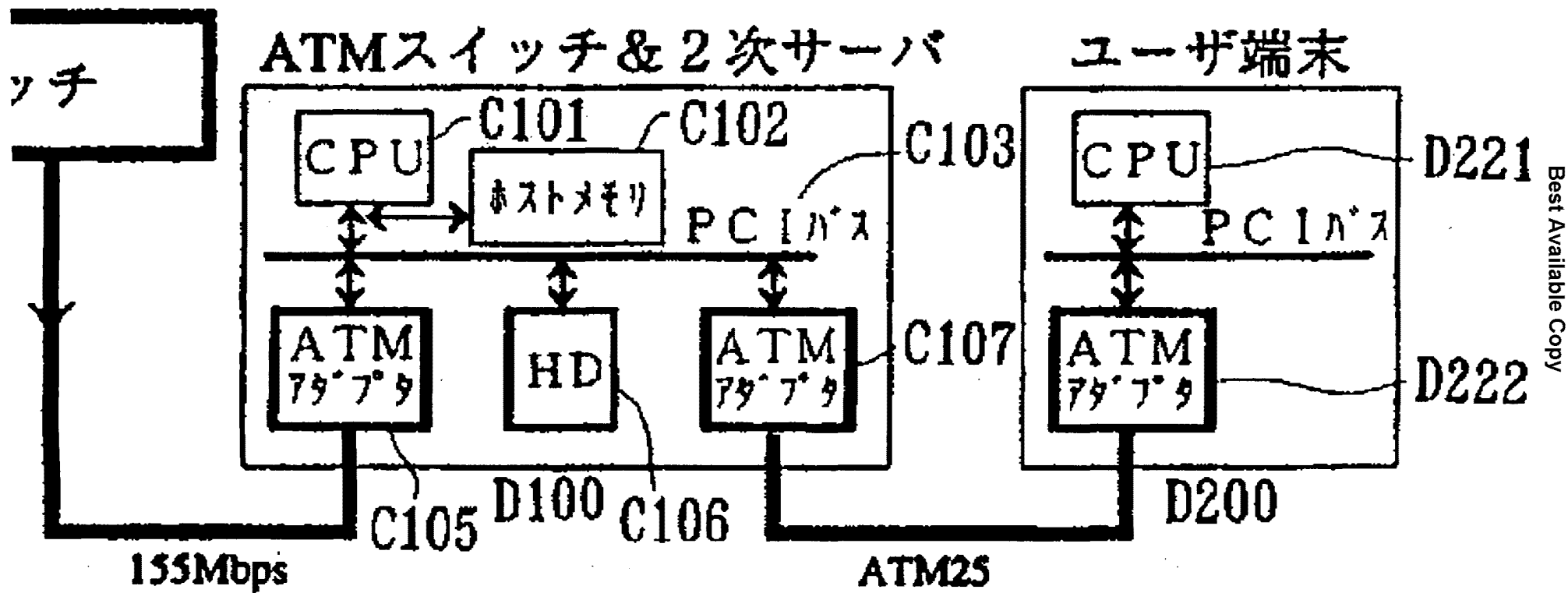


図 9

7296)

18-8-4

Best Available Copy

【図10】

[Figure 10]

コンテンツサーバ

ATMスイッチ

ATMスイッチ&2次サーバ

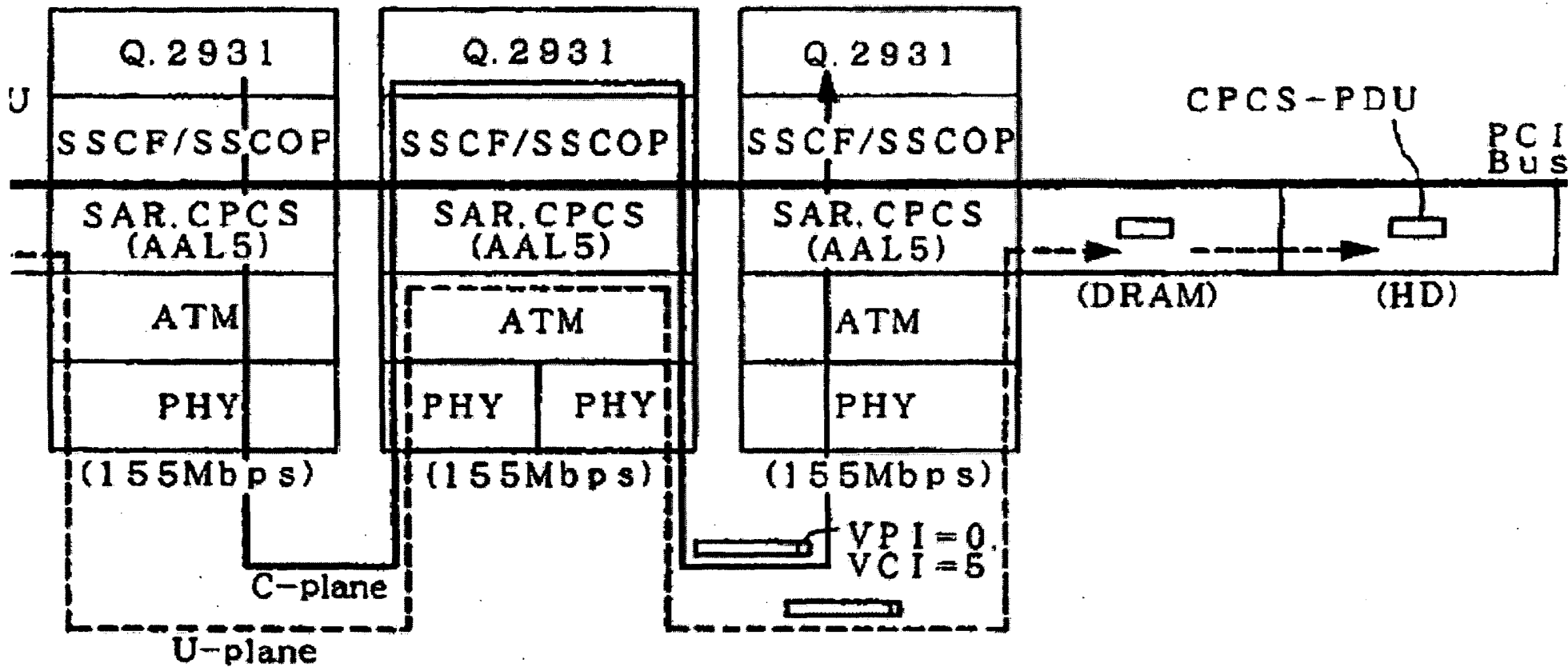


図 10

7,296)

18-8-4

【図11】

[Figure 11 ]

ATMスイッチ&2次サーバ

ユーザ端末

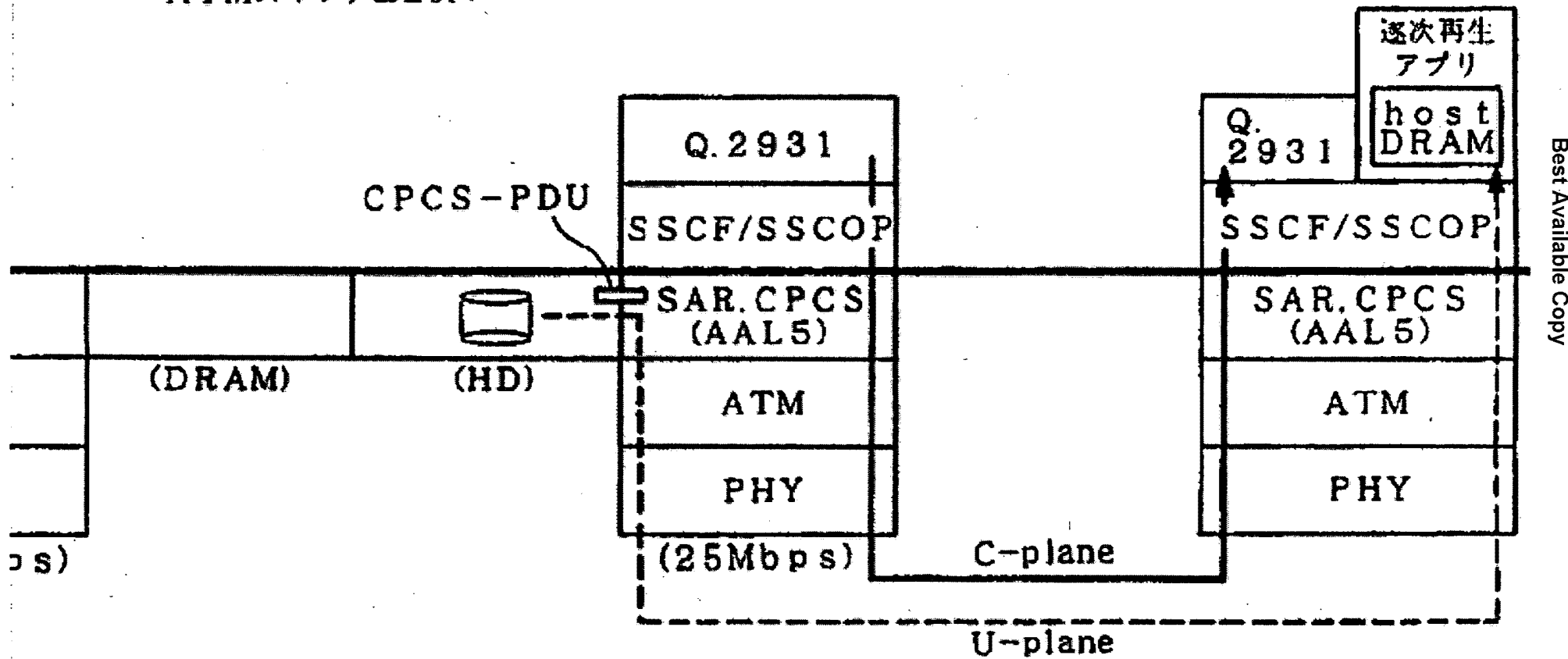


図 11

7.296)

18-8-4

【図12】

[Figure 12]

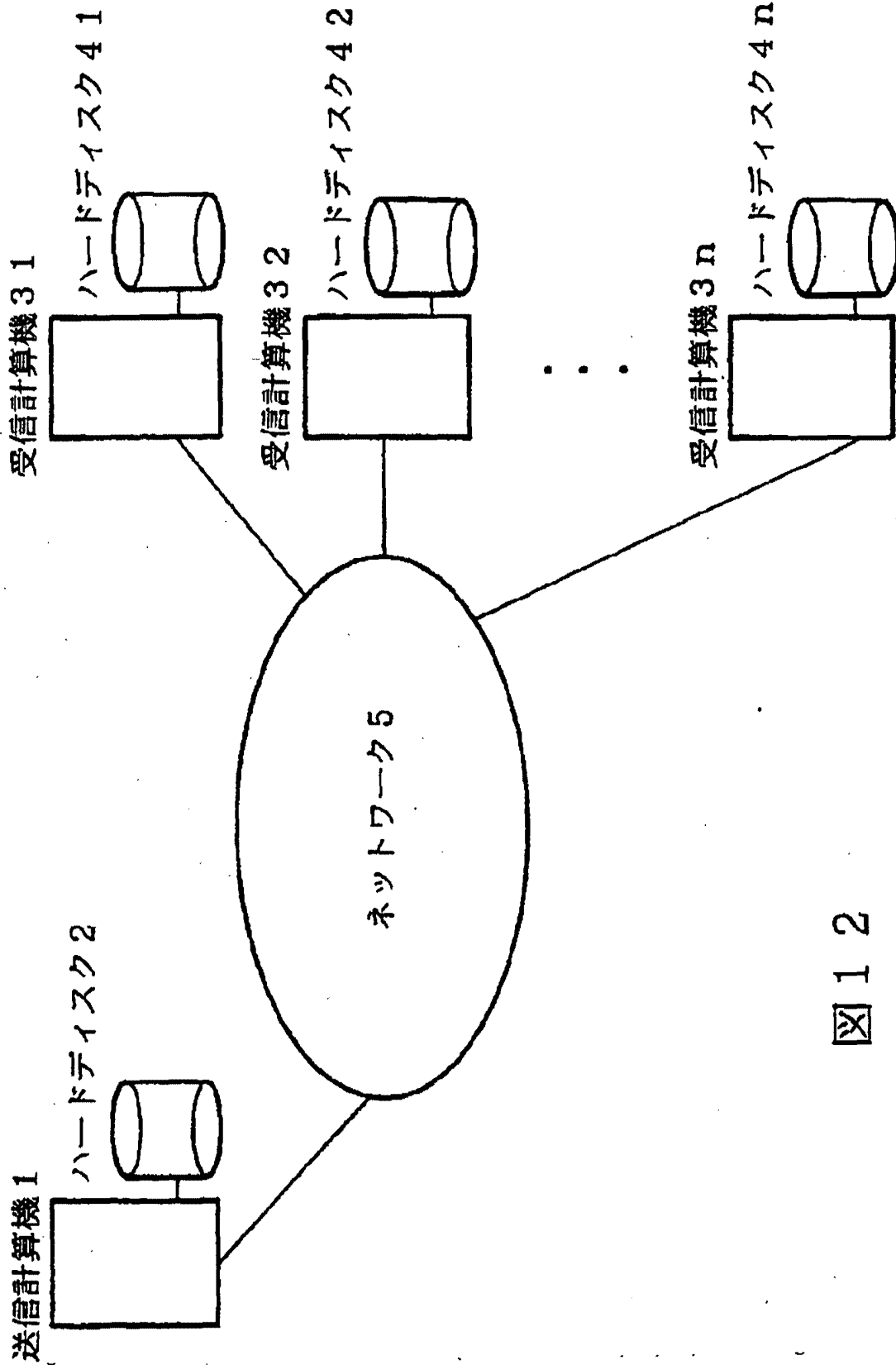


図12

(7,296)



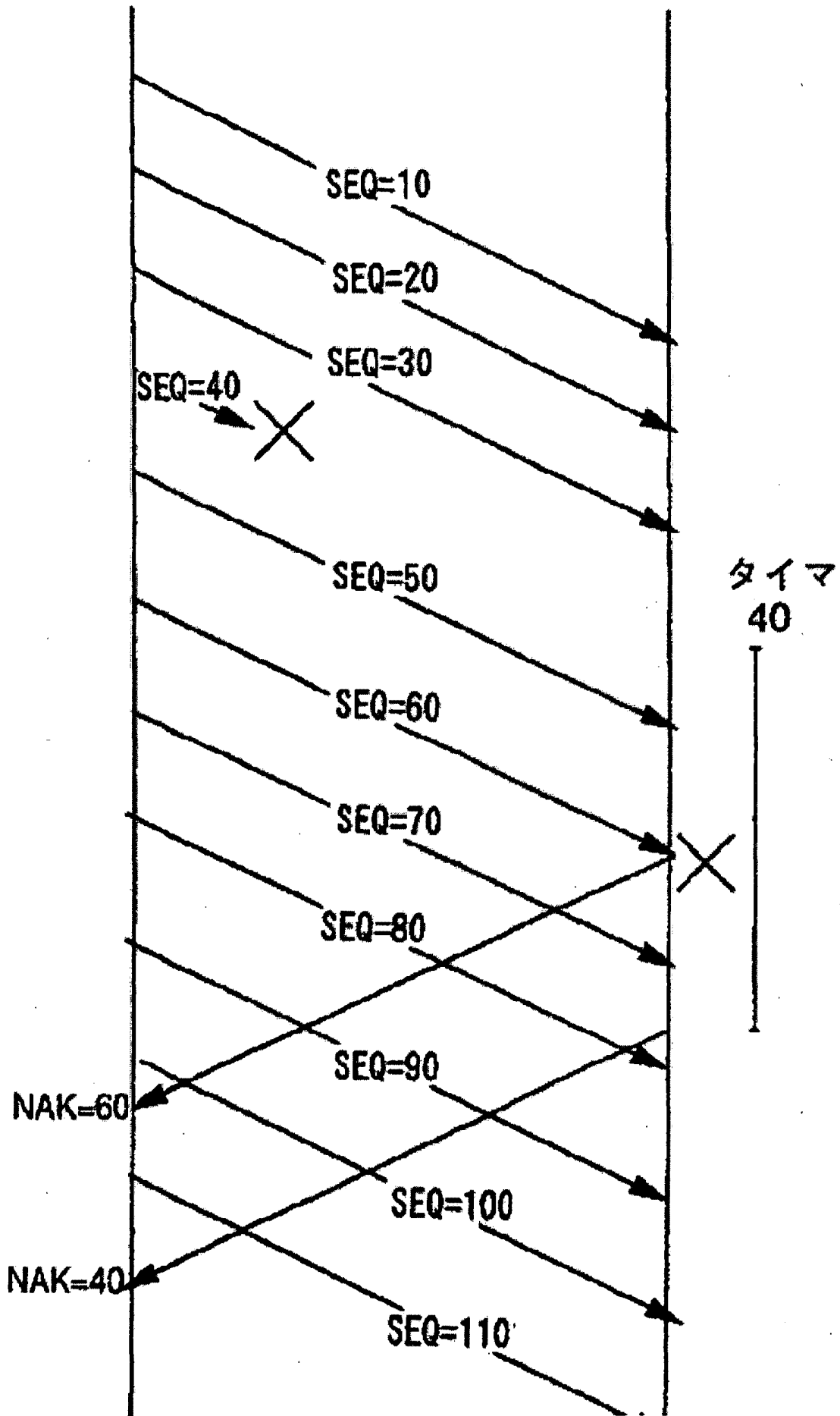
【図13】

[Figure 13]

送信側

受信側

18-8-4



7,296)

【14】

[Figure 14 ]

ビット数 0

16

31

UDP Source Port	UDP Destination Port
UDP Message Length	UDP Checksum
Data	

図 14

【図15】

[Figure 15]

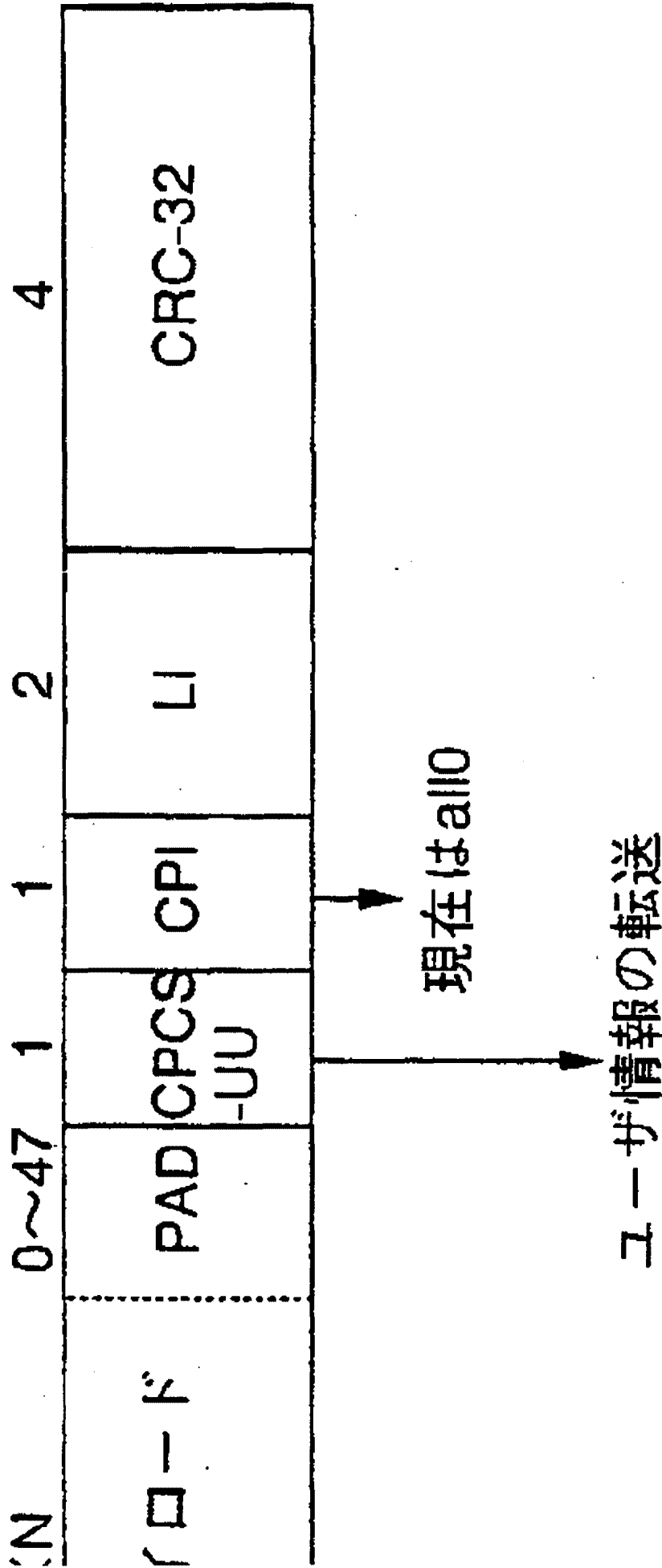
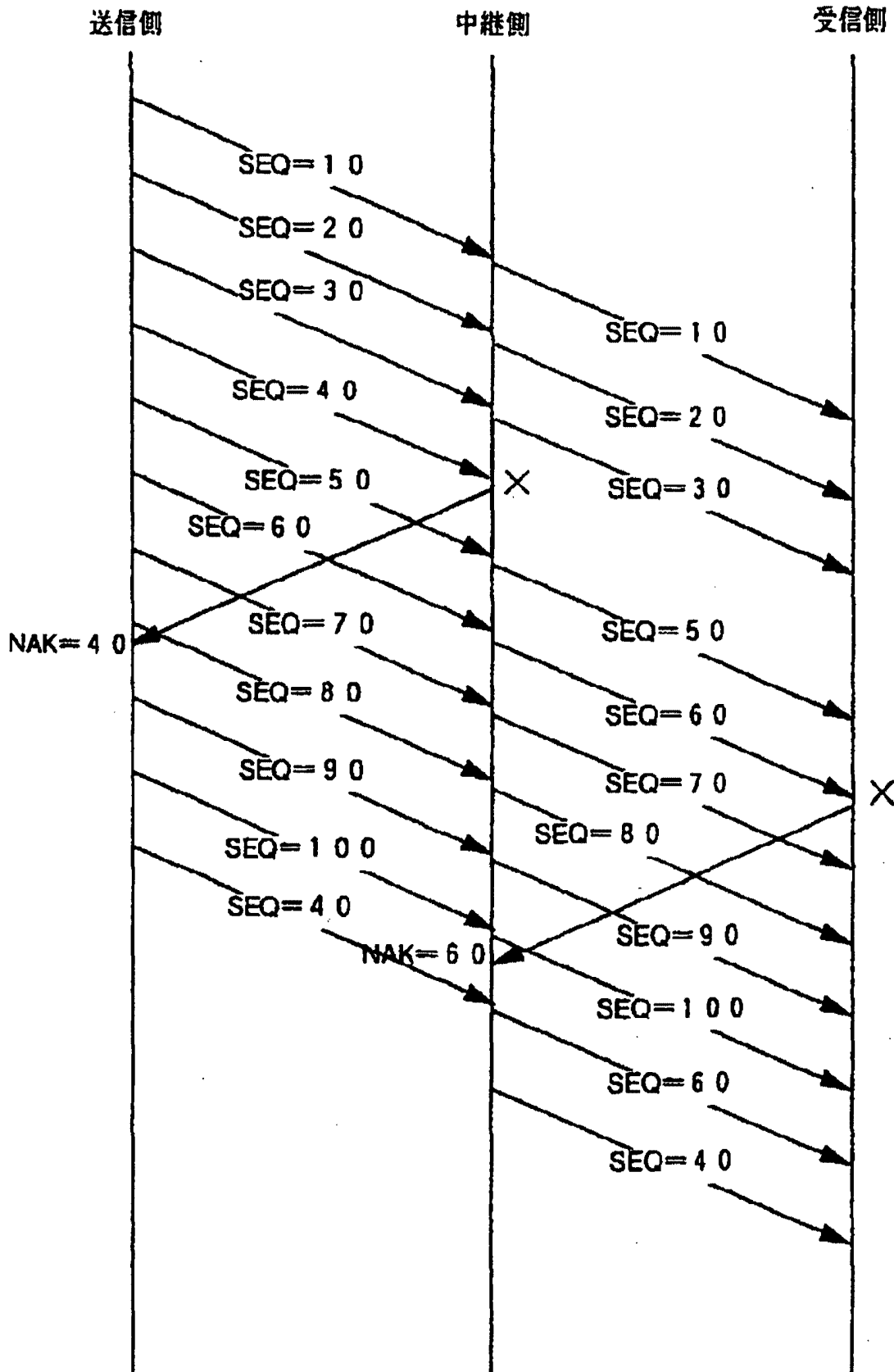


図 15

【図16】


[Figure 16]



(7,296)

図 16



【17】

[Figure 17]

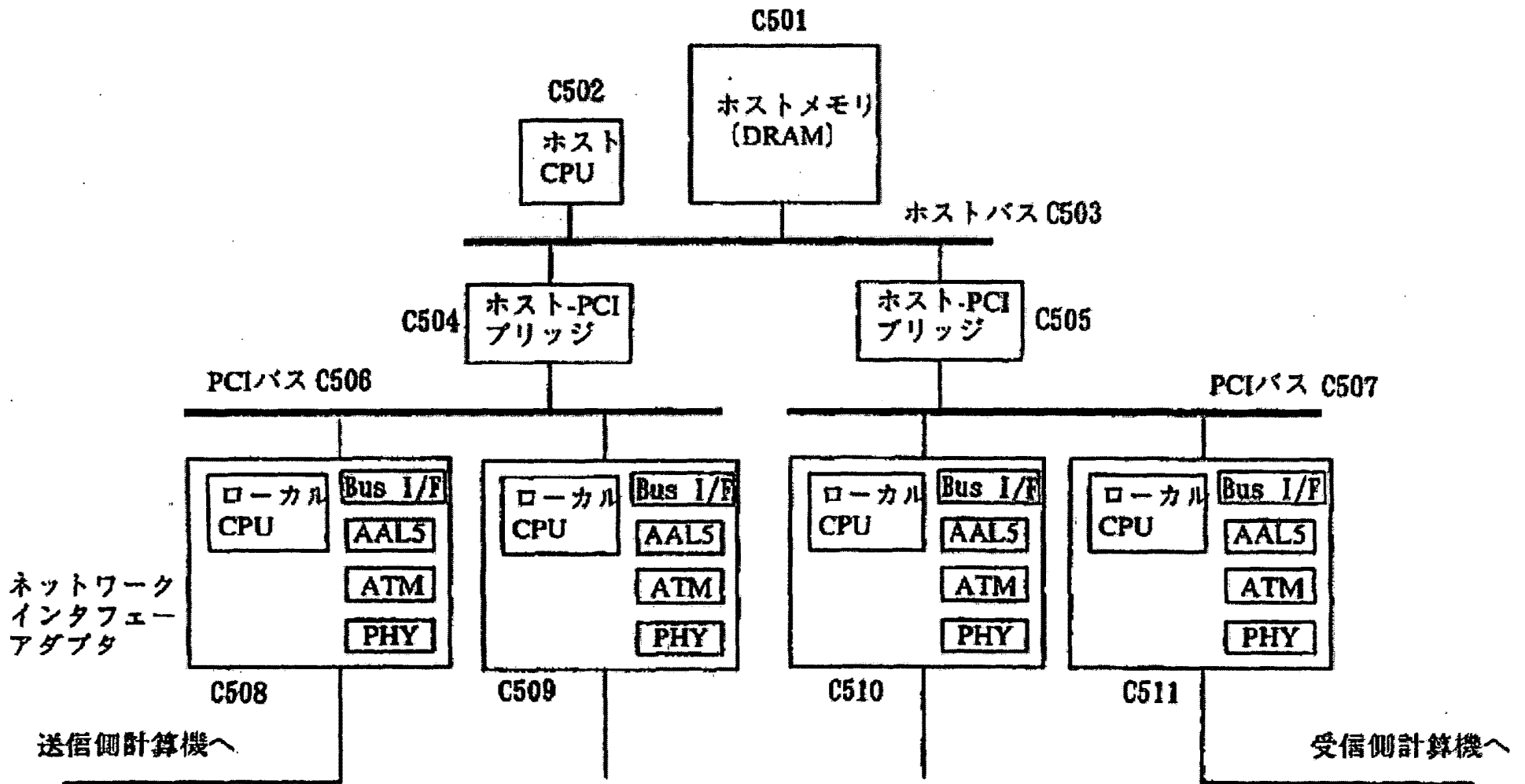


図 17

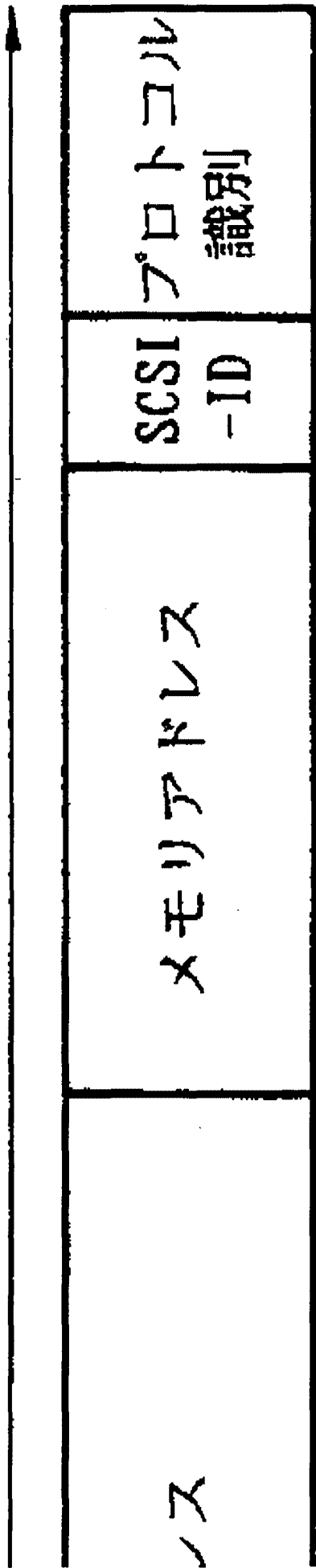
7,296)

18-8-4

【18】

[Figure 18 ]

64ビット



ト  
24ビット  
3ビット 5ビット

図 18

【図19】

[Figure 19]

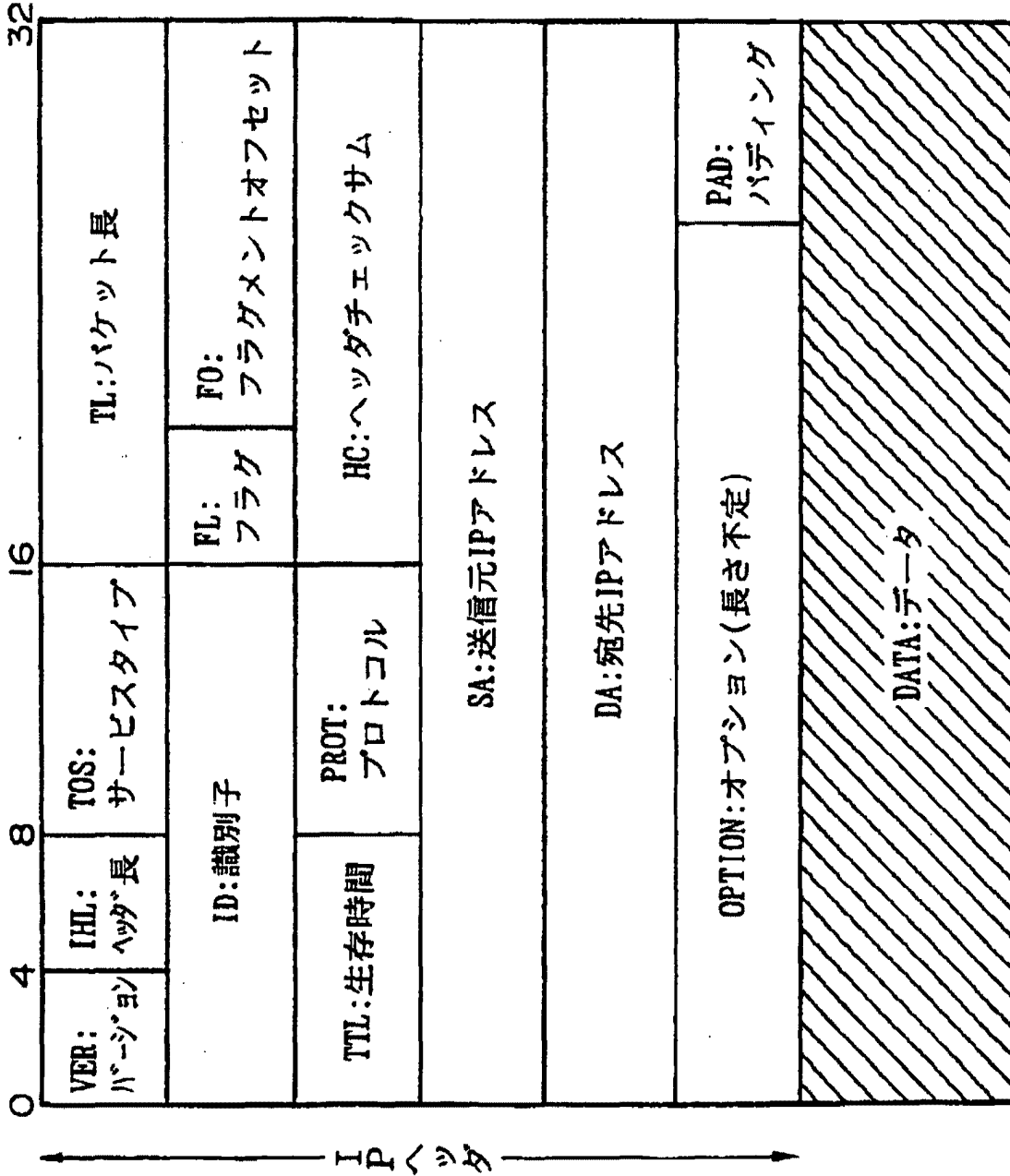


図 19

【図20】

[Figure 20]

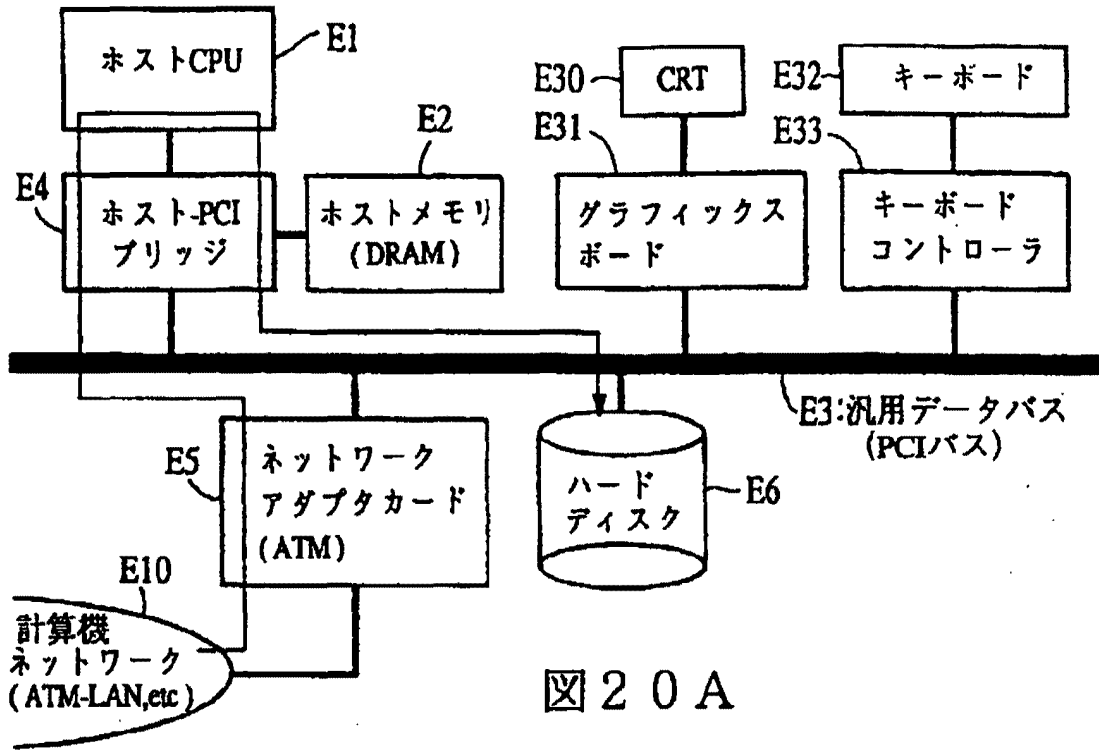


図20A

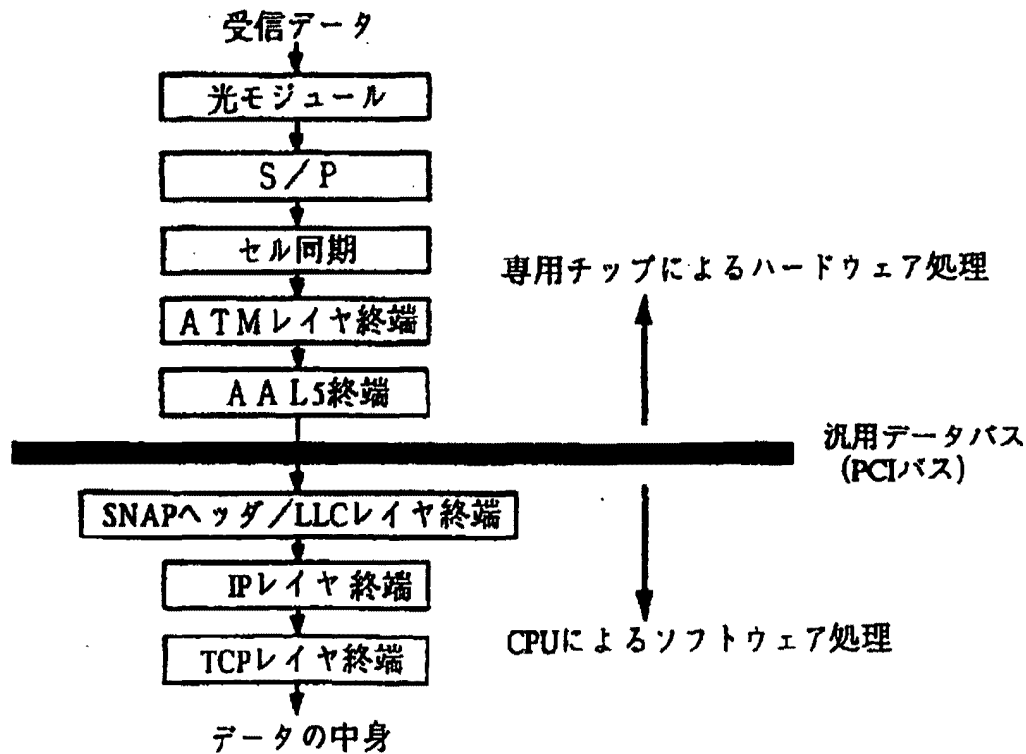


図20B

【図21】

{Figure 21 }

プロトコルスタック

ハードウェア

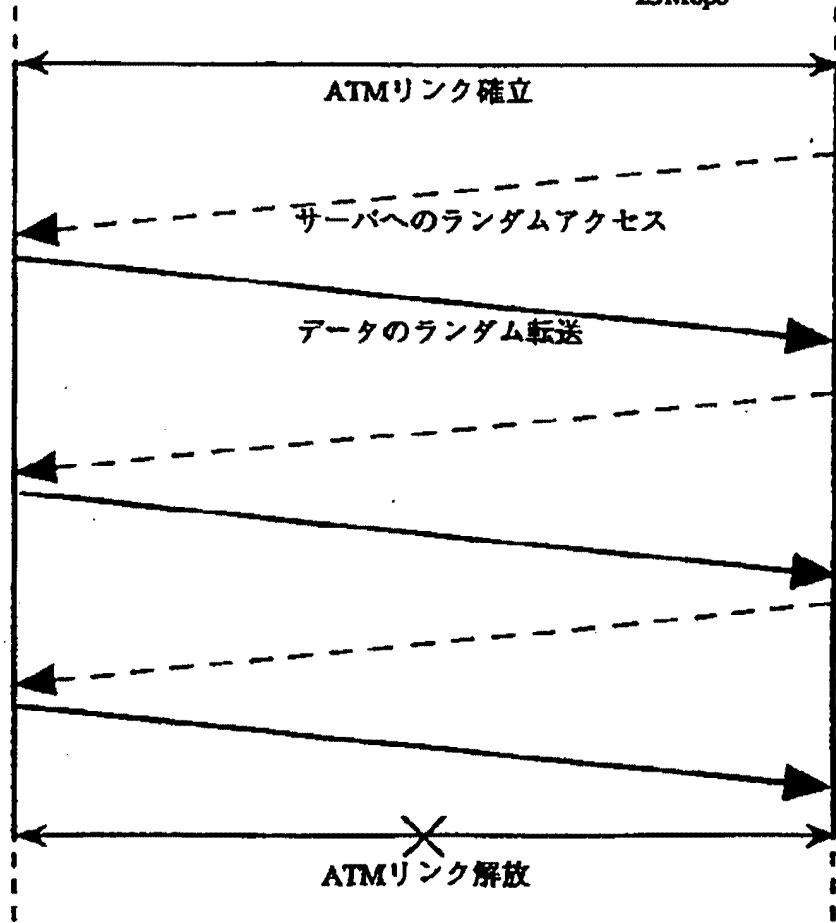
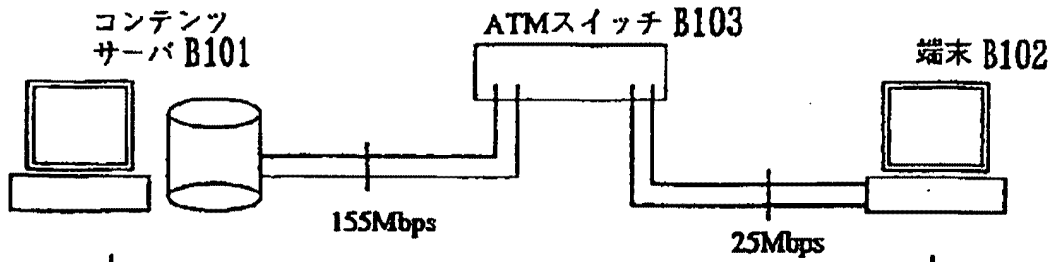
ftp (アプリケーション)	CPU
TCP	
IP	
SNAP/LLC	
AAL(type5)	SARchip
ATM	ATMchip
PHY	光モジュール、S/P変換、セル同期回路

図 2 1

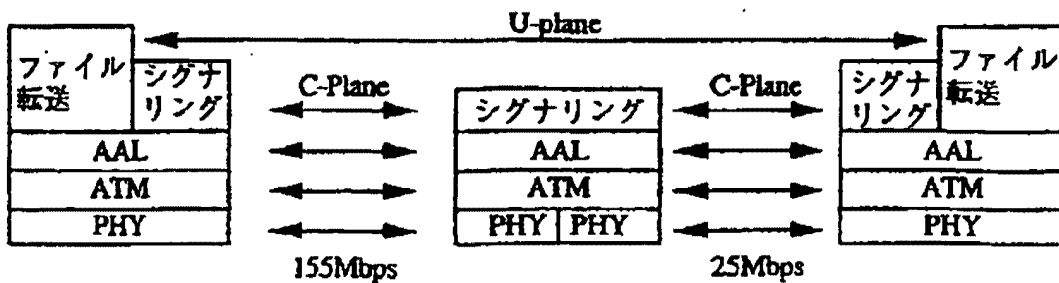
【図22】

{Figure 22 }





↓ プロトコルスタック

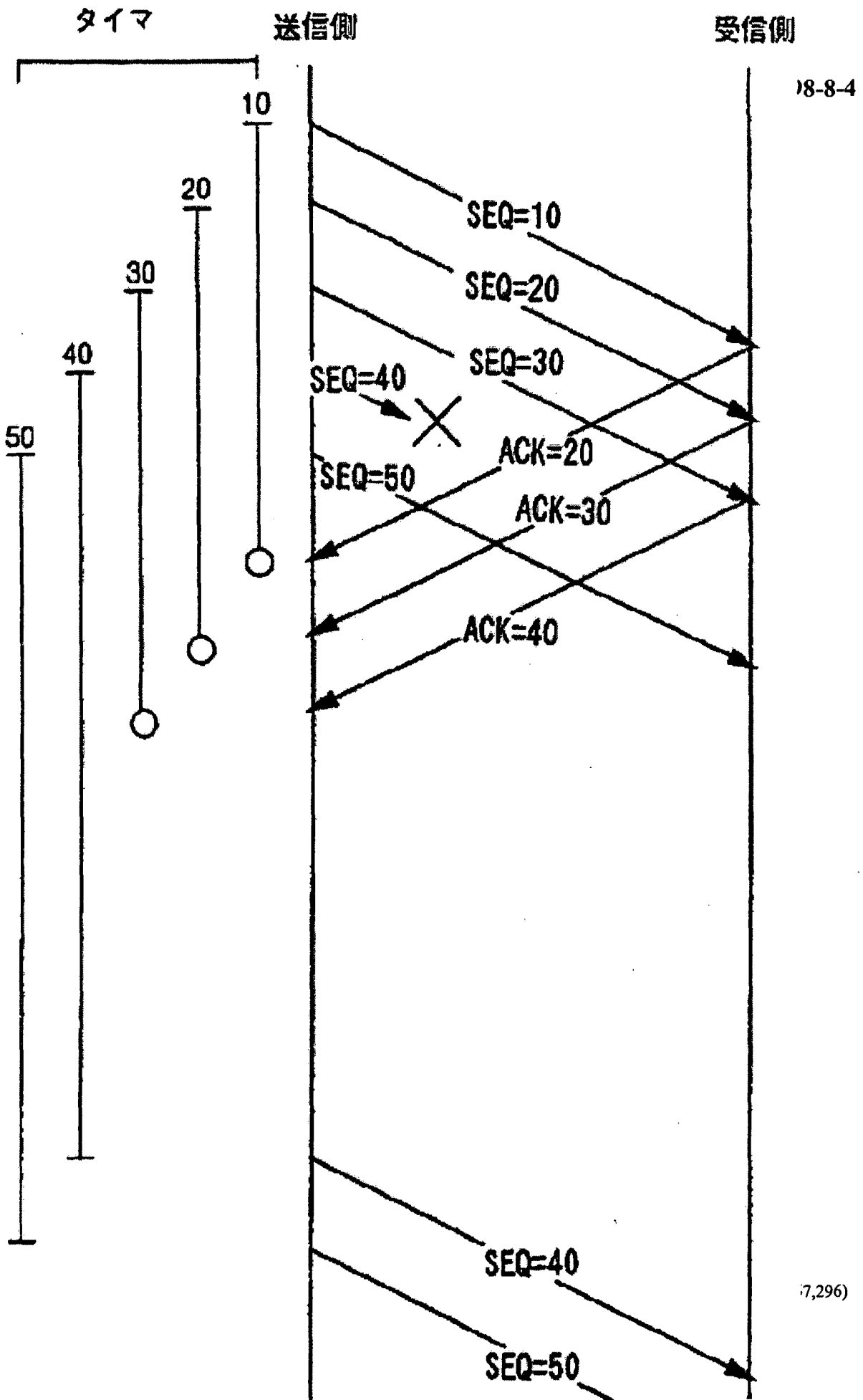


(7,296)

図 2 2

【23】

{Figure 23 }



18-8-4

(7,296)

**WO1997033227A1**

**1998-8-4**

**【國際調查報告】**

B 12



AU9064125

**(12) PATENT ABRIDGMENT (11) Document No. AU-B-64125/90**  
**(19) AUSTRALIAN PATENT OFFICE (10) Acceptance No. 647414**

- (54) Title  
**PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE**
- International Patent Classification(s)
- (51)<sup>s</sup> **G06F 015/16**
- (21) Application No. : **64125/90** (22) Application Date : **20.08.90**
- (87) PCT Publication Number : **WO91/03788**
- (30) Priority Data
- (31) Number (32) Date (33) Country  
**404959 08.09.89 US UNITED STATES OF AMERICA**
- (43) Publication Date : **08.04.91**
- (44) Publication Date of Accepted Application : **24.03.94**
- (71) Applicant(s)  
**AUSPEX SYSTEMS, INC.**
- (72) Inventor(s)  
**EDWARD JOHN ROW; LAURENCE B. BOUCHER; WILLIAM M. PITTS; STEPHEN E. BLIGHTMAN**
- (74) Attorney or Agent  
**DAVIES COLLISON CAVE , 1 Little Collins Street, MELBOURNE VIC 3000**
- (56) Prior Art Documents  
**AU 679806 53068/86 G06F 13/12**
- (57) Claim

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

.../2

(11) AU-B-64125/90

-2-

(10) 647414

including all requests by a network client to run  
client-defined procedures on said network server  
apparatus.

PCT

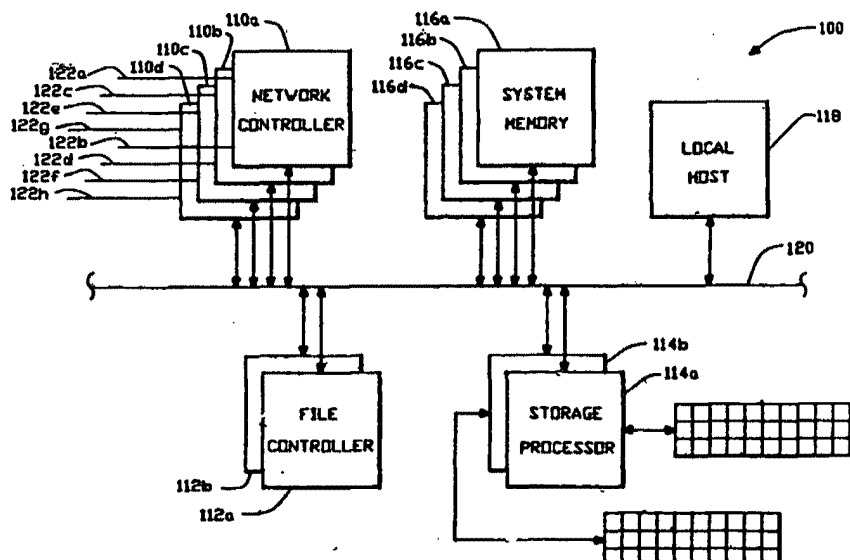
AOJP DATE 16/05/91

PCT NUMBER PCT/US90/04711

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

<p>(51) International Patent Classification 5 : <b>G06F 15/16</b></p>	<p><b>A1</b></p>	<p>(11) International Publication Number: <b>WO 91/03788</b> (43) International Publication Date: <b>21 March 1991 (21.03.91)</b></p>
<p>(21) International Application Number: <b>PCT/US90/04711</b> (22) International Filing Date: <b>20 August 1990 (20.08.90)</b> (30) Priority data: <b>404,959</b>                      <b>8 September 1989 (08.09.89)</b>    <b>US</b> (71) Applicant: <b>AUSPEX SYSTEMS, INC. [US/US]; 2952 Bunker Hill Lane, Santa Clara, CA 95054 (US).</b> (72) Inventors: <b>ROW, Edward, John ; 468 Mountain Laurel Court, Mountain View, CA 94064 (US). BOUCHER, Laurence, B. ; 20605 Montalvo Heights Drive, Saratoga, CA 95070 (US). PITTS, William, M. ; 780 Mora Drive, Los Altos, CA 94022 (US). BLIGHTMAN, Stephen, E. ; 775 Salt Lake Drive, San Jose, CA 95133 (US).</b></p>	<p>(74) Agents: <b>FLIESLER, Martin, C. et al.; Fliesler, Dubb, Meyer &amp; Lovejoy, 4 Embarcadero Center, Suite 400, San Francisco, CA 94111 (US).</b> (81) Designated States: <b>AT (European patent), AU, BE (European patent), CA, CH (European patent), DE (European patent)*, DK (European patent), ES (European patent), FR (European patent), GB (European patent), IT (European patent), JP, KR, LU (European patent), NL (European patent), SE (European patent).</b></p> <p><b>Published</b> <i>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.</i></p> <p style="font-size: 2em; text-align: center;"><b>647414</b></p>	

(54) Title: PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE



(57) Abstract

A file server architecture is disclosed, comprising as separate processors, a network controller unit (110), a file controller unit (112) and a storage processor unit (114). These units incorporate their own processors, and operate in parallel with a local Unix host processor (118). All networks are connected to the network controller unit (110), which performs all protocol processing up through the NFS layer. The virtual file system is implemented in the file controller unit (112) and the storage processor (114) provides high-speed multiplexed access to an array of mass storage devices. The file controller unit (112) controls file information caching through its own local cache buffer, and controls disk data caching through a large system memory which is accessible on a bus by any of the processors.

PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE

5

10           The present application is related to the following published International Patent Applications:

          1.    MULTIPLE    FACILITY    OPERATING    SYSTEM  
ARCHITECTURE, invented by David Hitz, Allan Schwartz,  
James Lau and Guy Harris, PCT Publication No.  
15    WO91/04540, international filing date April 4, 1991;

          2.    ENHANCED    VMEBUS    PROTOCOL    UTILIZING  
PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA  
TRANSFER, invented by Daryl Starr, PCT Publication No.  
          WO91/03786, international filing date March 21, 1991;  
20    and

          3.    BUS LOCKING FIFO MULTI-PROCESSOR COMMUNICATIONS  
SYSTEM UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND  
BLOCK MODE DATA TRANSFER invented by Daryl D. Starr,  
William Pitts and Stephen Blightman, PCT Publication  
25    No. WO91/11768, international filing date August 8,  
1991,

          The above applications are all assigned to the  
assignee of the present invention and are all expressly  
incorporated herein by reference.





-2-

BACKGROUND OF THE INVENTIONField of the Invention

5 The invention relates to computer data networks, and more particularly, to network file server architectures for computer networks.

Description of the Related Art

10 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  
15 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.

20 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  
25 exchange service requests and responses with server nodes.

30 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  
35 (NetWare, 3Plus, Vines, LANManager, LANServer); AppleTalk for MacIntoshes; and TCP/IP with NFS or RFS

**SUBSTITUTE SHEET**

-3-

for Unix. These protocols are all well-known in the industry.

5 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; 10 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. 15

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 20 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. 25 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 30 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 35 defined in a document entitled, "NFS: Network File

**SUBSTITUTE SHEET**

-4-

System Protocol Specification," Request For Comments (RFC) 1094, by Sun Microsystems, Inc. (March 1989). This document is incorporated herein by reference in its entirety.

5           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  
10           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  
15           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  
20           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  
25           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,  
30           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  
35           local networks together.

**SUBSTITUTE SHEET**

-5-

In addition to performance considerations, the low client-to-server ratio creates computing problems in several additional ways:

5           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.

10           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.

15           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).

20           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.

25           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,

**SUBSTITUTE SHEET**

-6-

extra primary memory, or even a faster processor do not resolve basic architectural I/O constraints; I/O throughput does not increase sufficiently.

5 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.

10 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.

25  
30  
35 In yet another computer architecture, a plurality of CPUs, each having its own cache memory for data and

**SUBSTITUTE SHEET**

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  
5 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  
10 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  
15 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

20

In accordance with the present invention there is provided a 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;

25

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  
5 to run client-defined procedures on said network server apparatus.

### BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described with respect to particular embodiments  
10 thereof, and reference will be made to the drawings, in which:

Fig. 1. is a block diagram of a prior art file server architecture;



-9-

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;

5 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;

10 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

15 Figs. 8A-C are a flowchart illustrating the operation of a fast transfer protocol BLOCK READ cycle.

#### DETAILED DESCRIPTION

20 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

25 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

30 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

35 32, and possibly a network #2 controller 34 connected

**SUBSTITUTE SHEET**



-10-

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  
5 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  
10 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  
15 implemented in the apparatus of Fig. 1:

Network Layer. The network layer converts data packets between a format specific to Ethernets and a format which is independent of the particular type of network used. the Ethernet-specific format which is  
20 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  
25 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  
30 fragment, and calls on the network layer to transmit the resulting internet datagrams. The internet header  
35

SUBSTITUTE SHEET

-11-

includes a local network destination address (translated from the internet address) as well as other parameters.

5 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, 10 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 20 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 25 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 30 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 35 services, and uses TCP for certain other services

**SUBSTITUTE SHEET**

-12-

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.

**SUBSTITUTE SHEET**

-13-

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

**SUBSTITUTE SHEET**

-14-

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  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 card 10, so the processing of these requests is not accomplished in a truly parallel manner. The

**SUBSTITUTE SHEET**

-15-

processes are instead merely time sliced. The CPU 10 therefore represents a major bottleneck in the processing of file server requests.

5 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.

10 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

15 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  
20 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  
25 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  
30 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  
35 each support up to three SCSI disk drives each. The tenth SCSI channel on each of the storage processors

SUBSTITUTE SHEET

-16-

114 is used for tape drives and other SCSI peripherals.

5 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 10 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 15 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.

#### 20 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 25 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: 30 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 35 the use of a messaging kernel running on each of the

SUBSTITUTE PAGE

-17-

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 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

**SUBSTITUTE SHEET**



-18-

server is precisely standard, licensed NFS. LNFS is described in more detail below.

5 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  
10 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  
15 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  
20 processors will now be set forth.

#### Network Controller 110

The optimized dataflow of the server 100 begins with the intelligent network controller 110. This  
25 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  
30 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  
35 in the Network Controller 110, not in the host 118.

**SUBSTITUTE SHEET**

-19-

The server 100 uses conventional NFS ported from Sun Microsystems, Inc., Mountain View, CA, and is NFS protocol compatible.

5 Non-NFS network traffic is passed directly to its destination host processor 118.

10 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-  
15 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.  
20

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.

#### 25 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  
30 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.

**SUBSTITUTE SHEET**

-20-

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  
5 110 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  
10 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  
15 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  
20 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  
25 involvement of the host 118.

#### Storage Processor

The intelligent storage processor 114 manages all disk and tape storage operations. While autonomous,  
30 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  
35 remote clients.

**SUBSTITUTE SHEET**

-21-

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.

**SUBSTITUTE SHEET**

-22-

The host processor 118 runs the following software modules:

5        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.

10        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.

15        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 20 112. Normally, the host 118 uses LNFS and goes through the file controller.

25        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.

30        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.

35        Yellow Pages and Authentication Demon. The Yellow Pages and Authentication services are Sun-ONC standard

**SUBSTITUTE SHEET**

-23-

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

**SUBSTITUTE SHEET**

-24-

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  
5 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  
10 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  
15 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  
20 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  
25 232. 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  
30 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  
35 microprocessor data bus 212 via a bidirectional buffer 250, and has similar components to those appearing on

**SUBSTITUTE SHEET**

-25-

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

**SUBSTITUTE SHEET**



the CPU memory 214 via the bidirectional buffer 284. LAN memories 236 and 256 are not directly addressed over VME bus 120.

5 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 message passing. Referring to the Appendix, the FIFOs 240 and 260 are programmed with  
10 the following bit settings in the Data Transfer Configuration Register:

Bit	Definition	Setting
0	WD Mode	N/A
1	Parity Chip	N/A
15 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)
20 7	Reset	no (0)

The Data Transfer Control Register is programmed as follows:

Bit	Definition	Setting
0	Enable PortA Req/Ack	yes (1)
25 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)
30 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  
35 reads an entire 32-bit word from port A with a single instruction using two port A access cycles. Port A

**SUBSTITUTE SHEET**

-27-

data transfer is disabled by unsetting bits 0 (Enable PortA Req/Ack) and 7 (PortA Master) of the Data Transfer Control Register.

5 The remainder of the control settings in FIFO 270 are the same as those in FIFOs 240 and 260 described above.

10 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.

25 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.

30 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 35 236 on the LAN data bus 232 (or 256 on the LAN data

**SUBSTITUTE SHEET**

-28-

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.

**SUBSTITUTE SHEET**

-29-

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

**SUBSTITUTE SHEET**

-30-

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  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 118 that a packet is available at a specified system memory address. The microprocessor 210 sends such a

**SUBSTITUTE SHEET**

-31-

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  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 memory 256 takes place similarly to the copying described above from LAN memory to system memory. For

**SUBSTITUTE SHEET**

-32-

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  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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,  
35 in contrast to the prior art architecture of Fig. 1. Moreover, all but the shortest copying work is

**SUBSTITUTE SHEET**

-33-

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 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

**SUBSTITUTE SHEET**



-34-

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.

5           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  
10       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  
15       NFS: get attributes of a file (GETATTR); set attributes of a file (SETATTR); look up a file (LOOKUP); create a file (CREATE); remove a file (REMOVE); rename a file (RENAME); create a new linked file (LINK); create a symlink (SYMLINK); remove a  
20       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  
25       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.

30           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  
35       LAN DMA controller 242, parity FIFO 240 and VME/FIFO DMA controller 272 to transmit the entire block of

SUBSTITUTE SHEET

-35-

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

**SUBSTITUTE SHEET**

-36-

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  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 transfer count to the next 128-byte boundary. The extra 32-bit word transfers do not involve the VME

**SUBSTITUTE SHEET**

-37-

bus, and only the desired number of 32-bit words will be unloaded from port A.

5 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  
10 128 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.

#### 15 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  
20 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  
25 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  
30 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  
35 as those described above with respect to EPROM 220 and

**SUBSTITUTE SHEET**

-38-

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.

5           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  
10       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  
15       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  
20       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  
25       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  
30       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  
35       memory 396, however, with the FC 112a acting as an VME bus slave. In such transfers, the remote master

**SUBSTITUTE SHEET**

-39-

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.

5     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

**SUBSTITUTE SHEET**

-40-

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

**SUBSTITUTE SHEET**

-41-

consulting (Santa Clara, California: 1988), incorporated herein by reference.

5 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 10 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.

15 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.

20 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 25 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.

30 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 35 from disk if it is not already locally cached. Also,

**SUBSTITUTE SHEET**



-42-

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.

5       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.

10       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.

15       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

35       In the file server 100, each of the storage processors 114 can interface the VME bus 120 with up

**SUBSTITUTE SHEET**

-43-

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.

5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 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  
 5 incorporated herein by reference. The eight  
 general-purpose I/O bits provided by MFP 524 are  
 configured according to the following table:

	<u>Bit</u>	<u>Direction</u>	<u>Definition</u>
10	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.
15	5	input	Channel Operation Done - A composite of the channel done bits from all 13 channels of the DMA controller, described below.
20	4	output	DMA Controller Enable. Enables the DMA Controller to run.
25	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.
30	1	output	External Interrupts Disable. Disables externally generated interrupts to the microprocessor 510.
35	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  
 40 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-  
 45 only register as seen by microprocessor 510. If the

**SUBSTITUTE SHEET**

-45-

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 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.

**SUBSTITUTE SHEET**

-46-

5 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.

15 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.

35 The parity FIFOs 544 are each configured in their Adaptec 6250 mode. Referring to the Appendix, the

**SUBSTITUTE SHEET**

-47-

FIFOs 544 are programmed with the following bit settings in the Data Transfer Configuration Register:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
	0	WD Mode	(0)
5	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)
10	6	Checksum Carry Wrap	(0)
	7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
15	0	Enable PortA Req/Ack	(1)
	1	Enable PortB Req/Ack	(1)
	2	Data Transfer Direction	as desired
	3	CPU parity enable	(0)
20	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

**SUBSTITUTE SHEET**

the following bit settings in the Data Transfer Configuration Register:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
	0	WD Mode	(0)
5	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)
10	6	Checksum Carry Wrap	(0)
	7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
15	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)
20	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.

25 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. FIFQ 554 provides one means by which the microprocessor 510 can communicate directly with the VME bus 120, as is described in more detail below.

30 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  
 35 the control bus 516. The channel enable register 560

**SUBSTITUTE SHEET**

-49-

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.

**SUBSTITUTE SHEET**



-50-

15 This is a null channel for which neither a ready status nor done status is generated.

5 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.

10 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 15 572. 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 20 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.

25 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.

30 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 35 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

**SUBSTITUTE SHEET**

-51-

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

**SUBSTITUTE SHEET**

-52-

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)

**SUBSTITUTE SHEET**

-53-

and the registers 382 on FC 112a (Fig. 4). The details of these registers are not important for an understanding of the present invention.

5     STORAGE PROCESSOR OPERATION

10     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.

20     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.

**SUBSTITUTE SHEET**

-54-

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.

5  
10 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
- 20 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
- 25 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 and 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.

35

**SUBSTITUTE SHEET**

-55-

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,

**SUBSTITUTE SHEET**

-56-

recovery actions similar to data check recovery take place.

#### Read/Write Cache Pages

5           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  
10           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-  
15           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  
20           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  
25           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  
30           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:

**SUBSTITUTE SHEET**

OP CODE OPERATION

	0	NO-OP	
5	1	ZEROES -> BUFFER	Move zeros from zeros register 576 to local data buffer 564.
10	2	ZEROES -> FIFO	Move zeros from zeros register 576 to the currently selected FIFO on common data bus 550.
15	3	ZEROES -> VMEbus	Move zeros from zeros register 576 out onto the VME bus 120. Used for initializing cache buffers in system memory 116.
20			
25	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.
30			
35			
40	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.
45			
50	6	VMEbus -> BUFFER & FIFO	Target data is moved from VME bus 120 to a SCSI

**SUBSTITUTE SHEET**



-58-

5			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.
10	7	BUFFER -> VMEbus	This operation is not used.
15	8	BUFFER -> FIFO	Participating data is transferred to create redundant data or recovered data on a disk drive. Used only in RAID 5 applications.
20	9	FIFO -> VMEbus	This operation is used to move target data directly from a disk drive onto the VME bus 120.
25	A	FIFO -> BUFFER	Used to move participating data for recovery and modify operations. Used only in RAID 5 applications.
30			
35	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

40 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

45

**SUBSTITUTE SHEET**

-59-

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.

#### 15 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

**SUBSTITUTE SHEET**

-60-

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.

5

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.

10

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.

15

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.

20

25

30

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:

35

**SUBSTITUTE SHEET**

-61-

	<u>Type</u>	<u>Description</u>
	0	Pointer to a new message being sent
	1	Pointer to a reply message
	2	Pointer to message to be forwarded
5	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

**SUBSTITUTE SHEET**

-62-

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

**SUBSTITUTE SHEET**

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  
5 descriptor then reflects the VME address of the message in system memory 116.

In summary, the embodiments of the present invention involve a new, server-specific I/O architecture that is optimised for a Unix file server's most common actions -- file operations. Roughly stated, a file server architecture is provided which  
10 comprises 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  
15 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  
20 optimised 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  
25 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  
5 instruction fetches and transmissions of message and file data.

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.



-64-

APPENDIX AVME/FIFO DMA Controller

5 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  
10 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.

15 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  
20 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  
25 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  
30 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  
35 address while a true condition causes execution from

SUBSTITUTE SHEET



-65-

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.

5            ALU. The alu comprises a single IDT49C402A integrated circuit. It is 16 bits in width and most  
10           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  
15           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  
20           below along with mnemonics and a description of each function.

25	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.
30	CIN            38	(Carry INput) This bit forces the carry input to the alu.
35	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.
40	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.

SUBSTITUTE SHEET

5 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.

10 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

- 20 0 This value selects a null flag.
- 25 1 When set this bit enables the buffer clock. When reset this bit disables the buffer clock.
- 30 2 When this bit is cleared VME bus transfers, buffer operations and RAS are all disabled.
- 35 3 NOT USED
- 40 4 When set this bit enables VME bus transfers.
- 45 5 When set this bit enables buffer operations.
- 6 6 When set this bit asserts the row address strobe to the dram buffer.
- 7 7 When set this bit selects page 0 of the control store.

50 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

**SUBSTITUTE SHEET**

-68-

		loads the data from the alu bus into the dram address counters.
5	3	D<14:7> -> mux addr<8:0> WR_VADL loads the data from the alu bus into the least significant 2 bytes of the VME address register.
10		D<15:2> -> VME addr<15:2> D1 -> ENB_tional registers D<15:2> -> VME addr<15:2> D1 -> ENB_ELK D0 -> ENB_BLK
15	4	WR_VADH loads the most significant 2 bytes of the VME address register.
20		D<15:0> -> VME addr<31:16>
	5	WR_RADD loads the dual ported ram address counters.
25		D<10:0> -> ram addr <10:0>
	6	WR_WCNT loads the word counters. D15 -> count enable* D<14:8> -> count <6:0>
30	7	WR_CO loads the co-channel select register.
35		D<7:4> -> CO<3:0>
	8	WR_NXT loads the next-channel select register.
40		D<3:0> -> NEXT<3:0>
	9	WR_CUR loads the current-channel select register.
45		D<3:0> -> CURR <3:0>
	10:14	RESERVED - NOT DEFINED
	15	JUMP causes the control store sequencer to select the alu data bus.
50		D<8:0> -> CS_A<8:0>

**SUBSTITUTE SHEET**

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.

5

TEST<3:0> Condition

10	0	FALSE	-always false
	1	TRUE	-always true
	2	ALU_COUT	-carry output of alu
	3	ALU_EQ	-equals output of alu
15	4	ALU_OVR	-alu overflow
	5	ALU_NEG	-alu negative
20	6	XFR_DONE	-transfer complete
	7	PAR_ERR	-buffer parity error
	8	TIMOUT	-bus operation timeout
25	9	ANY_ERR	-any error status
	14:10	RESERVED	-NOT DEFINED
30	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.

35

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.

40

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.

45

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.

5 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
10	0	CURR POINTER 0   STATUS 0
	4	INITIAL POINTER 0
15		
	58	CURR POINTER B   STATUS B
	5C	INITIAL POINTER B
20		
	60	not used   not used
	64	not used   not used
25		
	68	CURR POINTER D   STATUS D
	6C	INITIAL POINTER D
30		
	70	not used   STATUS E
	74	HIGH SPEED BUS ADDRESS 31:2 0 0
	78	PARAMETER BLOCK 0
35		
	??	PARAMETER BLOCK n

40 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  
 45 initialized to 0x0000 by the microprocessor 510 before enabling the channel. Upon detecting a value of 0x0000

**SUBSTITUTE SHEET**

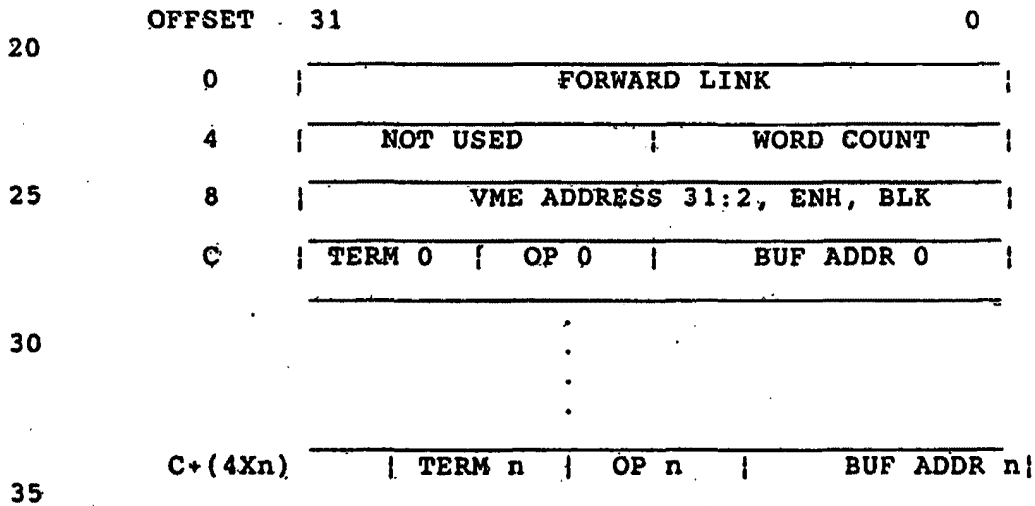
in the current block pointer the DMA controller 580 will copy the lower 16 bits from the initial pointer to the current pointer. Once the DMA controller 580 has completed the specified operations for the parameter block the current pointer will be updated to point to the next block. In the event that 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
- 15 2 BUS OPERATION TIMEOUT
- 3 BUS OPERATION ERROR
- 4 DATA PATH PARITY ERROR

The format of the parameter block is:



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:

**SUBSTITUTE SHEET**

-72-

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.

5

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:

10

|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.

15

20

VME ADDRESS - The VME address specifies the starting address for data transfers. Thirty bits allows the address to start at any quad byte boundary.

25

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.

30

35

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

**SUBSTITUTE SHEET**



-73-

when performing any transfer to or from 16 bit address space.

5           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.

10           |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|0|T|

          OP - Operations are specified by the op code. The op code byte has the following format:

25           |0|0|0|0|OP3|OP2|OP1|OP0|

The op codes are listed below ("FIFO" refers to any of the FIFOs 544 or 554):

**SUBSTITUTE SHEET**

-74-

	<u>OP CODE</u>	<u>OPERATION</u>
	0	NO-OP
	1	ZEROES -> BUFFER
	2	ZEROES -> FIFO
5	3	ZEROES -> VMEbus
	4	VMEbus -> BUFFER
	5	VMEbus -> FIFO
	6	VMEbus -> BUFFER & FIFO
	7	BUFFER -> VMEbus
10	8	BUFFER -> FIFO
	9	FIFO -> VMEbus
	A	FIFO -> BUFFER
	B	FIFO -> VMEbus & BUFFER
	C	RESERVED
15	D	RESERVED
	E	RESERVED
	F	RESERVED

**SUBSTITUTE SHEET**

-75-

APPENDIX BEnhanced VME Block Transfer Protocol

5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 of large amounts of data across a VME backplane bus by  
substantially increasing the data transfer rate.

**SUBSTITUTE SHEET**

-76-

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.

5       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,  
10       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  
15       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  
20       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  
25       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  
30       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  
35       transfer cycle, the master will assert and deassert DS0\* in the form of a pulse of a given duration. The

**SUBSTITUTE SHEET**

-77-

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

**SUBSTITUTE SHEET**

-78-

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 DSO\*. 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 DSO 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

**SUBSTITUTE SHEET**

-79-

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.

5 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.

10 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.

15 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.

20 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.

**SUBSTITUTE SHEET**

-80-

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

**SUBSTITUTE SHEET**



-81-

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.

5           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  
10           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  
15           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.  
20           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  
25           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  
30           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  
35           the data being transmitted by the slave over data

**SUBSTITUTE SHEET**

-82-

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.

5           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.

10           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.

15           The slave drives DTACK\* low, 845, and, after a predetermined period of time, drives DTACK\* high, 847.

20           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.

25           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

30

35

**SUBSTITUTE SHEET**

-83-

receipt of the DTACK\* high to low transition, the master will drive AS\* high 859 and release the AS\* line 861.

5 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  
10 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  
15 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.

**SUBSTITUTE SHEET**

-84-

APPENDIX CParity FIFO

5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 slave is responding. In slave mode, however, the Port  
A REQ input specifies that the master requires a data

**SUBSTITUTE SHEET**

-85-

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.

5     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.

10    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

15    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.

20    Clk

The clock input is used to generate some of the chip timing. It is expected to be in the 10-20 Mhz range.

25    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  
30    conjunction with Port A Ack.

**SUBSTITUTE SHEET**

-86-

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, Port B 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,

**SUBSTITUTE SHEET**

-87-

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.

5  
10  
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.

15  
20  
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.

25  
30  
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.

35  
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

**SUBSTITUTE SHEET**

-88-

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.

5  
10        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.

15  
20  
25        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.

30  
35        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

**SUBSTITUTE SHEET**



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
5	8	False	False	Port A byte 0	Port A byte 1	Port A byte 2	Port A byte 1
10	8	False	True	Port A byte 1	Port A byte 0	Port A byte 3	Port A byte 2
15	8	True	False	Port A byte 2	Port A byte 3	Port A byte 0	Port A byte 1
20	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
25	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
30	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

**SUBSTITUTE SHEET**

-90-

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.

5 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  
10 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.  
15 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  
20 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

##### 25 Data Transfer Configuration Register (Read/Write)

Register Address 0. This register is cleared by the reset signal.

- 30 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.
- 35 Bit 2 Parity Correct Mode. Set if the parity chip is to correct parallel parity on Port B.

**SUBSTITUTE SHEET**

-91-

- 5 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.
- 10 Bit 4 Invert Port A byte address 0. Set to invert the least significant bit of Port A byte address.
- 15 Bit 5 Invert Port A byte address 1. Set to invert the most significant bit of Port A byte address.
- 20 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.
- 25 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.
- 30 Data Transfer Control Register (Read/Write)  
 Register Address 1. This register is cleared by the reset signal or by writing to the reset bit.
- 35 Bit 0 Enable Data Transfer on Port A. Set to enable the Port A Req/Ack handshake.
- 40 Bit 1 Enable Data Transfer on Port B. Set to enable the Port B Req/Ack handshake.
- 45 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.

SUBSTITUTE SHEET

-92-

- 5  
10  
15  
20  
25  
30  
35  
40
- 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)

45     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.

**SUBSTITUTE SHEET**

-93-

- 5 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.
- 10 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.
- 15 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.
- 20
- 25 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)

30 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.

- 35 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.
- 40 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.
- 45 Bit 2 Port A to RAM Y. Set if Port A is accessing RAM Y, and reset if it is accessing RAM X.

**SUBSTITUTE SHEET**

-94-

- Bit 3 Port B to RAM Y. Set if Port B is accessing RAM Y, and reset if it is accessing RAM X .
- 5 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.
- 10

Bits 5-7 Not Used.

15 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.

20

Bits 0-4 RAM X word address  
 Bit 5 RAM X full  
 Bits 6-7 Not Used

25 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.

30

Bits 0-4 RAM Y word address  
 Bit 5 RAM Y full  
 Bits 6-7 Not Used

35 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

**SUBSTITUTE SHEET**

-95-

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

**SUBSTITUTE SHEET**

-96-

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.

5

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.

10

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.

15

```

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)

```

30

35

40

SUBSTITUTE SHEET



```

/*****
    The following routine initiates a Fifo data transfer using two
    values passed to it.
5      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 A and Port B
10     handshakes, both data transfer enables bits should be
                    set in this parameter.
*****/

FIFO_initiate_data_transfer(config_data, control_data)
15  unsigned char config_data, control_data;
    {
        FIFO1->config = config_data | FIFO_RESET; /* Set
        Configuration value & Reset */
        FIFO1->control = control_data & (~FIFO_PORT_ENABLES); /* Set
20     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.
30     *****/

FIFO_force_odd_length_transfer()
    {
        FIFO1->control &= ~FIFO_PORT_ENABLES; /* Disable Ports A & B
35     */
        if (PORT_A_TO_PORT_B(FIFO1)) {
            if (PORT_A_TO_RAM_Y(FIFO1)) {
                FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL; /*
40     Set RAM Y full */
            }
            else FIFO1->ram_X_addr = FIFO_FORCE_RAM_FULL; /* Set
        RAM X full */
            FIFO1->control |= FIFO_PORT_B_ENABLE; /*
45     Re-Enable Port B */
        }
        else {
            if (PORT_B_TO_RAM_Y(FIFO1)) {
                FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL; /*
50     Set RAM Y full */
            }
            else FIFO1->ram_X_addr = FIFO_FORCE_RAM_FULL; /* Set
        RAM X full */
    }

```

**SUBSTITUTE SHEET**

```

        FIFO1->control |= FIFO_PORT_A_ENABLE ;      /*
Re-Enable Port A */
    }
}

5  /*****
    The following routine returns how many odd bytes have been
    left in the Fifo at the end of a data transfer.
    *****/

10  int FIFO_count_odd_bytes()
    {
        int number_odd_bytes;
        number_odd_bytes=0;
        if (FIFO1->status & FIFO_DATA_IN_RAM) {
15      if (PORT_A_TO_PORT_B(FIFO1)) {
                number_odd_bytes =
                (PORT_A_BYTE_ADDRESS(FIFO1)) ;
                if (PORT_A_TO_RAM_Y(FIFO1))
20      4 ;
                    number_odd_bytes += (FIFO1->ram_Y_addr) *
                    else number_odd_bytes += (FIFO1->ram_X_addr) * 4 ;
                }
            else {
25      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);
30  }

    /*****
    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
35  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.

40      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
45      Bit 5 = ram Y address register failed
        Bit 6 = data register failed
        Bit 7 = checksum register failed
    *****/

50  #define RAM_DEPTH 64      /* number of long words in Fifo Ram */
    reg_expected_data[6] = { 0x7F, 0xFF, 0x00, 0x1F, 0x3F, 0x3F };

```

- 99 -

```

char FIFO_uprocessor_interface_test()
{
    unsigned long test_data;
    char *register_addr;
    5     int i;
        char j,error;
        FIFO1->config = FIFO_RESET;          /* reset the chip */
        error=0;
        register_addr = (char *) FIFO1;
    10     j=1;

        /* first test registers 0 thru 5 */

        for (i=0; i<6; i++) {
    15         *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 */
    20         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;
    25     }

        /* now test Ram data & checksum registers
        test 1s throughout Ram & then test 0s */

    30     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
    35     */
                FIFO1->data = test_data;
            FIFO1->control = 0;
            for (i=0;i<RAM_DEPTH;i++)
                if (FIFO1->data != test_data) error |= j; /* read &
    40     check data */
            if (FIFO1->checksum) error |= 0x80; /* checksum
        should = 0 */
        }

    45     /* 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 |
    50     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 */

```

SUBSTITUTE SHEET

- 100 -

```
        test_data += 0x04040404;
    }
    test_data = 0x00010203;          /* address pattern start */
    FIFO1->control = FIFO_CHECKSUM_ENABLE;
5   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
10   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
15   */
    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 */
20   return (error);
}
```

**SUBSTITUTE SHEET**

## THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

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 retrieval requests 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.

6. A data control unit for use with a data network and a mass storage device, and in response to file system level storage and retrieval requests from said data network, comprising:

a data bus different from said network;

a buffer memory bank coupled to said bus;

storage processor apparatus coupled to said bus and coupleable to said mass storage device;

file processor apparatus coupled to said bus, said file processor apparatus including a local memory bank;



first 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; and

second means on said file processor unit for translating said file system level retrieval requests into requests to retrieve data from specified physical retrieval locations in said mass storage device, said first and second means for translating collectively including means for caching file control information through said local memory bank in said file processor unit,

said data control unit further comprising means for caching the file data, to be stored or retrieved according to said storage and retrieval requests, through said buffer memory bank.

7. A network node for use with a data network and a mass storage device, comprising:

a system buffer memory;

a host processor unit having direct memory access to said system buffer memory;

a network control unit coupleable to said network and having direct memory access to said system buffer memory;

a data control unit coupleable to said mass storage device and having direct memory access to said system buffer memory;

first means for satisfying requests from said network to store data from said network on said mass storage device;

second means for satisfying requests from said network to retrieve data from said mass storage device to said network; and

third means for transmitting predefined categories of messages from said network to said host processor unit for processing in said host processor unit, said first, second and third means collectively including

means for transmitting from said network control unit to said system memory bank by direct memory access file data from said network for storage on said mass storage device,

means for transmitting from said system memory bank to said data control unit by direct memory access said file data from said network for storage on said mass storage device,

means for transmitting from said data control unit to said system memory bank by direct memory access file data for retrieval from said mass storage device to said network, and

means for transmitting from said system memory bank to said network control unit said file data for retrieval from said mass storage device to said network;



at least said network control unit including a microprocessor and local instruction storage means distinct from said system <sup>2</sup>buffer memory, all instructions for said microprocessor residing in said local instruction storage means.

8. A network file server for use with a data network and a mass storage device, comprising:

a host processor unit running a Unix operating system;

an interface processor unit coupleable to said network and to said mass storage device, said interface processor unit including means for decoding all NFS requests from said network, means for performing all procedures for satisfying said NFS requests, means for encoding any NFS reply messages for return transmission on said network, and means for transmitting predefined non-NFS categories of messages from said network to said host processor unit for processing in said host processor unit.

9. 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.

10. Apparatus according to claim 9, 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.



11. Apparatus according to claim 9, 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.

12. Apparatus according to claim 9, 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.

13. Apparatus according to claim 9, 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.



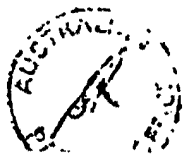
14. Apparatus according to claim 9, further comprising a buffer memory coupled to said interchange bus, and wherein said means for delivering said assembled requests comprises:

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.

15. Apparatus according to claim 9, 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.

16. Apparatus according to claim 9, 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.

17. Apparatus according to claim 9, 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.

18. Apparatus according to claim 17, 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.

19. Apparatus according to claim 9, 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.

20. Apparatus according to claim 9, 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.

21. Apparatus according to claim 20, 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.

22. Apparatus according to claim 21, 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.

23. Apparatus according to claim 20, 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.

24. Apparatus according to claim 23, 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.



25. Apparatus according to claim 20, 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.

26. Apparatus according to claim 20, 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.

27. Apparatus according to claim 9, 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.

28. Apparatus according to claim 27, 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.



29. Apparatus according to claim 27, 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.

30. Apparatus according to claim 9, wherein said network controller comprises:

a microprocessor;

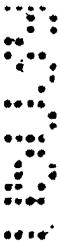
a local instruction memory containing local instruction code;

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.



31. 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.

32. Apparatus according to claim 31, 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

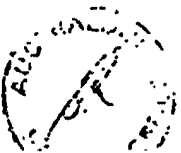


said second class of messages and assembling them into assembled messages in a format which omits said network node destination addresses.

33. Apparatus according to claim 31, 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.

34. Apparatus according to claim 31, 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.

35. Apparatus according to claim 31, 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.

36. Apparatus according to claim 31, 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.

37. Apparatus according to claim 31, 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.

38. Apparatus according to claim 31, 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.

39. File server apparatus for use with a mass storage device, comprising:

a requesting unit capable of issuing calls to file system procedures in a device-independent form;

a file controller including means for converting said file system procedure calls from said device-independent form to a device-specific form and means for issuing device-specific commands in response to at least a subset of said procedure calls, said file controller operating in parallel with said requesting unit; and

a storage processor including means for executing said device-specific commands on said mass storage device, said storage processor operating in parallel with said requesting unit and said file controller.

40. Apparatus according to claim 39, further comprising:

an interchange bus;

first delivery means for delivering said file system procedure calls from said requesting unit to said file controller over said interchange bus; and



second delivery means for delivering said device-specific commands from said file controller to said storage processor over said interchange bus.

41. Apparatus according to claim 39, further comprising:

an interchange bus coupled to said requesting unit and to said file controller;

first memory means in said requesting unit and addressable over said interchange bus;

second memory means in said file controller;

means in said requesting unit for preparing in said first memory means one of said calls to file system procedures;

means for notifying said file controller of the availability of said one of said calls in said first memory means; and

means in said file controller for controlling an access to said first memory means for reading said one of said calls over said interchange bus into said second memory means in response to said notification.

42. Apparatus according to claim 41, wherein said means for notifying said file controller comprises:

a command FIFO in said file controller addressable over said interchange bus; and

means in said requesting unit for controlling an access to said FIFO for writing a descriptor into said FIFO over said interchange bus, said descriptor



describing an address in said first memory means of said one of said calls and an indication that said address points to a message being sent.

43. Apparatus according to claim 41, further comprising:

means in said file controller for controlling an access to said first memory means over said interchange bus for modifying said one of said calls in said first memory means to prepare a reply to said one of said calls; and

means for notifying said requesting unit of the availability of said reply in said first memory.

44. Apparatus according to claim 41, further comprising:

a command FIFO in said requesting processor addressable over said interchange bus; and

means in said file controller for controlling an access to said FIFO for writing a descriptor into said FIFO over said interchange bus, said descriptor describing said address in said first memory and an indication that said address points to a reply to said one of said calls.

45. Apparatus according to claim 39, further comprising:

an interchange bus coupled to said file controller and to said storage processor;

3



second memory means in said file controller and addressable over said interchange bus;

means in said file controller for preparing one of said device-specific commands in said second memory means;

means for notifying said storage processor of the availability of said one of said commands in said second memory means; and

means in said storage processor for controlling an access to said second memory means for reading said one of said commands over said interchange bus in response to said notification.

46. Apparatus according to claim 45, wherein said means for notifying said storage processor comprises:

a command FIFO in said storage processor addressable over said interchange bus; and

means in said file controller for controlling an access to said FIFO for writing a descriptor into said FIFO over said interchange bus, said descriptor describing an address in said second memory of said one of said calls and an indication that said address points to a message being sent.

47. Apparatus according to claim 39, wherein said means for converting said file system procedure calls comprises:



a file control cache in said file controller, storing device-independent to device-specific conversion information; and

means for performing said conversions in accordance with said conversion information in said file control cache.

48. Apparatus according to claim 39, wherein said mass storage device includes a disk drive having numbered sectors, wherein one of said file system procedure calls is a read data procedure call,

said apparatus further comprising an interchange bus and a system buffer memory addressable over said interchange bus,

said means for converting said file system procedure calls including means for issuing a read sectors command in response to one of said read data procedure calls, said read sectors command specifying a starting sector on said disk drive, a count indicating the amount of data to read, and a pointer to a buffer in said system buffer memory, and

said means for executing device-specific commands including means for reading data from said disk drive beginning at said starting sector and continuing for the number of sectors indicated by said count, and controlling an access to said system buffer memory for writing said data over said interchange bus to said buffer in said system buffer memory.

49. Apparatus according to claim 48, wherein said file controller further includes means for determining whether the data specified in said one of said read data procedure calls is already present in said system buffer memory, said means for converting issuing said read sectors command only if said data is not already present in said system buffer memory.

50. Apparatus according to claim 48, further comprising:

means in said storage processor for controlling a notification of said file controller when said read sectors command has been executed;

means in said file controller, responsive to said notification from said storage processor, for controlling a notification of said requesting unit that said read data procedure call has been executed; and

means in said requesting unit, responsive to said notification from said file controller, for controlling an access to said system buffer memory for reading said data over said interchange bus from said buffer in said system buffer memory to said requesting unit.

51. Apparatus according to claim 39, wherein said mass storage device includes a disk drive having numbered sectors, wherein one of said file system procedure calls is a write data procedure call,

5  
5  
5





said apparatus further comprising an interchange bus and a system buffer memory addressable over said interchange bus,

said means for converting said file system procedure calls including means for issuing a write sectors command in response to one of said write data procedure calls, said write data procedure call including a pointer to a buffer in said system buffer memory containing data to be written, and said write sectors command including a starting sector on said disk drive, a count indicating the amount of data to write, and said pointer to said buffer in said buffer memory, and

said means for executing device-specific commands including means for controlling an access to said buffer memory for reading said data over said interchange bus from said buffer in said system buffer memory, and writing said data to said disk drive beginning at said starting sector and continuing for the number of sectors indicated by said count.

52. Apparatus according to claim 51, further comprising:

means in said requesting unit for controlling an access to said system buffer memory for writing said data over said interchange bus to said buffer in said system buffer memory; and

means in said requesting unit for issuing said one of said write data procedure calls when said data has been written to said buffer in said system buffer memory.

53. Apparatus according to claim 52, further comprising:

means in said requesting unit for issuing a buffer allocation request; and

means in said file controller for allocating said buffer in said system buffer memory in response to said buffer allocation request, and for providing said pointer, before said data is written to said buffer in said system buffer memory.

54. Network controller apparatus for use with a first data network carrying signals representing information packets encoded according to a first physical layer protocol, comprising:

a first network interface unit, a first packet bus and first packet memory addressable by said first network interface unit over said first packet bus, said first network interface unit including means for receiving signals over said first network representing incoming information packets, extracting said incoming information packets and writing said incoming information packets into said first packet memory over said first packet bus;

a first packet bus port;

first packet DMA means for reading data over said first packet bus from said first packet memory to said first packet bus port; and

a local processor including means for accessing said incoming information packets in said first packet memory and, in response to the contents of said incoming information packets, controlling said first packet DMA means to read selected data over said first packet bus from said first packet memory to said first packet bus port, said local processor including a CPU, a CPU bus and CPU memory containing CPU instructions, said local processor operating in response to said CPU instructions, said CPU instructions being received by said CPU over said CPU bus independently of any of said writing by said first network interface unit of incoming information packets into said first packet memory over said first packet bus and independently of any of said reading by said first packet DMA means of data over said first packet bus from said first packet memory to said first packet bus port.

55. Apparatus according to claim 54, wherein said first network interface unit further includes means for reading outgoing information packets from said first packet memory over said first packet bus, encoding said outgoing information packets according to said first physical layer protocol, and transmitting signals over



said first network representing said outgoing information packets,

said local processor further including means for preparing said outgoing information packets in said first packet memory, and for controlling said first network interface unit to read, encode and transmit said outgoing information packets,

said receipt of CPU instructions by said CPU over said CPU bus being independent further of any of said reading by said first network interface unit of outgoing information packets from said first packet memory over said first packet bus.

56. Apparatus according to claim 54, further comprising a first FIFO having first and second ports, said first port of said first FIFO being said first packet bus port.

57. Apparatus according to claim 56, for use further with an interchange bus, further comprising interchange bus DMA means for reading data from said second port of said first FIFO onto said interchange bus,

said local processor further including means for controlling said interchange bus DMA means to read said data from said second port of said first FIFO onto said interchange bus.

58. Apparatus according to claim 54, for use further with a second data network carrying signals



representing information packets encoded according to a second physical layer protocol, further comprising:

a second network interface unit, a second packet bus and second packet memory addressable by said second network interface unit over said second packet bus, said second network interface unit including means for reading outgoing information packets from said second packet memory over said second packet bus, encoding said outgoing information packets according to said second physical layer protocol, and transmitting signals over said second network representing said outgoing information packets;

a second packet bus port; and

second packet DMA means for reading data over said second packet bus from said second packet bus port to said second packet memory,

said local processor further including means for controlling said second packet DMA means to read data over said second packet bus from said second packet bus port to said second packet memory, and for controlling said second network interface unit to read, encode and transmit outgoing information packets from said data in said second packet memory,

said receipt of CPU instructions by said CPU over said CPU bus being independent further of any of said reading by said second packet DMA means of data over said second packet bus from said second packet bus



port to said second packet memory, and independent further of any of said reading by said second network interface unit of outgoing information packets from said second packet memory over said second packet bus,

and all of said accesses to said first packet memory over said first packet bus being independent of said accesses to said second packet memory over said second packet bus.

59. Apparatus according to claim 58, wherein said second physical layer protocol is the same as said first physical layer protocol.

60. Apparatus according to claim 58, further comprising means, responsive to signals from said processor, for coupling data from said first packet bus port to said second packet bus port.

61. Apparatus according to claim 61, further comprising:

first and second FIFOs, each having first and second ports, said first port of said first FIFO being said first packet bus port and said first port of said second FIFO being said second packet bus port;

an interchange bus; and

interchange bus DMA means for transferring data between said interchange bus and either said second port of said first FIFO or said second port of said second FIFO, selectably in response to DMA control signals from said local processor.



62. Apparatus according to claim 62, wherein said interchange bus DMA means comprises:

a transfer bus coupled to said second port of said first FIFO and to said second port of said second FIFO;

coupling means coupled between said transfer bus and said interchange bus; and

a controller coupled to receive said DMA control signals from said processor and coupled to said first and second FIFOs and to said coupling means to control data transfers over said transfer bus.

63. Storage processing apparatus for use with a plurality of storage devices on a respective plurality of channel buses, and an interchange bus, said interchange bus capable of transferring data at a higher rate than any of said channel buses, comprising:

data transfer means coupled to each of said channel buses and to said interchange bus, for transferring data in parallel between said data transfer means and each of said channel buses at the data transfer rates of each of said channel buses, respectively, and for transferring data between said data transfer means and said interchange bus at a data transfer rate higher than said data transfer rates of any of said channel buses; and

a local processor including transfer control means for controlling said data transfer means to transfer data between said data transfer means and specified



ones of said channel buses and for controlling said data transfer means to transfer data between said data transfer means and said interchange bus,

said local processor including a CPU, a CPU bus and CPU memory containing CPU instructions, said local processor operating in response to said CPU instructions, said CPU instructions being received by said CPU over said CPU bus independently of any of said data transfers between said channel buses and said data transfer means and independently of any of said data transfers between said data transfer means and said interchange bus.

64. Apparatus according to claim 63, wherein the highest data transfer rate of said interchange bus is substantially equal to the sum of the highest data transfer rates of all of said channel buses.

65. Apparatus according to claim 63, wherein said data transfer means comprises:

a FIFO corresponding to each of said channel buses, each of said FIFOs having a first port and a second port;

a channel adapter coupled between the first port of each of said FIFOs and a respective one of said channels; and

DMA means coupled to the second port of each of said FIFOs and to said interchange bus, for





transferring data between said interchange bus and one of said FIFOs as specified by said local processor,

said transfer control means in said local processor comprising means for controlling each of said channel adapters separately to transfer data between the channel bus coupled to said channel adapter and the FIFO coupled to said channel adapter, and for controlling said DMA controller to transfer data between separately specified ones of said FIFOs and said interchange bus, said DMA means performing said transfers sequentially.

66. Apparatus according to claim 65, wherein said DMA means comprises a command memory and a DMA processor, said local processor having means for writing FIFO/interchange bus DMA commands into said command memory, each of said commands being specific to a given one said FIFOs and including an indication of the direction of data transfer between said interchange bus and said given FIFO, each of said FIFOs generating a ready status indication, said DMA processor controlling the data transfer specified in each of said commands sequentially after the corresponding FIFO indicates a ready status, and notifying said local processor upon completion of the data transfer specified in each of said commands.

67. Apparatus according to claim 65 further comprising an additional FIFO coupled between said CPU



bus and said DMA memory, said local processor further having means for transferring data between said CPU and said additional FIFO, and said DMA means being further for transferring data between said interchange bus and said additional FIFO in response to commands issued by said local processor.

D  
S

68. 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;

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;

means in said interface processor unit for satisfying requests from said host processor unit to store data from said host processor unit on said mass storage device; and

means in said interface processor unit for satisfying requests from said host processor unit to retrieve data from said mass storage device to said host processor unit.

69. Apparatus according to claim 68, 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; and

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.

70. Apparatus according to claim 69, 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.

71. Apparatus according to claim 68, 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 retrieval requests 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.

72. A network node for use with a data network and a mass storage device, comprising:

a system buffer memory;

a network control unit coupleable to said network and having direct memory access to said system buffer memory;

a data control unit coupleable to said mass storage device and having direct memory access to said system buffer memory;

first means for satisfying requests from said network to store data from said network on said mass storage device; and

second means for satisfying requests from said network to retrieve data from said mass storage device to said network, said first and second means collectively including

means for transmitting from said network control unit to said system memory bank by direct memory access file data from said network for storage on said mass storage device,

means for transmitting from said system memory bank to said data control unit by direct memory access said file data from said network for storage on said mass storage device,

means for transmitting from said data control unit to said system memory bank by direct memory access file data for retrieval from said mass storage device to said network, and

means for transmitting from said system memory bank to said network control unit said file data for retrieval from said mass storage device to said network;

at least said network control unit including a microprocessor and local instruction storage means distinct from said system buffer memory, all instructions for said microprocessor residing in said local instruction storage means.

73. A network file server for use with a data network and a mass storage device, comprising:

a host processor unit; and

an interface processor unit coupleable to said network, to said mass storage device and to said host processor unit, said interface processor unit including means for decoding all NFS requests from said network, means for performing all procedures for satisfying said NFS requests, means for encoding any NFS reply messages for return transmission on said network, and means for satisfying file system requests from said host processor unit.

74. 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; and

means in said network controller for satisfying requests received over said interchange bus from said first additional processor.

75. Apparatus according to claim 74, 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.

76. Apparatus according to claim 74, wherein said packets each include a network node destination address, 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.

77. Apparatus according to claim 74, 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.

78. Apparatus according to claim 74, 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.

79. Apparatus according to claim 74, 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.

80. Apparatus according to claim 79, 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.

81. Apparatus according to claim 74, 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.

82. Apparatus according to claim 74, 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.

83. Apparatus according to claim 82, 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,

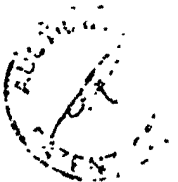
said apparatus further including means for delivering said data to said network controller,

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.

84. Apparatus according to claim 83, 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.

85. Apparatus according to claim 82, 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.

86. Apparatus according to claim 85, 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.

87. Apparatus according to claim 82, 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.

88. Apparatus according to claim 82, 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.



89. Apparatus according to claim 74, 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.

90. 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; and

means for satisfying requests received over said interchange bus from said host computer.

91. Apparatus according to claim 90, 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.

92. Apparatus according to claim 90, 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.

93. Apparatus according to claim 90, 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.



94. Apparatus according to claim 90, 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.

95. Apparatus according to claim 90, 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.

96. A network file server for use with a data network and a mass storage device, comprising:

means for decoding NFS requests from said network;

means for performing procedures for satisfying said NFS requests, including accessing said mass storage device if required; and

means for encoding any NFS reply messages for return transmission on said network,

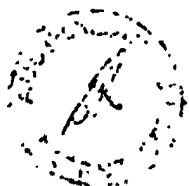
said network file server lacking means in said network file server for satisfying any non-NFS requests from said network.

Dated this 4th day of March, 1993

AUSPEX SYSTEMS, INC.

By its Patent Attorneys

DAVIES COLLISON CAVE



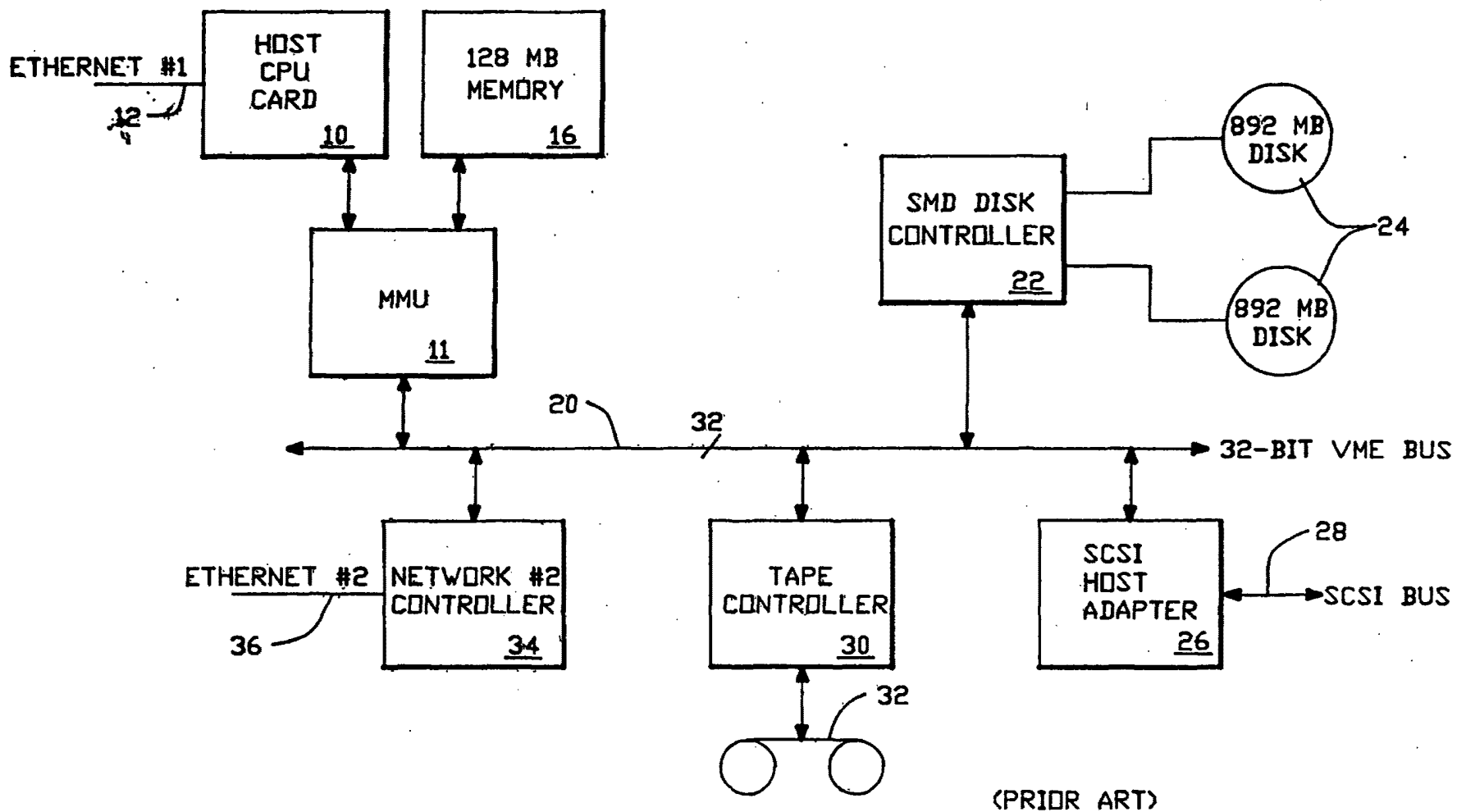


FIG.-1

(PRIOR ART)

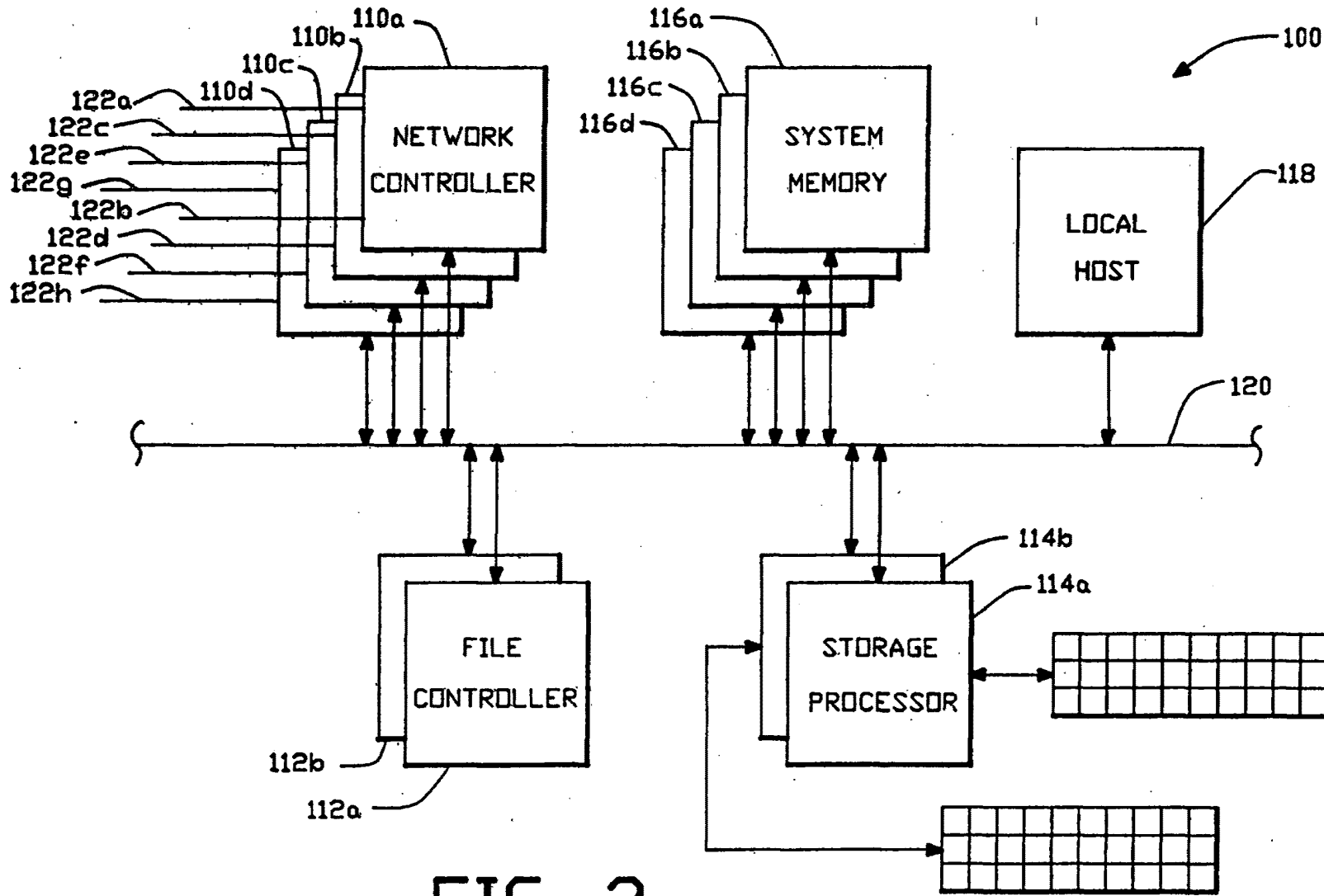


FIG.-2

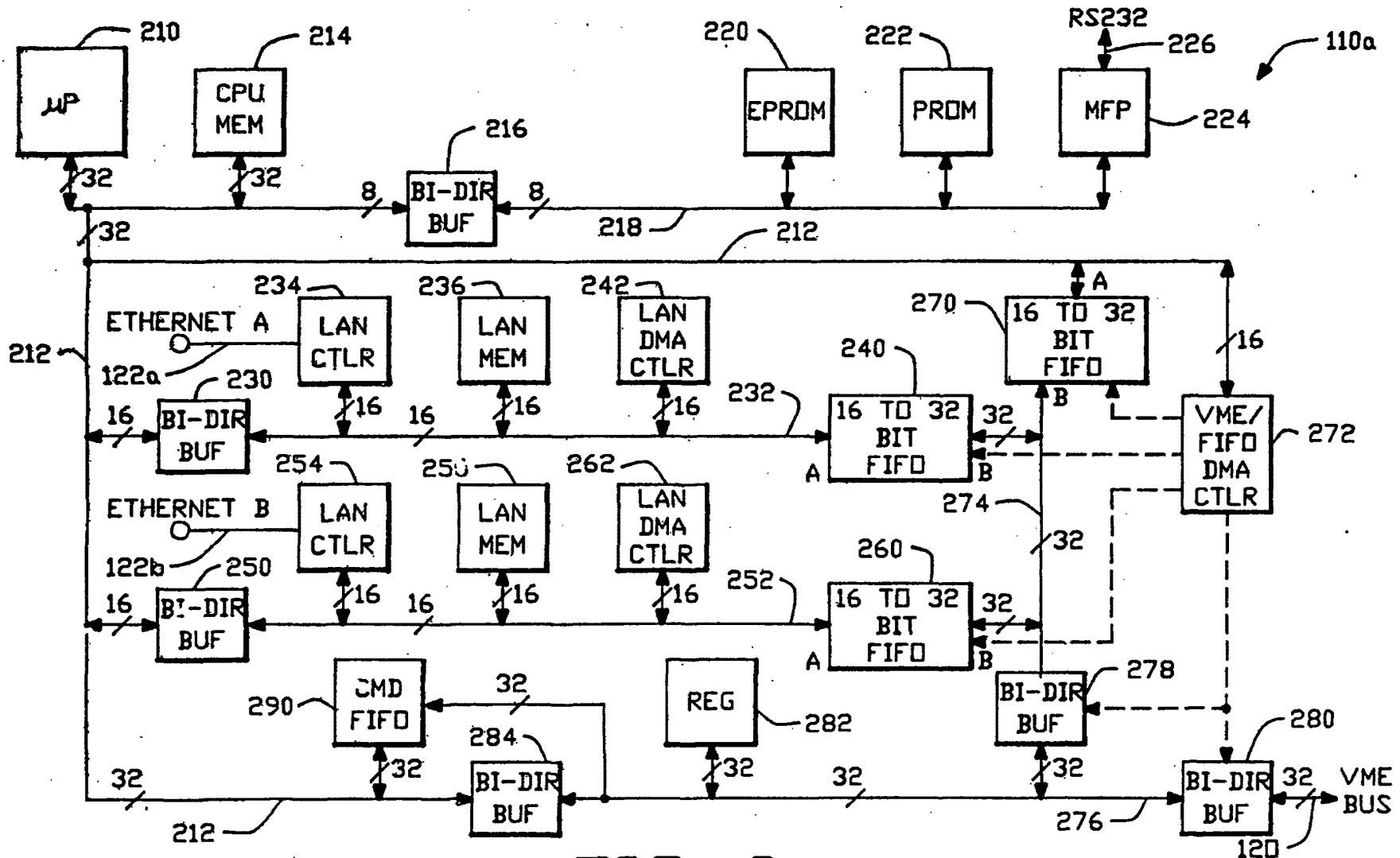


FIG.-3 (NETWORK CONTROLLER)

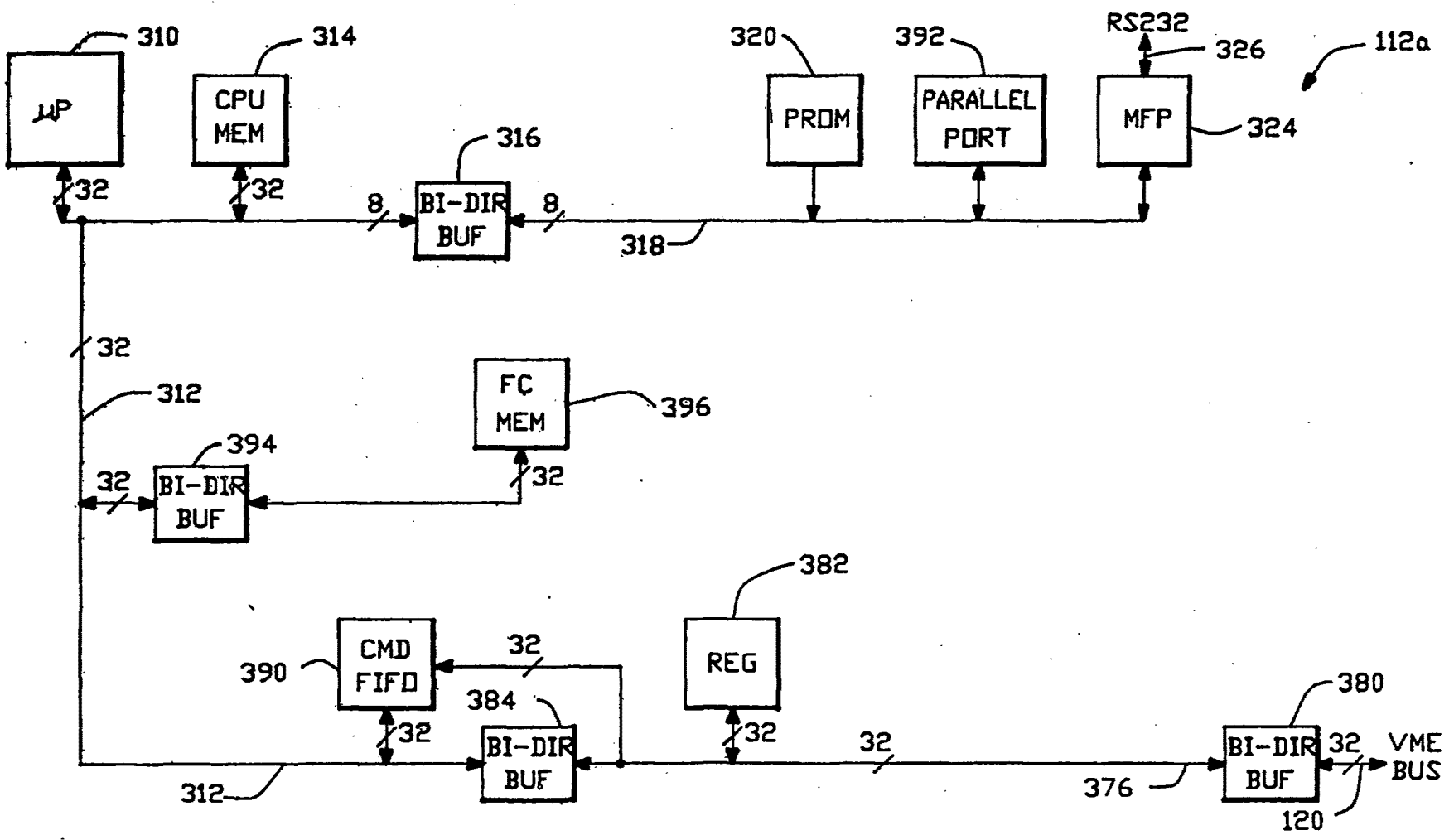


FIG.-4 (FILE CONTROLLER)



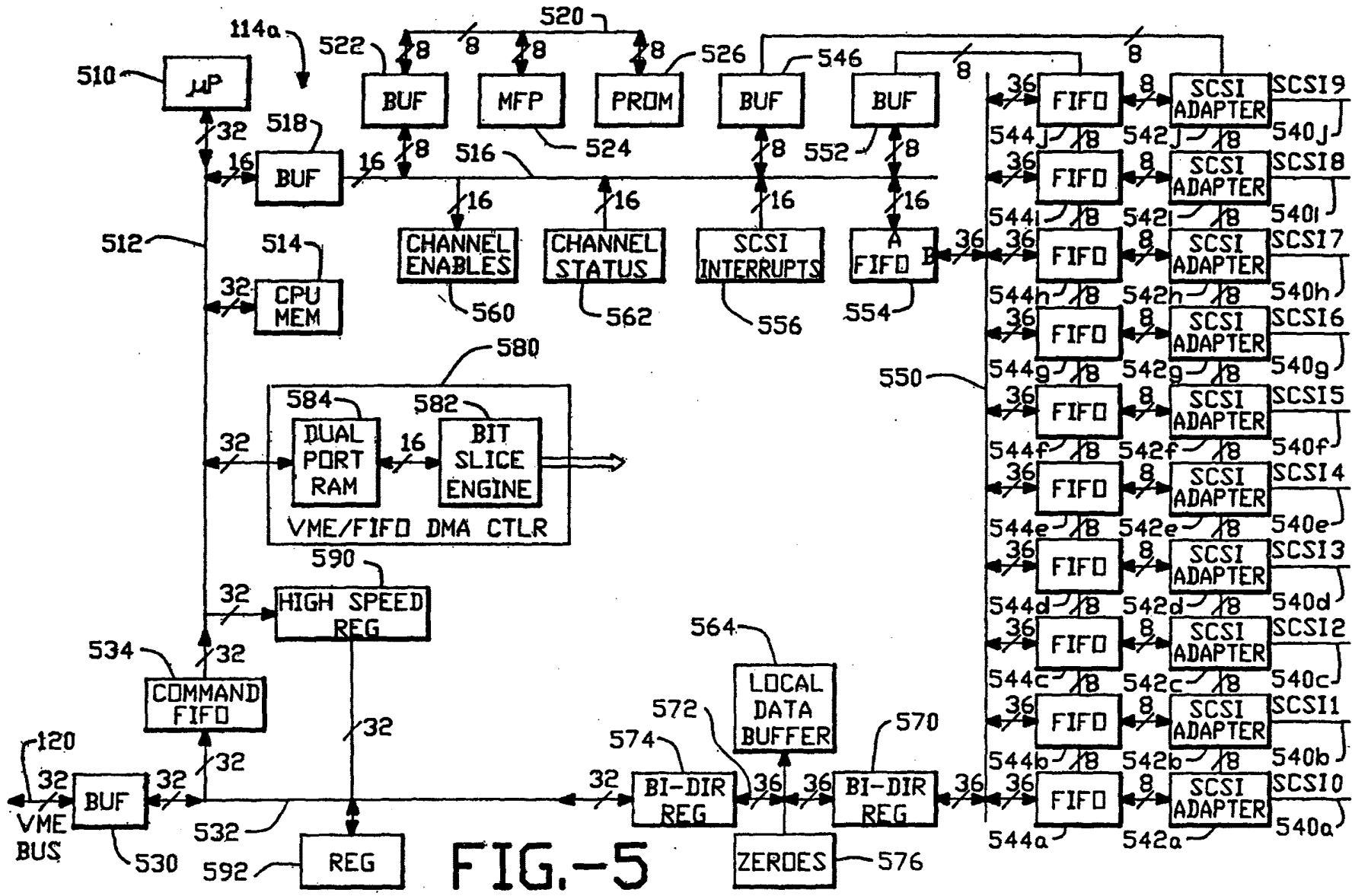
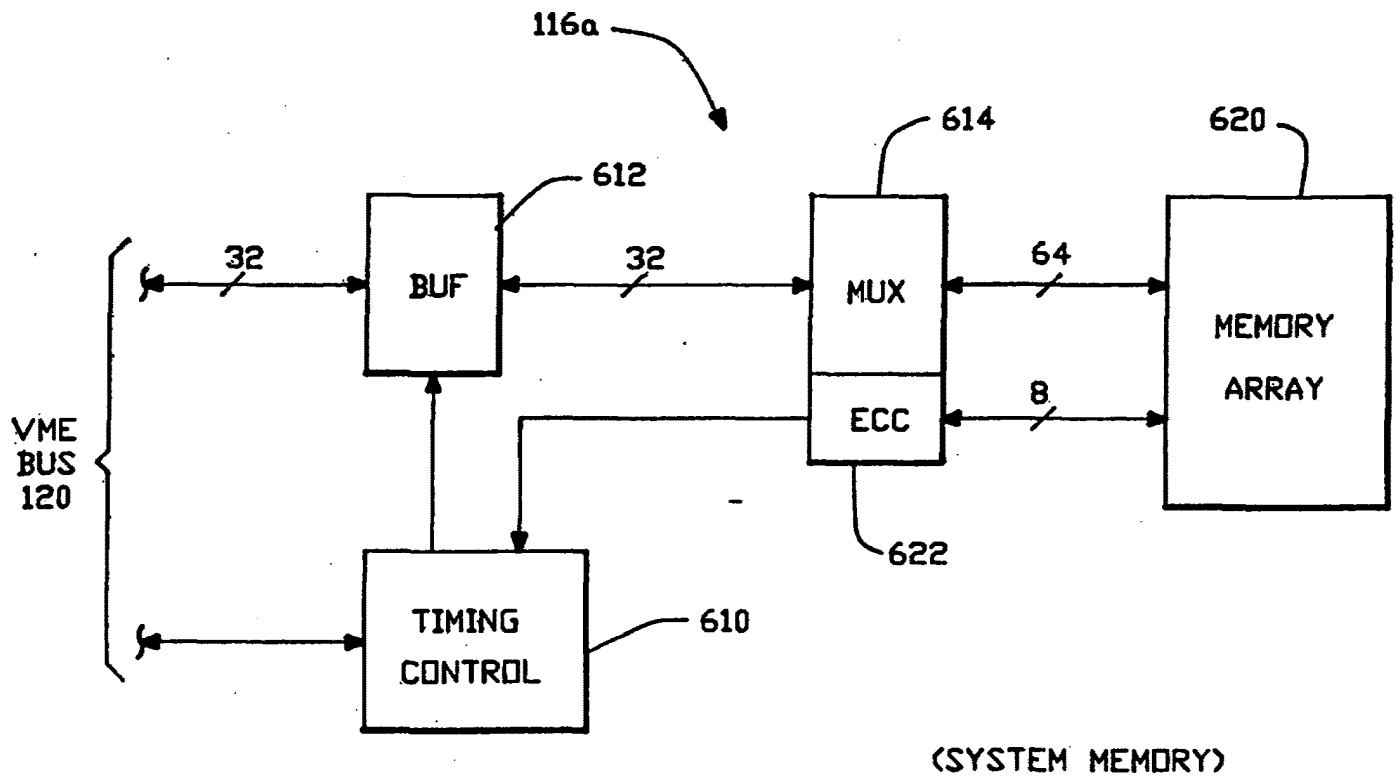


FIG.-5



6/12

FIG.-6

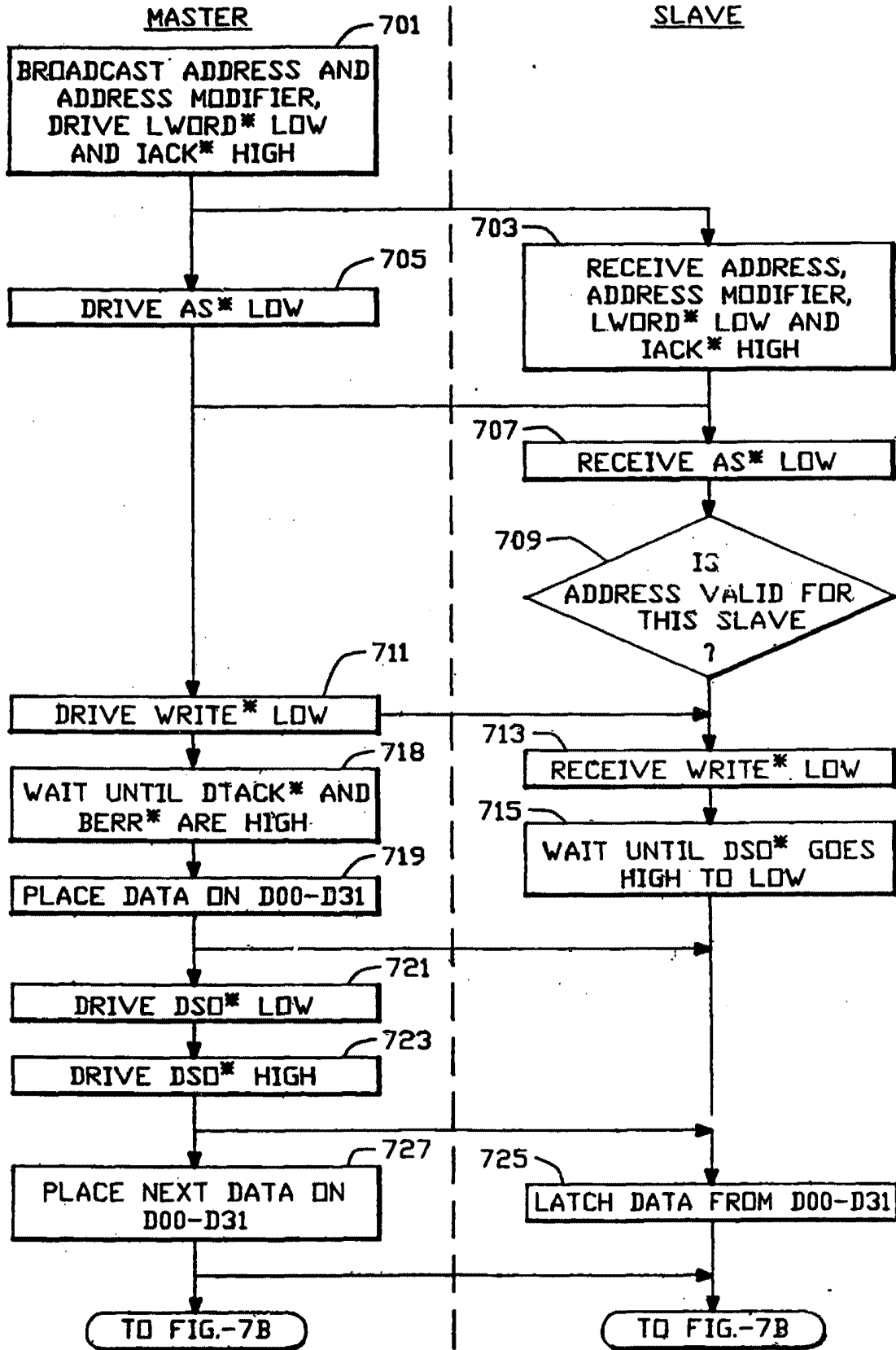


FIG.-7A

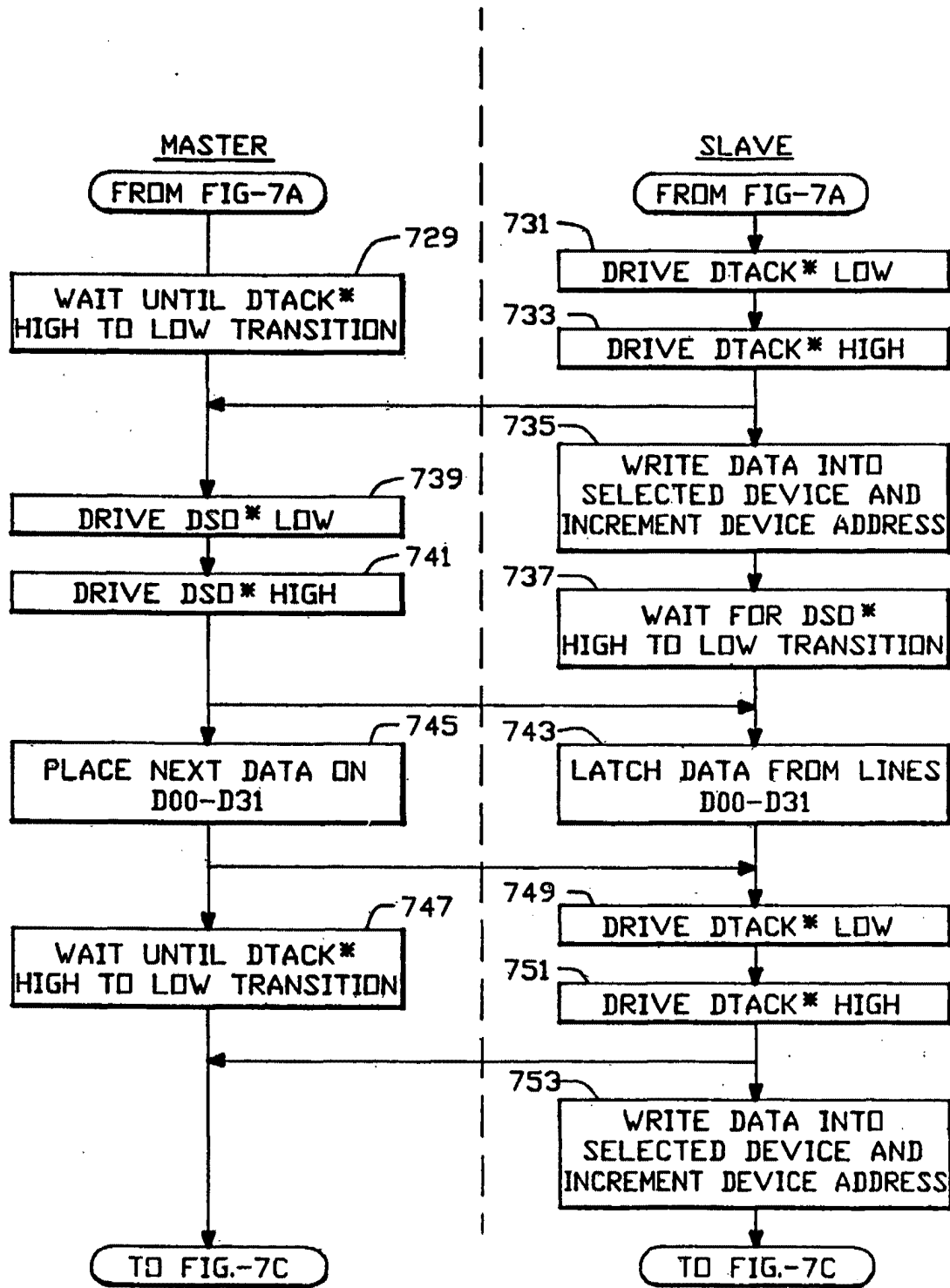


FIG.-7B

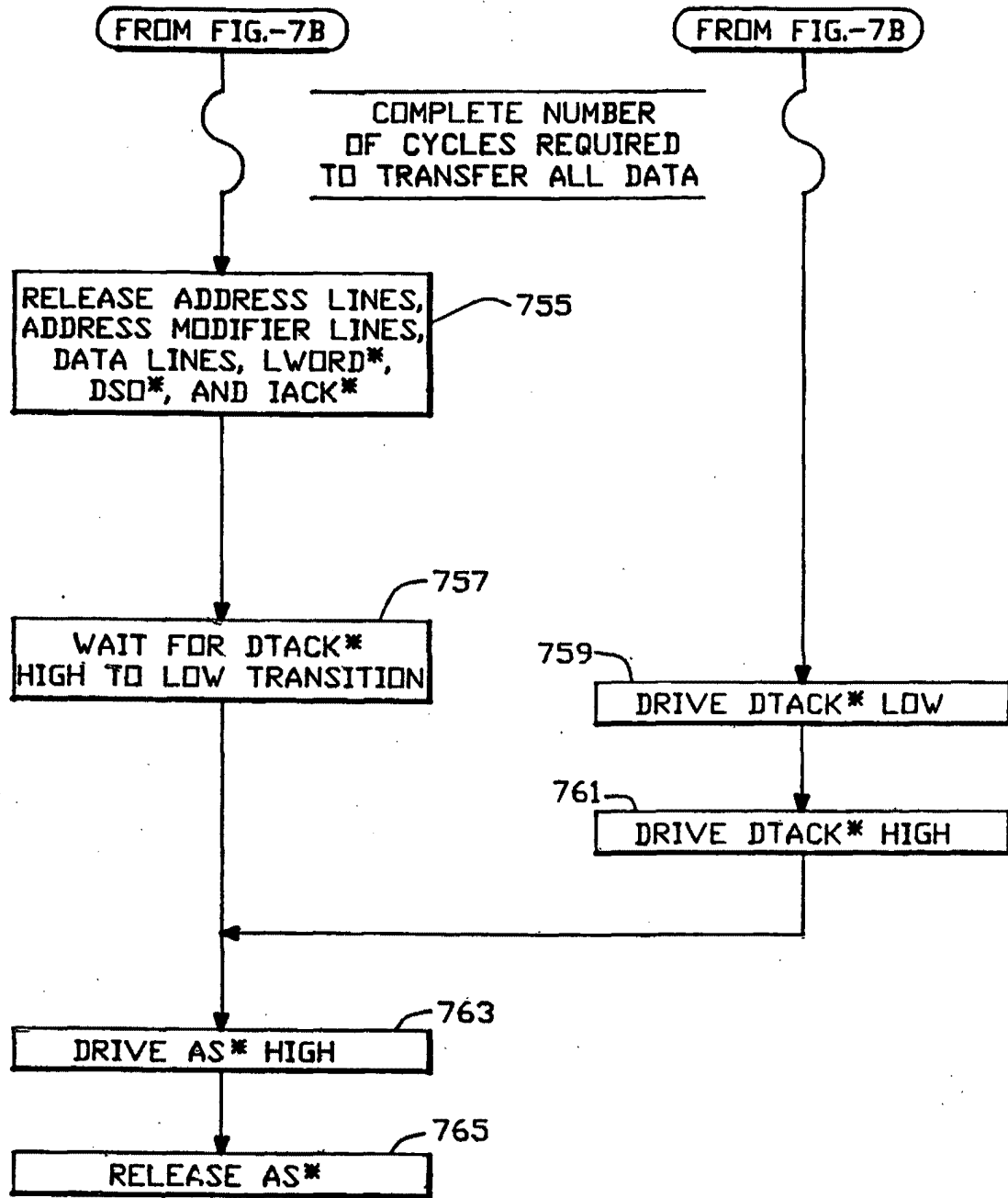


FIG.-7C

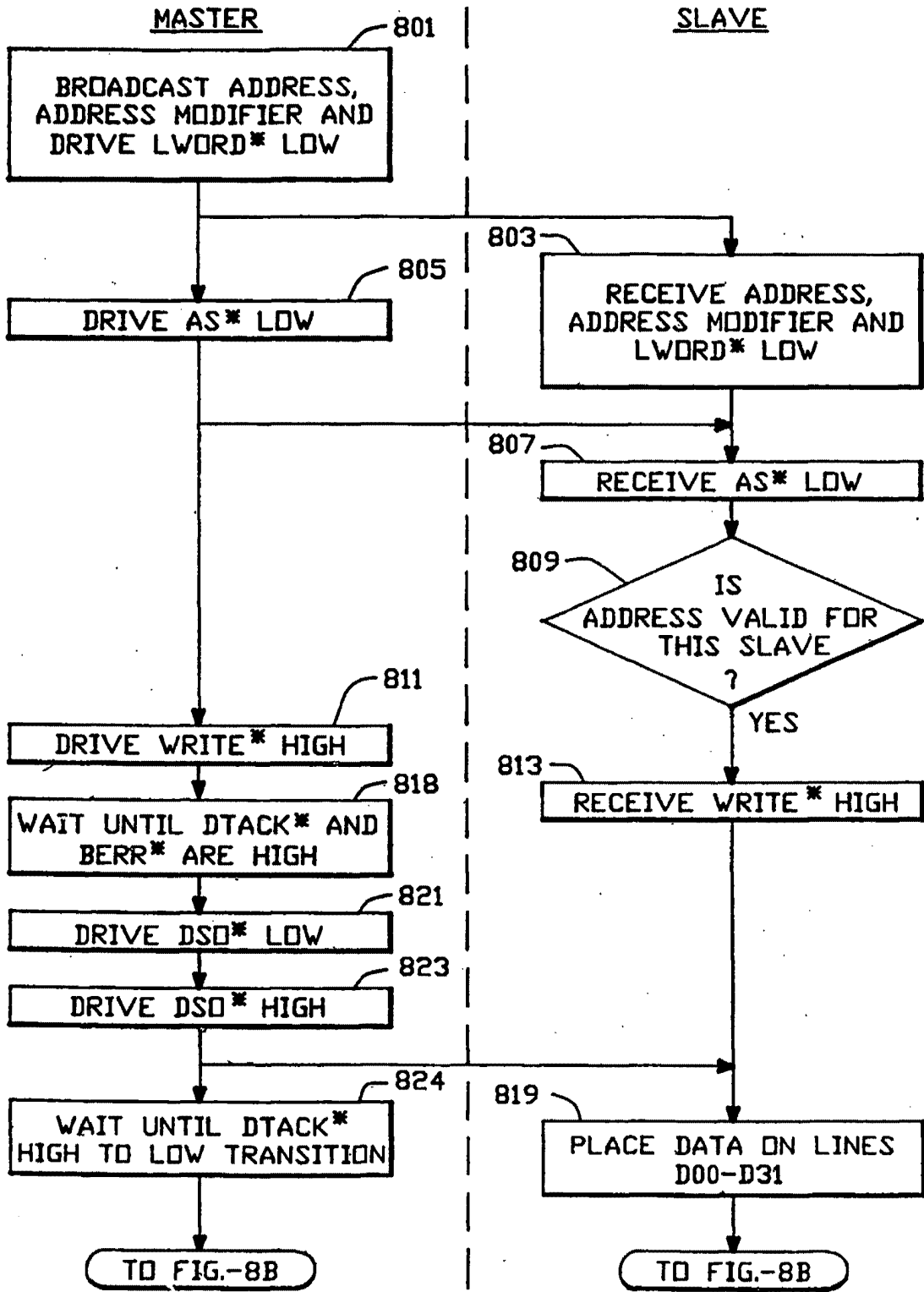


FIG.-8A

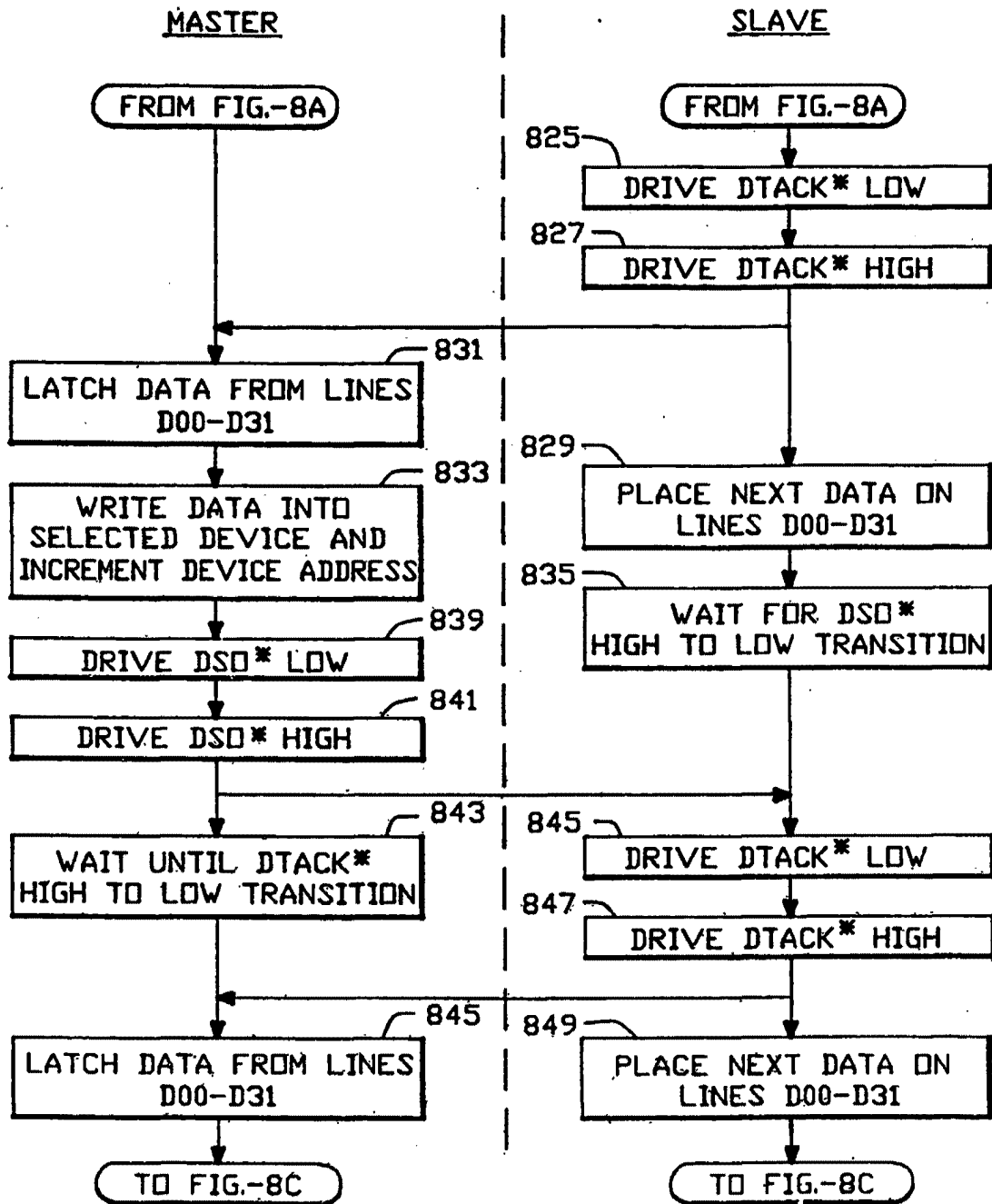


FIG.-8B

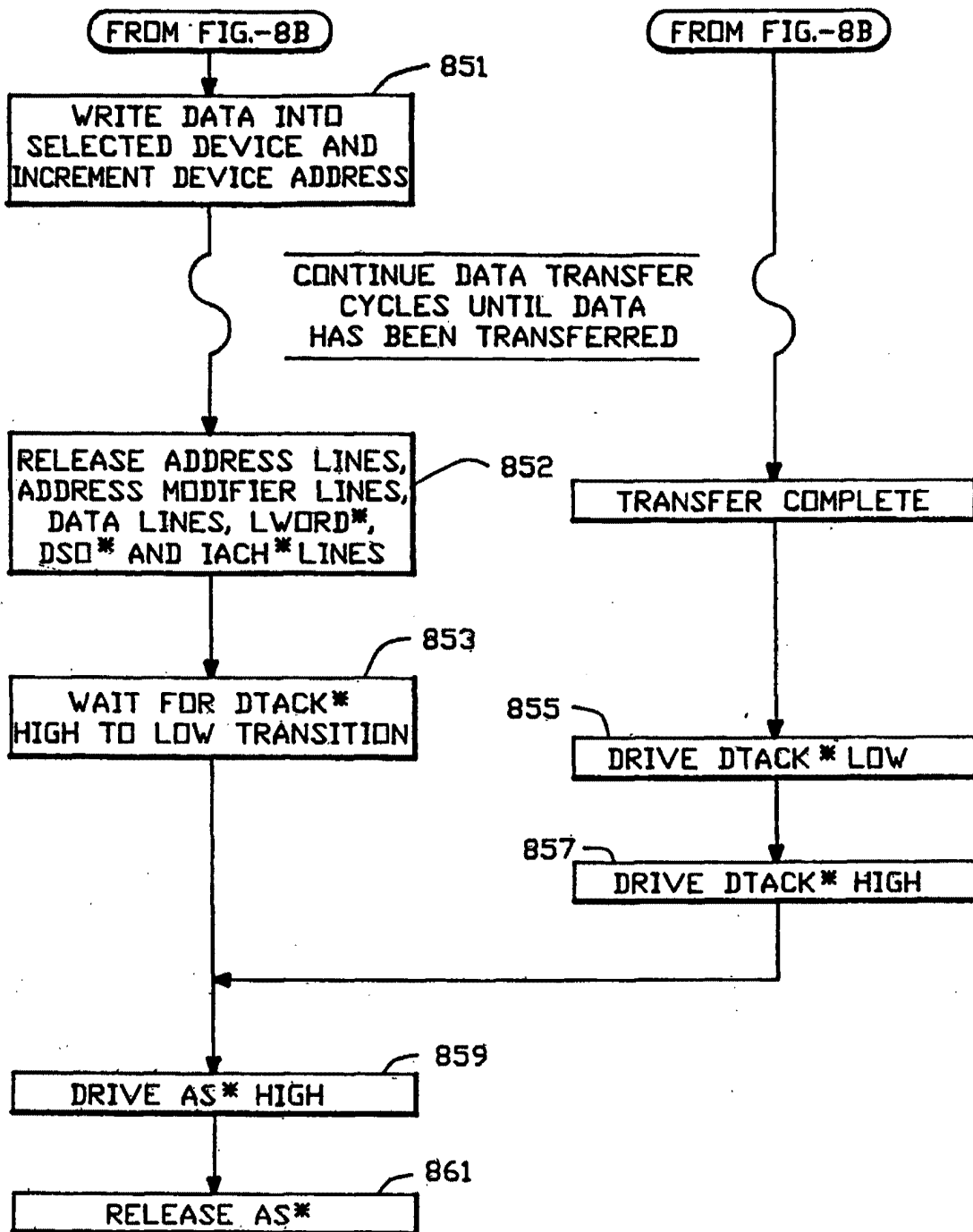


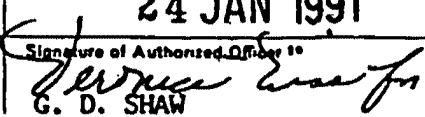
FIG.-8C



# INTERNATIONAL SEARCH REPORT

International Application No

PCT/US90/04711

<b>I. CLASSIFICATION OF SUBJECT MATTER</b> (If several classification symbols apply, indicate all) *		
According to International Patent Classification (IPC) or to both National Classification and IPC		
IPC (5) :	G06F 15/16	
U.S. Cl :	364/200	
<b>II. FIELDS SEARCHED</b>		
Minimum Documentation Searched *		
Classification System	Classification Symbols	
U.S.	364/200,900	
Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched *		
<b>III. DOCUMENTS CONSIDERED TO BE RELEVANT **</b>		
Category *	Citation of Document, <sup>16</sup> with indication, where appropriate, of the relevant passages <sup>17</sup>	Relevant to Claim No. <sup>18</sup>
Y P	US,A 4,897,781 (CHANG) 30 January 1990 See the entire document.	1-8
Y P	US,A 4,887,204 (JOHNSON) 12 December 1989 See the entire document.	1-8
Y	US,A 4,819,159 (SHIPLEY) 04 April 1989 See the entire document.	1-8
Y	US,A 4,710,868 (COCKE) 01 December 1987 See the entire document.	1-8
<p>* Special categories of cited documents: <sup>15</sup></p> <p>"A" document defining the general state of the art which is not considered to be of particular relevance</p> <p>"E" earlier document but published on or after the international filing date</p> <p>"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)</p> <p>"O" document referring to an oral disclosure, use, exhibition or other means</p> <p>"P" document published prior to the international filing date but later than the priority date claimed</p> <p>"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</p> <p>"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step</p> <p>"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.</p> <p>"&amp;" document member of the same patent family</p>		
<b>IV. CERTIFICATION</b>		
Date of the Actual Completion of the International Search <sup>1</sup>	Date of Mailing of this International Search Report <sup>2</sup>	
30 NOVEMBER 1990	24 JAN 1991	
International Searching Authority <sup>3</sup>	Signature of Authorized Officer <sup>19</sup>	
ISA/US	 G. D. SHAW	



AU9465905

*BPA* *03*

**(12) PATENT ABRIDGMENT (11) Document No. AU-B-65905/94**  
**(19) AUSTRALIAN PATENT OFFICE (10) Acceptance No. 670376**

- (54) Title  
PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE
- International Patent Classification(s)
- (51)<sup>s</sup> G06F 015/16
- (21) Application No. : 65905/94 (22) Application Date : 23.06.94
- (30) Priority Data
- (31) Number (32) Date (33) Country  
404959 08.09.89 US UNITED STATES OF AMERICA
- (43) Publication Date : 01.09.94
- (44) Publication Date of Accepted Application : 11.07.96
- (62) Related to Division(s) : 64125/90
- (71) Applicant(s)  
AUSPEX SYSTEMS, INC.
- (72) Inventor(s)  
EDWARD JOHN ROW; LAURENCE B BOUCHER; WILLIAM M PITTS; STEPHEN E BLIGHTMAN
- (74) Attorney or Agent  
DAVIES COLLISON CAVE , 1 Little Collins Street, MELBOURNE VIC 3000
- (56) Prior Art Documents  
US 4897781  
US 4887204  
US 4819159
- (57) Claim

1. A network file server for use with a data network and a mass storage device, comprising:

a host processor unit; and

an interface processor unit coupleable to said network, to said mass storage device and to said host processor unit, said interface processor unit including means for decoding all NFS requests from said network, means for performing all procedures for satisfying said NFS requests, means for encoding any NFS reply messages for return transmission on said network, and means for satisfying file system requests from said host processor unit, and

means for transmitting predefined non-NFS categories of messages from said network to said host processor unit for processing in said host processor unit.

2. A network file server for use with a data network and a mass storage device comprising;

a host processor unit running a UNIX operating system; and

an interface processor unit coupleable to said network, to said mass storage device and to said host processor unit, said interface processor unit including means for

(11) AU-B-65905/94  
(10) 670376

-2-

decoding all NFS requests from said network, means for performing all procedures for satisfying said NFS requests, means for encoding any NFS reply messages for return transmission on said network, and means for satisfying file system requests from said host processor unit.

3. Apparatus for use with a data network and a mass storage device, comprising the combination of first and second processing units,

said first processing unit being coupled to said network and performing procedures for satisfying requests from said network which are within a predefined non-NFS class of requests,

and said second processing unit being coupled to said network and to said mass storage device and decoding NFS requests from said network, performing procedures for satisfying said NFS requests, and encoding NFS reply messages for return transmission on said network, said second processing unit not satisfying any requests from said network which are within said predefined non-NFS class of requests.

A U S T R A L I A 670376  
Patents Act 1990

PATENT REQUEST : STANDARD PATENT

We being the persons identified below as the Applicant, request the grant of a patent to the person identified below as the Nominated Person, for an invention described in the accompanying standard complete specification.

Full application details follow:

[71/70] Applicant/Nominated Person:

Address: Auspex Systems, Inc.  
2952 Bunker Hill Lane  
Santa Clara, California, 95054  
UNITED STATES OF AMERICA

[54] Invention Title:

"Parallel I/O Network File Server Architecture"

[72] Name(s) of actual inventor(s):

Edward John Row  
Laurence B Boucher  
William M Pitts  
Stephen E Blightman

[74] Address for service in Australia:

DAVIES COLLISON CAVE, Patent Attorneys, of 1 Little Collins Street,  
Melbourne, Victoria, Australia. Attorney Code: DM

DIVISIONAL APPLICATION DETAILS:

[62] Original Application No. 64125/90

*Keith Hosie*

(a member of the firm of DAVIES COLLISON CAVE  
for and on behalf of the Applicant).

22 June, 1994

60871 21 23 94

AUSTRALIA

*Patents Act 1990*

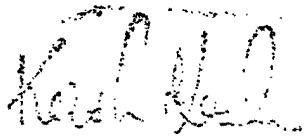
NOTICE OF ENTITLEMENT

We, Auspex Systems, Inc the applicant/Nominated Person in respect of Application No. 65905/94, state the following:-

The Nominated Person is entitled to the grant of the patent because the Nominated Person derives title to the invention from the inventors by assignment.

The person nominated for the grant of the patent is the applicant and Nominated Person of the original application No. 64125/90.

8 May, 1996



---

(A member of the firm of Davies & Collison for  
and on behalf of the applicant(s))

A U S T R A L I A  
Patents Act 1952  
COMPLETE SPECIFICATION  
FOR A STANDARD PATENT  
(ORIGINAL)

---

Name of Applicant:           Auspex Systems, Inc.

Address for Service:       DAVIES COLLISON CAVE, Patent Attorneys,  
1 Little Collins Street, Melbourne, 3000.

Invention Title:            "Parallel I/O Network File Server Architecture"

The following statement is a full description of this invention, including the best method of performing it known to me/us:

PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE

5

10       The present application is related to the following published International Patent Applications:

1. MULTIPLE FACILITY OPERATING SYSTEM ARCHITECTURE, invented by David Hitz, Allan Schwartz, James Lau and Guy Harris, PCT Publication No. 15 WO91/04540, international filing date April 4, 1991;

2. ENHANCED VMEBUS PROTOCOL UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA TRANSFER, invented by Daryl Starr, PCT Publication No. 20 WO91/03736, international filing date March 21, 1991; and

3. BUS LOCKING FIFO MULTI-PROCESSOR COMMUNICATIONS SYSTEM UTILIZING PSEUDOSYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA TRANSFER invented by Daryl D. Starr, William Pitts and Stephen Blightman, PCT Publication 25 No. WO91/11768, international filing date August 8, 1991.

The above applications are all assigned to the assignee of the present invention and are all expressly incorporated herein by reference.

BACKGROUND OF THE INVENTION

Field of the Invention

5 The invention relates to computer data networks, and more particularly, to network file server architectures for computer networks.

Description of the Related Art

10 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  
15 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.

20 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  
25 exchange service requests and responses with server nodes.

30 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  
35 (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.

5 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; 10 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, 15 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 20 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. 25 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 30 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 35 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.

5           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.

10           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.

15           Such low client/server ratios cause piecemeal 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:

- 5       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.
  - 10       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.
  - 15       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).
  - 20       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.

5 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  
10 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  
15 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.

20 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  
25 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  
30 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.

35 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  
5 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.

10

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, among other things each processor runs  
15 the entire operating system. Interprocessor communication is also not optimal for file server applications.

#### SUMMARY OF THE INVENTION

20 In accordance with the present invention there is provided a network file server for use with a data network and a mass storage device, comprising:

a host processor unit; and

an interface processor unit coupleable to said network, to said mass storage device and to said host processor unit, said interface processor unit including means for  
25 decoding all NFS requests from said network, means for performing all procedures for satisfying said NFS requests, means for encoding any NFS reply messages for return transmission on said network, and means for satisfying file system requests from said host processor unit, and

means for transmitting predefined non-NFS categories of messages from said  
30 network to said host processor unit for processing in said host processor unit.

In yet another aspect, there is provided a network file server for use with a data network and a mass storage device comprising:

a host processor unit running a UNIX operating system; and

5 an interface processor unit coupleable to said network, to said mass storage device and to said host processor unit, said interface processor unit including means for decoding all NFS requests from said network, means for performing all procedures for satisfying said NFS requests, means for encoding any NFS reply messages for return transmission on said network, and means for satisfying file system requests from said host processor unit.

10

The invention also provides a network file server for use with a data network and

a mass storage device, said network file server including a first unit comprising:  
means for decoding NFS requests from said network;  
means for performing procedures for satisfying said NFS requests, including  
accessing said mass storage device if required; and  
5 means for encoding any NFS reply messages for return transmission on said  
network,  
said first unit lacking means in said first unit for executing any programs which  
make UNIX operating system calls.

10 In accordance with the invention there is also provided a network file server for  
use with a data network and a mass storage device, said network file server including a  
first unit comprising:  
means for decoding NFS requests from said network;  
means for performing procedures for satisfying said NFS requests, including  
15 accessing said mass storage device if required; and  
means for encoding any NFS reply messages for return transmission on said  
network,  
said first unit lacking any UNIX kernel.

20 The invention further provides a network file server unit for use with a data  
network and a mass storage device, said network file server unit comprising:  
means for decoding NFS requests from said network;  
means for performing procedures for satisfying said NFS requests, including  
accessing said mass storage device if required; and  
25 means for encoding any NFS reply messages for return transmission on said  
network,  
said first unit lacking any UNIX application programs running on said first unit.

#### BRIEF DESCRIPTION OF THE DRAWINGS

30 The invention is described in greater detail hereinafter, by way of example only,  
with reference to the accompanying 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;

5 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;

10 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

15 Figs. 8A-C are a flowchart illustrating the operation of a fast transfer protocol BLOCK READ cycle.

#### DETAILED DESCRIPTION

20 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  
25 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  
30 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  
35 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 format 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.

5 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.

5  
10  
15  
20  
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.

25  
30  
35  
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  
5 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  
10 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 35, packets  
15 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  
20 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  
25 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  
30 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  
35 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.

5 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.

10 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

15 . 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  
20 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  
25 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  
30 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  
35 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.

5 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 10 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 15 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.

20 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 25 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: 30 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 35 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.

5 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.

10 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.

15 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.

20

25 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.

30 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

35 100; the external network protocol supported by the



server is precisely standard, licensed NFS. LNFS is described in more detail below.

5 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.

10 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.

15 An overview of the software in each of the processors will now be set forth.

#### Network Controller 110

20 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.

25

30

35

The server 100 uses conventional NFS ported from Sun Microsystems, Inc., Mountain View, CA, and is NFS protocol compatible.

5 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.

10  
15  
20

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.

25 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.

30

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  
5 110 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  
10 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  
15 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  
20 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  
25 involvement of the host 118.

#### Storage Processor

The intelligent storage processor 114 manages all disk and tape storage operations. While autonomous,  
30 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  
35 remote clients.

5 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,  
10 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  
15 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  
20 devices and system memory 116.

#### Host Processor 118

25 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  
30 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  
35 demons operate transparently.

The host processor 118 runs the following software modules:

5     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.

10     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.

15     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.

20     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.

30     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.

35     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  
5 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  
10 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  
15 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  
20 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  
25 232. 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  
30 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  
35 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.

5 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 message passing. Referring to the Appendix, the FIFOs 240 and 260 are programmed with  
10 the following bit settings in the Data Transfer Configuration Register:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
	0	WD Mode	N/A
	1	Parity Chip	N/A
15	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)
20	7	Reset	no (0)

The Data Transfer Control Register is programmed as follows:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
	0	Enable PortA Req/Ack	yes (1)
25	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)
30	6	Checksum Enable	yes (1)
	7	PortA Master	yes (1)

35 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.

5 The remainder of the control settings in FIFO 270 are the same as those in FIFOs 240 and 260 described above.

10 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.

25 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.

30 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 35 236 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 235. 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  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 118 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  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 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.

15 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.

30 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.

5       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  
10       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  
15       NFS: get attributes of a file (GETATTR); set attributes of a file (SETATTR); look up a file (LOOKUP); create a file (CREATE); remove a file (REMOVE); rename a file (RENAME); create a new linked file (LINK); create a symlink (SYMLINK); remove a  
20       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  
25       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.

30       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  
35       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  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 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.

5 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 10 128 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.

15 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 20 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 25 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 30 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 35 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.

5        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  
10        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  
15        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  
20        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  
25        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  
30        116.

The file controller embodiment shown in Fig. 4 does  
35        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.

5 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.

5 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  
10 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.

15 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.

20 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-  
25 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.

30 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  
35 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.

5       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.

10       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  
15:       during the period between the initial request and the release request. This prevents corruption of the data by other clients.

20       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  
25       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  
30       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

35       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.

5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 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
10	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.
15	5 input	Channel Operation Done - A composite of the channel done bits from all 13 channels of the DMA controller, described below.
20	4 output	DMA Controller Enable. Enables the DMA Controller to run.
25	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.
30	1 output	External Interrupts Disable. Disables externally generated interrupts to the microprocessor 510.
35	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 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.

5 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  
10 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.

15 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  
20 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  
25 516 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  
30 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.

35 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:

<u>Bit</u>	<u>Definition</u>	<u>Setting</u>	
	0	WD Mode	(0)
5	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)
10	6	Checksum Carry Wrap	(0)
	7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

<u>Bit</u>	<u>Definition</u>	<u>Setting</u>	
15	0	Enable PortA Req/Ack	(1)
	1	Enable PortB Req/Ack	(1)
	2	Data Transfer Direction	as desired
	3	CPU parity enable	(0)
20	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:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
	0	WD Mode	(0)
5	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)
10	6	Checksum Carry Wrap	(0)
	7	Reset	(0)

The Data Transfer Control Register is programmed as follows:

	<u>Bit</u>	<u>Definition</u>	<u>Setting</u>
15	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)
20	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.

25 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.

30 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  
35 the control bus 515. 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

- 15 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.
- 20
- 25 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.
- 30 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.
- 35
- 40 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.
- 45 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.

5 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.

10 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.

20 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.

25 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

30

35

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.

5     STORAGE PROCESSOR OPERATION

10     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.

20     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 and 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

5 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  
10 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.

15 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  
20 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  
25 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  
30 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  
35 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

5           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  
10           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.  
15

          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  
20           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  
25           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  
30           580 are as follows:

OP CODE OPERATION

	<u>OP CODE</u>	<u>OPERATION</u>	
	0	NO-OP	
5	1	ZEROES -> BUFFER	Move zeros from zeros register 576 to local data buffer 564.
10	2	ZEROES -> FIFO	Move zeros from zeros register 576 to the currently selected FIFO on common data bus 550.
15	3	ZEROES -> VMEbus	Move zeros from zeros register 576 out onto the VME bus 120. Used for initializing cache buffers in system memory 116.
20			
	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.
25			
30			
	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.
35			
40			
45			
50	6	VMEbus -> BUFFER & FIFO	Target data is moved from VME bus 120 to a SCSI



5			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.
10	7	BUFFER -> VMEbus	This operation is not used.
15	8	BUFFER -> FIFO	Participating data is transferred to create redundant data or recovered data on a disk drive. Used only in RAID 5 applications.
20	9	FIFO -> VMEbus	This operation is used to move target data directly from a disk drive onto the VME bus 120.
25	A	FIFO -> BUFFER	Used to move participating data for recovery and modify operations. Used only in RAID 5 applications.
30	B	FIFO -> VMEbus & BUFFER	This operation is used to save target data for participation in data recovery. Used only in RAID 5 applications.
35			

SYSTEM MEMORY

40 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

45

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.

15 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.

30 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:

	<u>Type</u>	<u>Description</u>
	0	Pointer to a new message being sent
	1	Pointer to a reply message
	2	Pointer to message to be forwarded
5	3	Pointer to message to be freed; also message acknowledgment

All messages are 128-bytes long.

10 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 15 214 for this purpose over the VME bus 120.

20 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 25 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. 30

35 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

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  
5 instruction fetches and transmissions of message and file data.

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.

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 summary, the embodiments of the present invention involve a new, server-specific I/O architecture that is optimised for a Unix file server's most common actions -- file operations. Roughly stated, a file server architecture is provided which comprises 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 optimised 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

APPENDIX A

VME/FIFO DMA Controller

In storage processor 114a, DMA controller 580  
5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 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.

5  
10  
15  
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.

20  
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.

25  
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.

30  
CIN 38 (Carry INput) This bit forces the carry input to the alu.

35  
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.

40  
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

20 0 This value selects a null flag.

25 1 When set this bit enables the buffer clock. When reset this bit disables the buffer clock.

30 2 When this bit is cleared VME bus transfers, buffer operations and RAS are all disabled.

35 3 NOT USED

40 4 When set this bit enables VME bus transfers.

45 5 When set this bit enables buffer operations.

50 6 When set this bit asserts the row address strobe to the dram buffer.

55 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
- 5 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
- 15 1 SGL\_CLK  
generates a single transition of buffer clock.
- 20 2 SET\_VB  
forces vme and buffer enable to be set.
- 25 3 CL\_PERR  
clears buffer parity error status.
- 30 4 SET\_DN  
set channel done status for the currently selected channel.
- 35 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
- 45 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.

5	3	D<14:7> -> mux addr<8:0> WR_VADL loads the data from the alu bus into the least significant 2 bytes of the VME address register.
10		D<15:2> -> VME addr<15:2> D1 -> ENB_tional registers D<15:2> -> VME addr<15:2> D1 -> ENB_ENH D0 -> ENB_BLK
15	4	WR_VADH loads the most significant 2 bytes of the VME address register.
20		D<15:0> -> VME addr<31:16>
25	5	WR_RADD loads the dual ported ram address counters.
		D<10:0> -> ram addr <10:0>
30	6	WR_WCNT loads the word counters. D15 -> count enable* D<14:8> -> count <6:0>
35	7	WR_CO loads the co-channel select register. D<7:4> -> CO<3:0>
40	8	WR_NXT loads the next-channel select register. D<3:0> -> NEXT<3:0>
45	9	WR_CUR loads the current-channel select register. D<3:0> -> CURR <3:0>
10:14		RESERVED - NOT DEFINED
50	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.

5

TEST<3:0> Condition

10

0	FALSE	-always false
1	TRUE	-always true
2	ALU_COUT	-carry output of alu
3	ALU_EQ	-equals output of alu

15

4	ALU_OVR	-alu overflow
5	ALU_NEG	-alu negative

20

6	XFR_DONE	-transfer complete
7	PAR_ERR	-buffer parity error
8	TIMOUT	-bus operation timeout

25

9	ANY_ERR	-any error status
---	---------	-------------------

14:10 RESERVED -NOT DEFINED

30

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.

35

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.

40

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.

45

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.

5 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
10	0	CURR POINTER 0	STATUS 0
	4	INITIAL POINTER 0	
15			
	58	CURR POINTER B	STATUS B
	5C	INITIAL POINTER B	
20	60	not used	not used
	64	not used	not used
25	68	CURR POINTER D	STATUS D
	6C	INITIAL POINTER D	
	70	not used	STATUS E
30	74	HIGH SPEED BUS ADDRESS 31:2;0;0;	
	78	PARAMETER BLOCK 0	
35			
	??	PARAMETER BLOCK n	

40 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  
45 initialized to 0x0000 by the microprocessor 510 before enabling the channel. Upon detecting a value of 0x0000

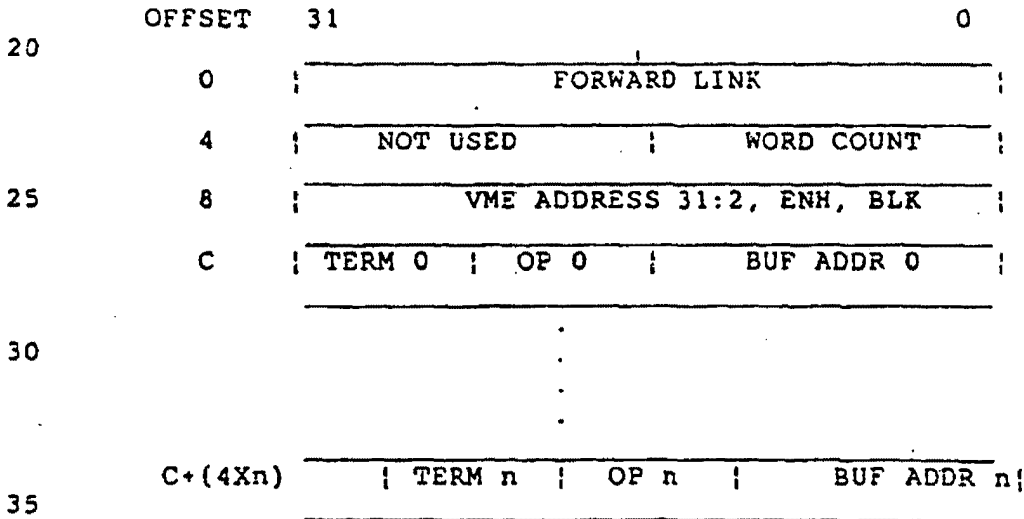
in the current block pointer the DMA controller 580 will copy the lower 16 bits from the initial pointer to the current pointer. Once the DMA controller 580 has completed the specified operations for the parameter block the current pointer will be updated to point to the next block. In the event that 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
- 15 2 BUS OPERATION TIMEOUT
- 3 BUS OPERATION ERROR
- 4 DATA PATH PARITY ERROR

The format of the parameter block is:



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.

5

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:

10

|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.

15

20

VME ADDRESS - The VME address specifies the starting address for data transfers. Thirty bits allows the address to start at any quad byte boundary.

25

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.

30

35

ELK - 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.

5        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.

10        |A19|A18|A17|A16|A15|A14|A13|A12|A11|A10|A9|A8|A7|A6|A5|A4|

15        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.

20        |0|0|0|0|0|0|0|0|0|0|T|

25        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):

	<u>OP CODE</u>	<u>OPERATION</u>
	0	NO-OP
	1	ZEROES -> BUFFER
	2	ZEROES -> FIFO
5	3	ZEROES -> VMEbus
	4	VMEbus -> BUFFER
	5	VMEbus -> FIFO
	6	VMEbus -> BUFFER & FIFO
	7	BUFFER -> VMEbus
10	8	BUFFER -> FIFO
	9	FIFO -> VMEbus
	A	FIFO -> BUFFER
	B	FIFO -> VMEbus & BUFFER
	C	RESERVED
15	D	RESERVED
	E	RESERVED
	F	RESERVED

APPENDIX B

Enhanced VME Block Transfer Protocol

5 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  
10 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  
15 signal line. The slave then responds by transmitting  
data transfer acknowledge signal of a given duration  
on the data transfer acknowledge signal line.

20 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.

25 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  
30 asserts a data transfer acknowledge signal and the  
master, in response to the data transfer acknowledge  
signal, captures the data broadcast by the slave.

35 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.

5       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.

10       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.

15       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.

20       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

25

30

35

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.

5 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  
10 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  
15 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  
20 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  
25 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.

30 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  
35 707. 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.

5

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.

10

15

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.

20

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.

25

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.

30

35

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 DSC\* 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 757.

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.

5       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.

10       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.

15       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.

20       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.

5       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.

10       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.

15       The slave drives DTACK\* low, 845, and, after a predetermined period of time, drives DTACK\* high, 847.

20       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.

25       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

35

receipt of the DTACK\* high to low transition, the master will drive AS\* high 859 and release the AS\* line 861.

5 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  
10 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  
15 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

5 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  
10 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  
15 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  
20 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  
25 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  
30 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  
35 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.

5 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.

10 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

15 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.

20 Clk

The clock input is used to generate some of the chip timing. It is expected to be in the 10-20 Mhz range.

25 Read En, Write En

30 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

5 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, Port B 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

10 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

15 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:

- 20 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.

25 Reset

This signal resets all the registers within the chip and causes all bidirectional pins to be in a high impedance state.

30 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.

5  
10  
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.

15  
20  
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.

25  
30  
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.

35  
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 Reg 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.

5 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 10 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. 15 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 20 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

##### 25 Data Transfer Configuration Register (Read/Write)

Register Address 0. This register is cleared by the reset signal.

30 Bit 0 WD Mode. Set if data transfers are to use the Western Digital WD33C93A protocol, otherwise the Adaptec 5250 protocol will be used.

Bit 1 Parity Chip. Set if this chip is to accumulate Port B parities.

35 Bit 2 Parity Correct Mode. Set if the parity chip is to correct parallel parity on Port B.

5 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.

10 Bit 4 Invert Port A byte address 0. Set to invert the least significant bit of Port A byte address.

15 Bit 5 Invert Port A byte address 1. Set to invert the most significant bit of Port A byte address.

20 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.

25 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.

30 Data Transfer Control Register (Read/Write)

Register Address 1. This register is cleared by the reset signal or by writing to the reset bit.

35 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.

40 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.

45

5 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.

10  
15 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.

20  
25 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.

30 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.

35  
40 Bit 7 Port A Master. Set if Port A is to operate in the master mode on Port A during the data transfer.

45 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.

- 5 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.
- 10 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.
- 15 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.
- 20
- 25 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)

30 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.

- 35 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.
- 40 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.
- 45 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 .

5 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.

15 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

25 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

35 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  
5 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  
10 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)

15 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  
20 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  
25 to read it.

PROGRAMMING THE FIFO CHIP

In general the fifo chip is programmed by writing  
30 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  
35 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.

5

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.

10

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.

15

```
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;
};
```

20

```
#define FIFO1 ((struct FIFO_regs*) FIFO_BASE_ADDRESS)
```

30

```
#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
```

35

40

```
#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.
```

```
5      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  
10      automatically using both the Port A and Port B  
                  handshakes, both data transfer enables bits should be  
                  set in this parameter.
```

```
*****/
```

```
FIFO_initiate_data_transfer(config_data, control_data)  
15  unsigned char config_data, control_data;  
    {  
      FIFO1->config = config_data | FIFO_RESET; /* Set  
20      Configuration value & Reset */  
      FIFO1->control = control_data & (~FIFO_PORT_ENABLES); /* Set  
      everything but enables */  
      FIFO1->control = control_data; /* Set data transfer  
      enables */  
    }
```

```
25  /*  
      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.
```

```
30  *****/
```

```
FIFO_force_odd_length_transfer()  
    {  
      FIFO1->control &= ~FIFO_PORT_ENABLES; /* Disable Ports A & B  
35      */  
      if (PORT_A_TO_PORT_B(FIFO1)) {  
          if (PORT_A_TO_RAM_Y(FIFO1)) {  
              FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL; /*  
40      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 */  
45      }  
      else {  
          if (PORT_B_TO_RAM_Y(FIFO1)) {  
              FIFO1->ram_Y_addr = FIFO_FORCE_RAM_FULL; /*  
50      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 */  
    }  
}
```

```
5  /*****  
    The following routine returns how many odd bytes have been  
left in the Fifo at the end of a data transfer.  
*****/
```

```
10 int FIFO_count_odd_bytes()  
   {  
       int number_odd_bytes;  
       number_odd_bytes=0;  
       if (FIFO1->status & FIFO_DATA_IN_RAM) {  
15         if (PORT_A_TO_PORT_B(FIFO1)) {  
             number_odd_bytes =  
(PORT_A_BYTE_ADDRESS(FIFO1));  
             if (PORT_A_TO_RAM_Y(FIFO1))  
20             number_odd_bytes += (FIFO1->ram_Y_addr) *  
4;  
             else number_odd_bytes += (FIFO1->ram_X_addr) * 4;  
         }  
         else {  
25             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);  
30   }
```

```
   /*****  
   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  
35 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.

```
40     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  
45     Bit 5 = ram Y address register failed  
     Bit 6 = data register failed  
     Bit 7 = checksum register failed  
*****/
```

```
50 #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;
5   int i;
    char j,error;
    FIFO1->config = FIFO_RESET;          /* reset the chip */
    error=0;
    register_addr = (char *) FIFO1;
10   j=1;

    /* first test registers 0 thru 5 */

    for (i=0; i<6; i++) {
15       *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 */
20       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;
25     }

    /* now test Ram data & checksum registers
       test 1s throughout Ram & then test 0s */

30     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
35         */
            FIFO1->data = test_data;
        FIFO1->control = 0;
        for (i=0; i<RAM_DEPTH; i++)
            if (FIFO1->data != test_data) error |= j; /* read &
40         check data */
        if (FIFO1->checksum) error |= 0x80; /* checksum
        should = 0 */
    }

45     /* 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 |
50     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;
5   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
10   address pattern */
        test_data += 0x04040404;
    }
    if (FIFO1->checksum != 0x0102) error |= 0x80; /* test checksum of
address pattern */
15   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
*/
20   if (FIFO1->status) error |= 0x04; /* status should be 0 */
    return (error);
}
```

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A network file server for use with a data network and a mass storage device, comprising:
  - 5 a host processor unit; and  
an interface processor unit coupleable to said network, to said mass storage device and to said host processor unit, said interface processor unit including means for decoding all NFS requests from said network, means for performing all procedures for satisfying said NFS requests, means for encoding any NFS reply messages for return  
10 transmission on said network, and means for satisfying file system requests from said host processor unit, and  
means for transmitting predefined non-NFS categories of messages from said network to said host processor unit for processing in said host processor unit.
- 15 2. A network file server for use with a data network and a mass storage device comprising;
  - a host processor unit running a UNIX operating system; and  
an interface processor unit coupleable to said network, to said mass storage device  
20 and to said host processor unit, said interface processor unit including means for decoding all NFS requests from said network, means for performing all procedures for satisfying said NFS requests, means for encoding any NFS reply messages for return  
transmission on said network, and means for satisfying file system requests from said  
host processor unit.
- 25 3. Apparatus for use with a data network and a mass storage device, comprising the combination of first and second processing units,
  - said first processing unit being coupled to said network and performing  
procedures for satisfying requests from said network which are within a predefined non-  
NFS class of requests,  
30 and said second processing unit being coupled to said network and to said mass  
storage device and decoding NFS requests from said network, performing procedures for

satisfying said NFS requests, and encoding NFS reply messages for return transmission on said network, said second processing unit not satisfying any requests from said network which are within said predefined non-NFS class of requests.

5 4. Apparatus according to claim 3, wherein said predefined non-NFS class of requests includes a predefined set of remote procedure calls.

5. Apparatus according to claim 3, wherein said first processing unit includes a general purpose operating system and wherein said second processing unit does not  
10 include a general purpose operating system.

6. Apparatus according to claim 3, wherein said second processing unit comprises:  
a network control unit coupleable to said network;

a data control unit coupleable to said mass storage device;

15 a buffer memory;

means in said network control unit for decoding said NFS requests and for encoding said NFS reply messages;

means for transmitting to said data control unit requests responsive to NFS requests from said network to store specified data from said network on said mass storage  
20 device;

means for transmitting said specified storage data from said network to said buffer memory and from said buffer memory to said data control unit;

means for transmitting to said data control unit requests responsive to NFS requests from said network to retrieve specified retrieval data from said mass storage  
25 device to said network;

means for transmitting said specified retrieval data from said data control unit to said buffer memory and from said buffer memory to said network.

7. A network file server for use with a data network and a mass storage device, said  
30 network file server comprising a first unit and a second unit, said first unit including:

means for decoding NFS requests from said network;

means for performing procedures for satisfying said NFS requests, including accessing said mass storage device if required; and

means for encoding any NFS reply messages for return transmission on said network, only said second unit running a general purpose operating system.

5

8. A network file server according to claim 7, wherein said second unit runs a UNIX operating system kernel.

9. A network file server for use with a data network and a mass storage device, said  
10 network file server comprising:

a network control module, including a network interface coupled to receive NFS requests from said network; and

a file system control module, including a mass storage device interface coupled to said mass storage device,

15 a communication path coupled directly between said network control module and said file system control module, said communication path carrying file retrieval requests prepared by said network control module in response to received NFS requests to retrieve specified retrieval data from said mass storage device,

20 said file system control module retrieving said specified retrieval data from said mass storage device in response to said file 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.

25

10. A network file server according to claim 9, wherein said file system control module returns said specified retrieval data directly to said network control module.

11. A network file server according to claim 9, wherein said network control module  
30 further prepares file storage requests in response to received NFS requests to store specified storage data on said mass storage device, said network control module

communicating said file storage requests to said file system control module.

and wherein said file system control module further stores said specified storage data on said mass storage device in response to said file storage requests.

5 12. A network file server according to claim 11, wherein said file storage requests are communicated to said file system control module via said communication path.

13. A network file server substantially as hereinbefore described with reference to the accompanying drawings.

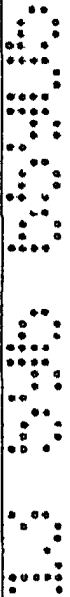
10

Dated this 7th day of May, 1996

AUSPEX SYSTEMS, INC.

By its Patent Attorneys

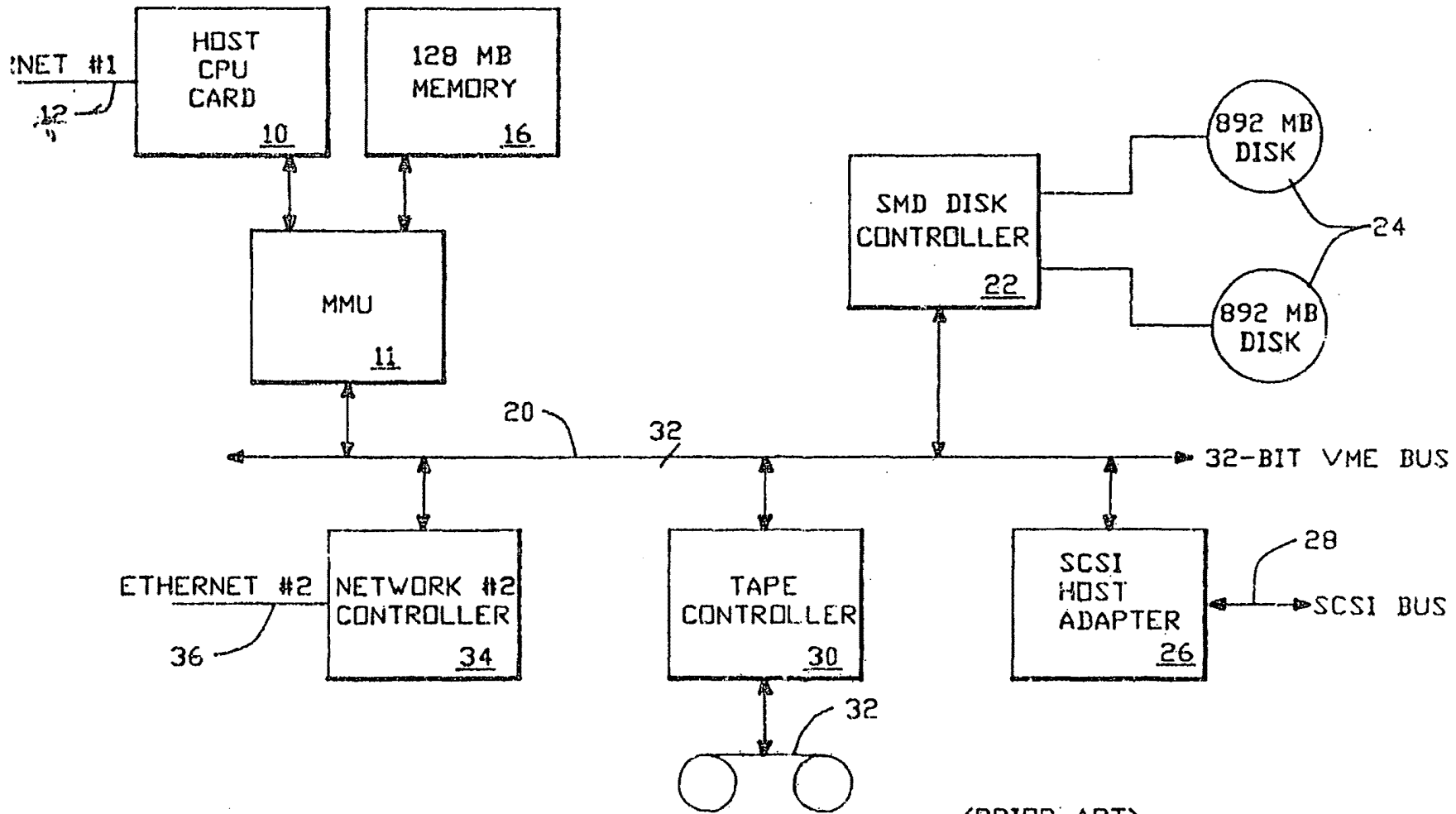
15 DAVIES COLLISON CAVE





## ABSTRACT

A file server architecture is disclosed, comprising as separate processors, a network controller unit, a file controller unit and a storage processor unit. These units  
5 incorporate their own processors, and operate in parallel with a local Unix host processor. All networks are connected to the network controller unit, which performs all protocol processing up through the NFS layer. The virtual file system is implemented in the file control unit, and the storage processor provides high-speed multiplexed access to an array of mass storage devices. The file controller unit control file information caching through  
10 its own local cache buffer, and controls disk data caching through a large system memory which is accessible on a bus by any of the processors.



1/12

FIG.-1

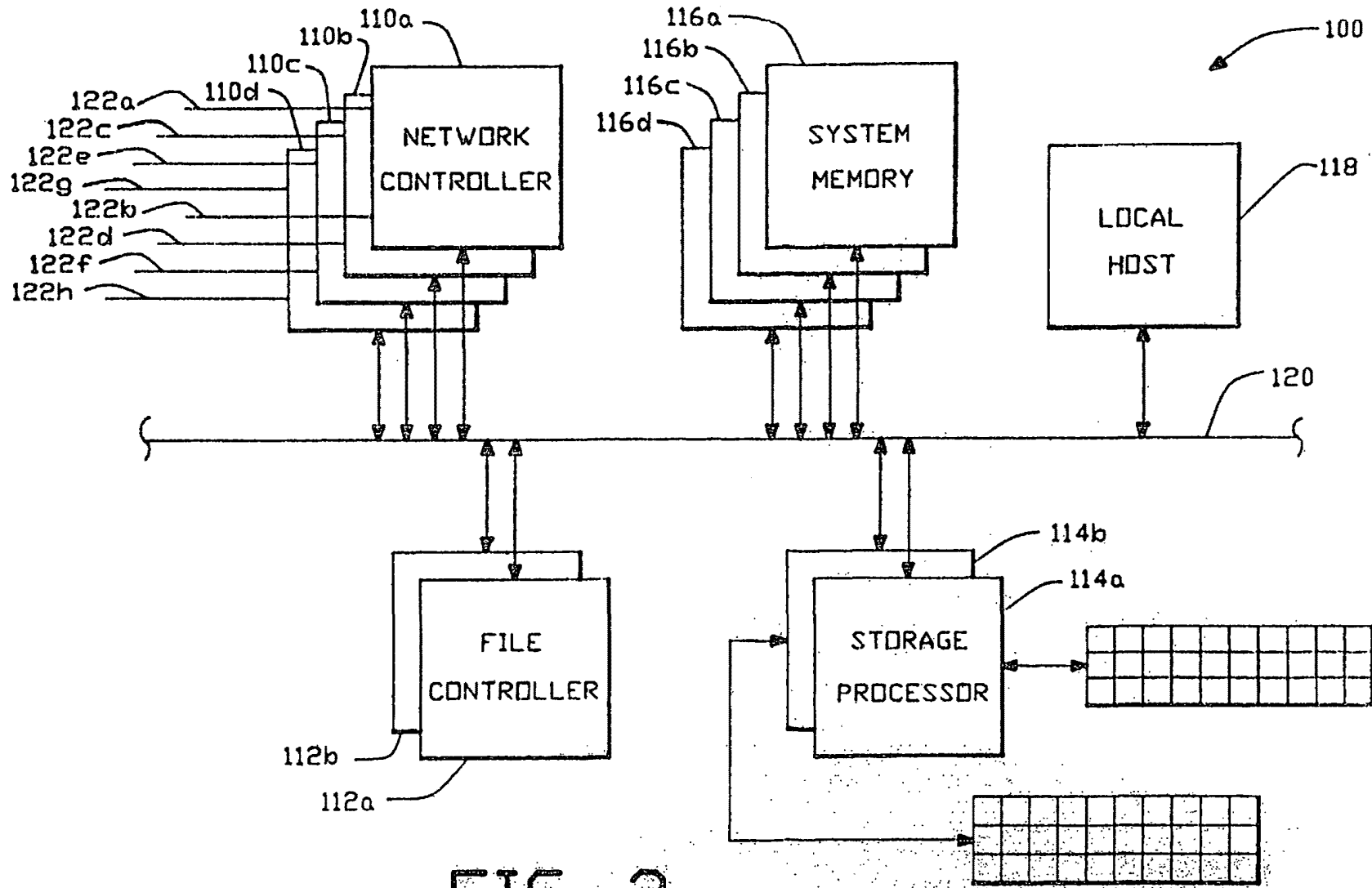


FIG.-2

2/12

65905/97

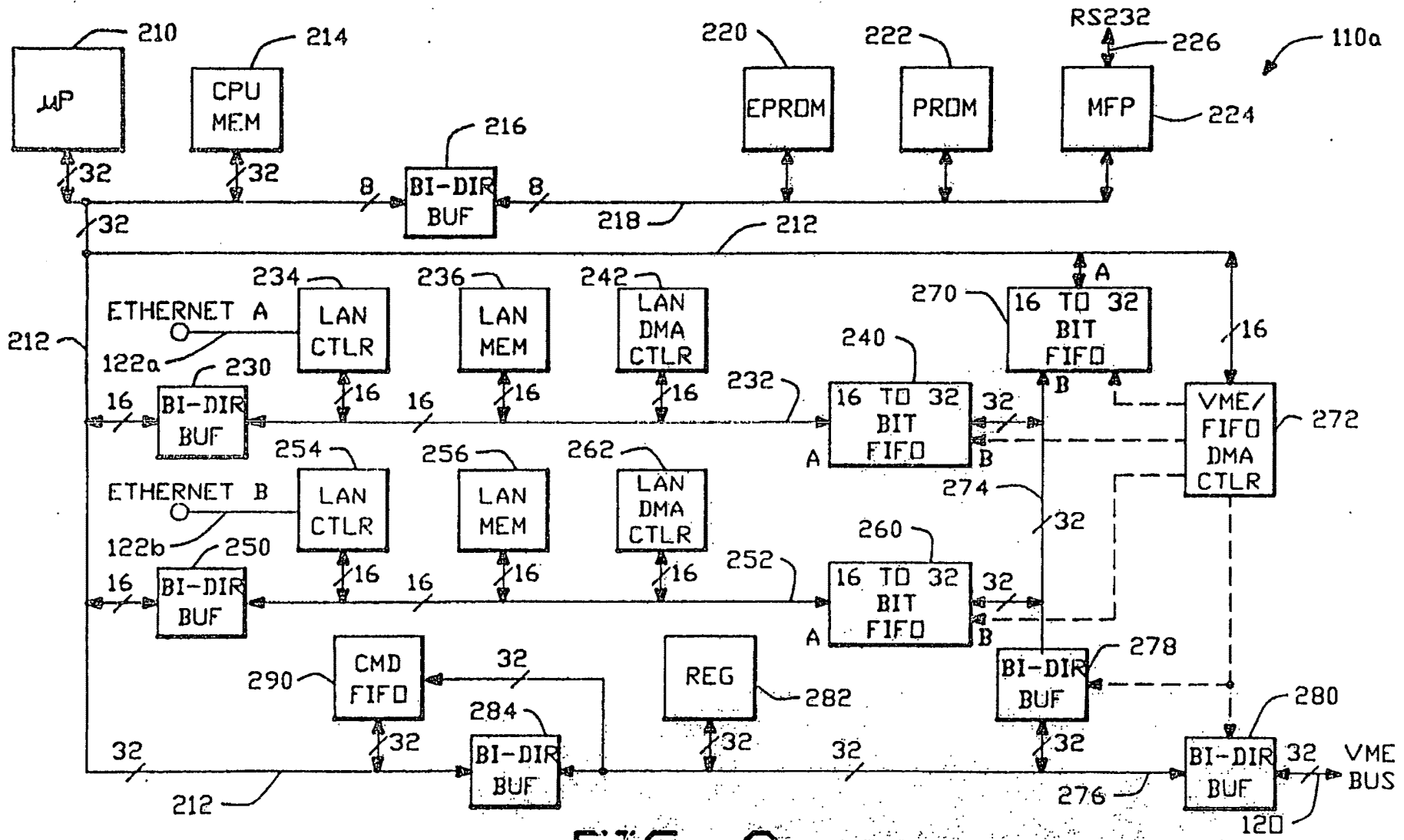
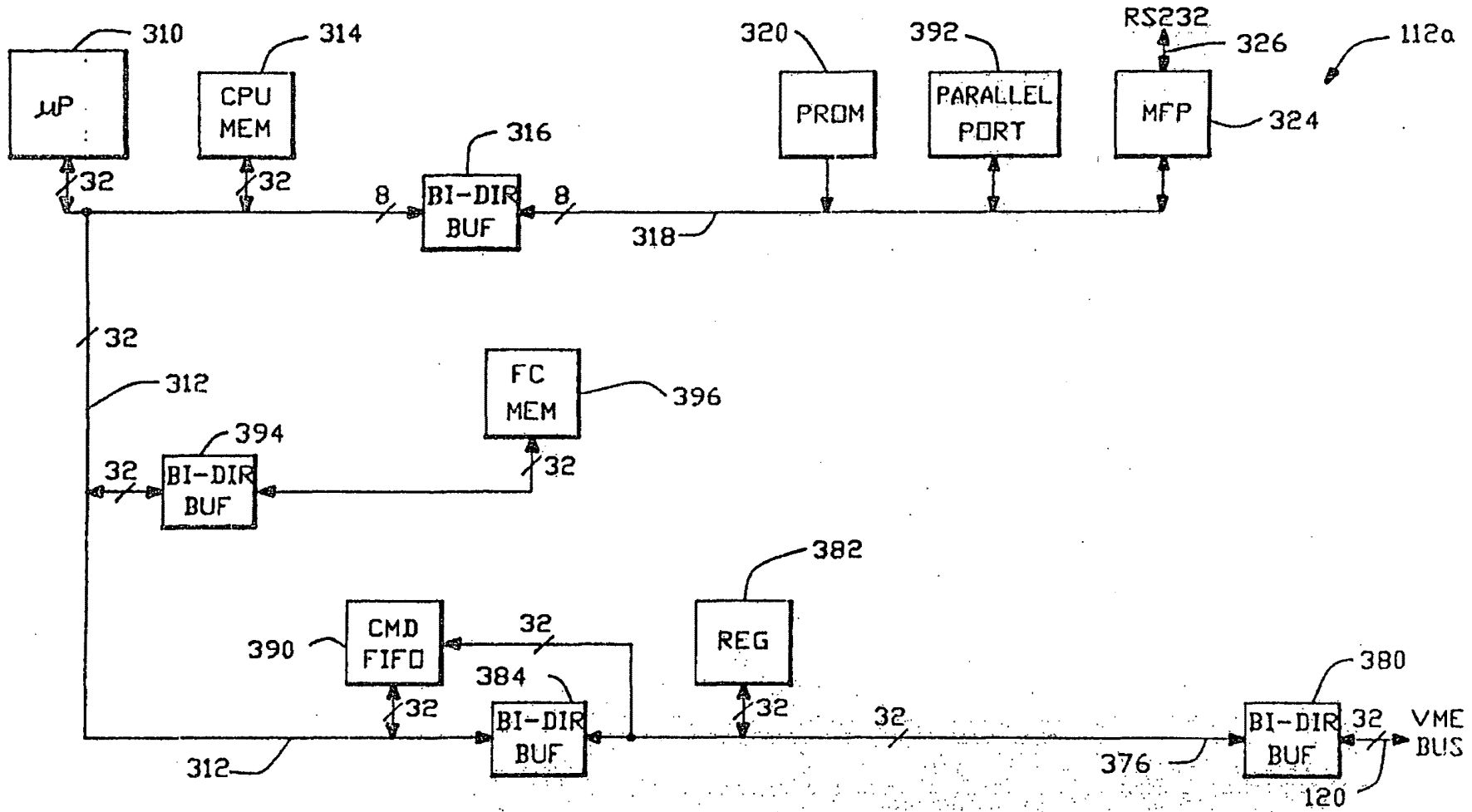


FIG.-3 (NETWORK CONTROLLER)

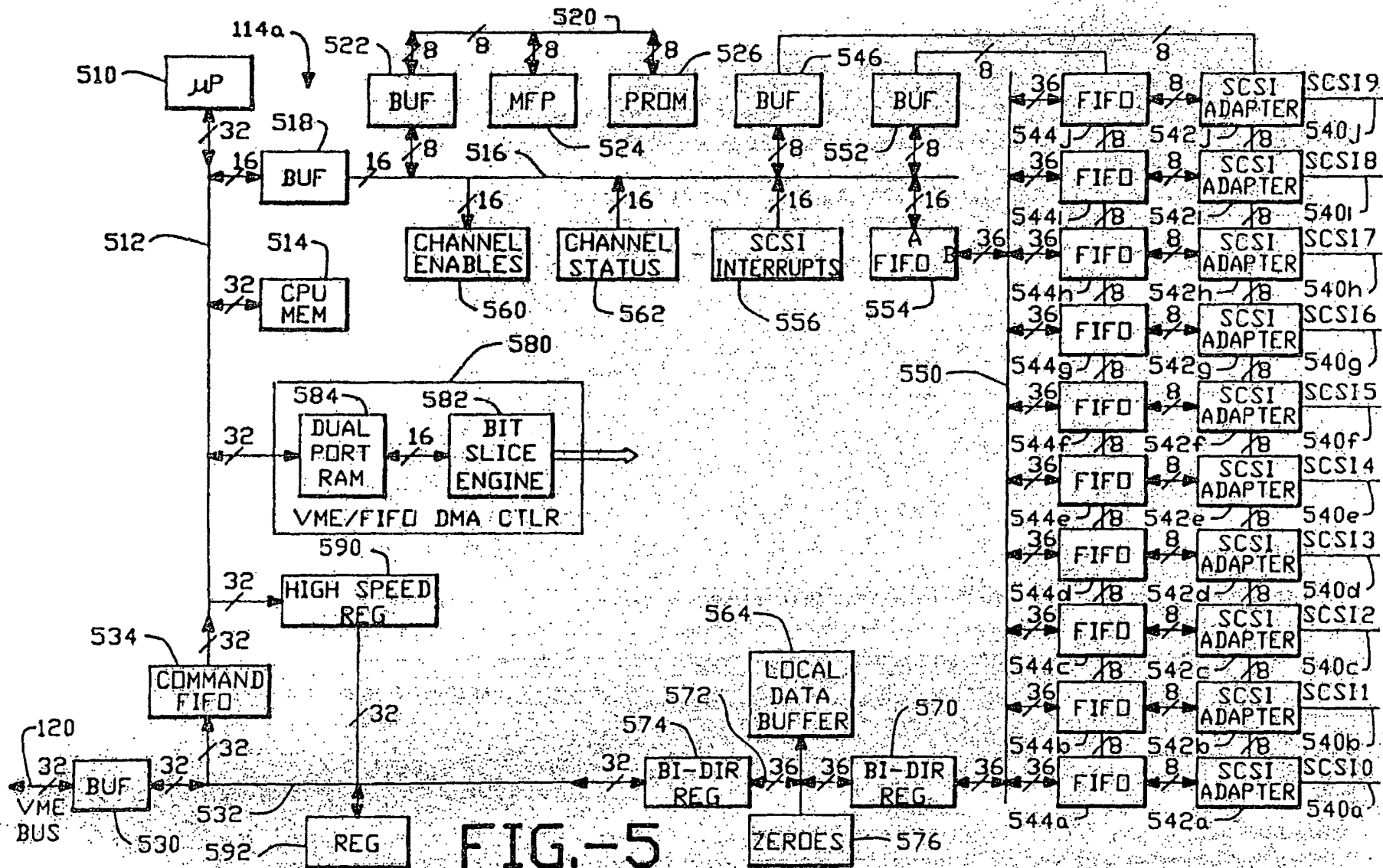
3/12



4/12

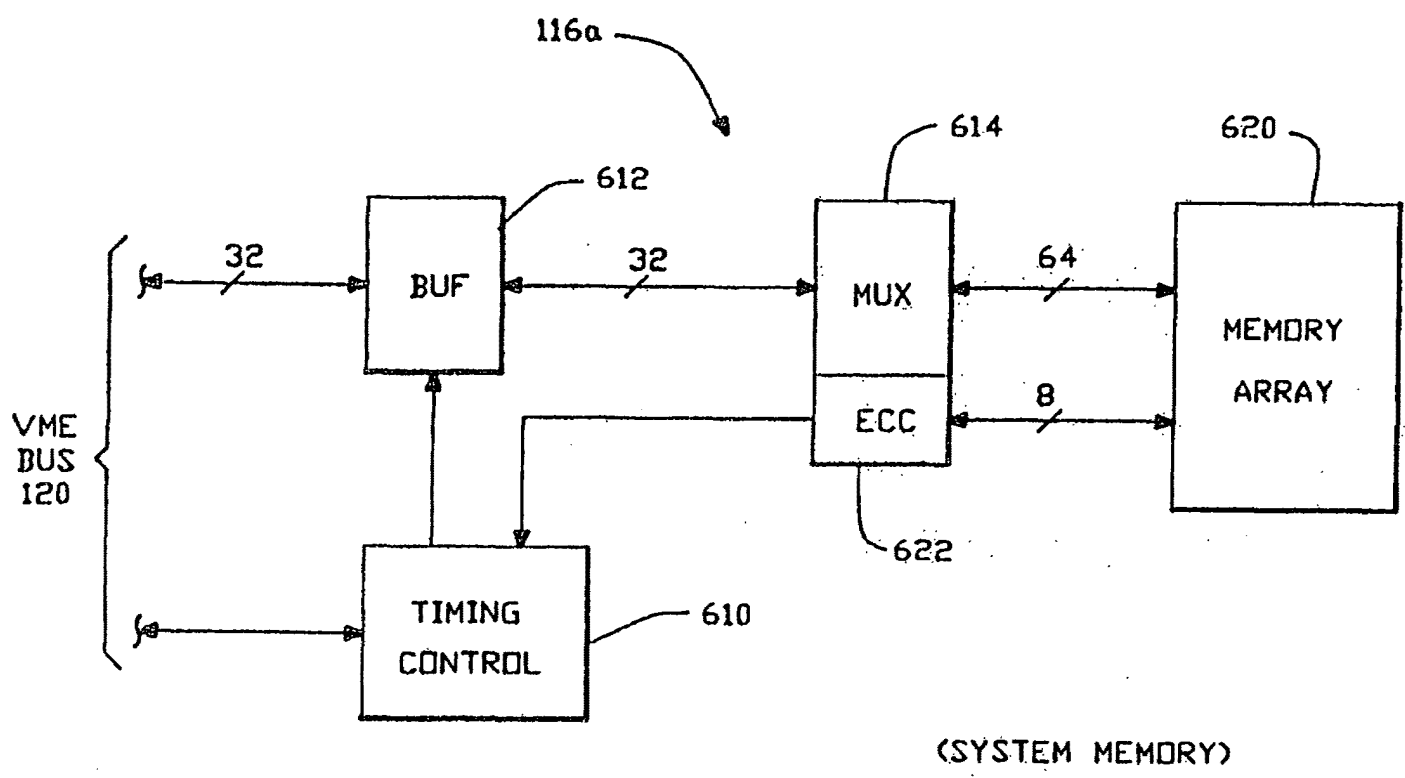
FIG. 4 (FILE CONTROLLER)

20 00 00 00 00



5/12

... ..



6/12

FIG.-6

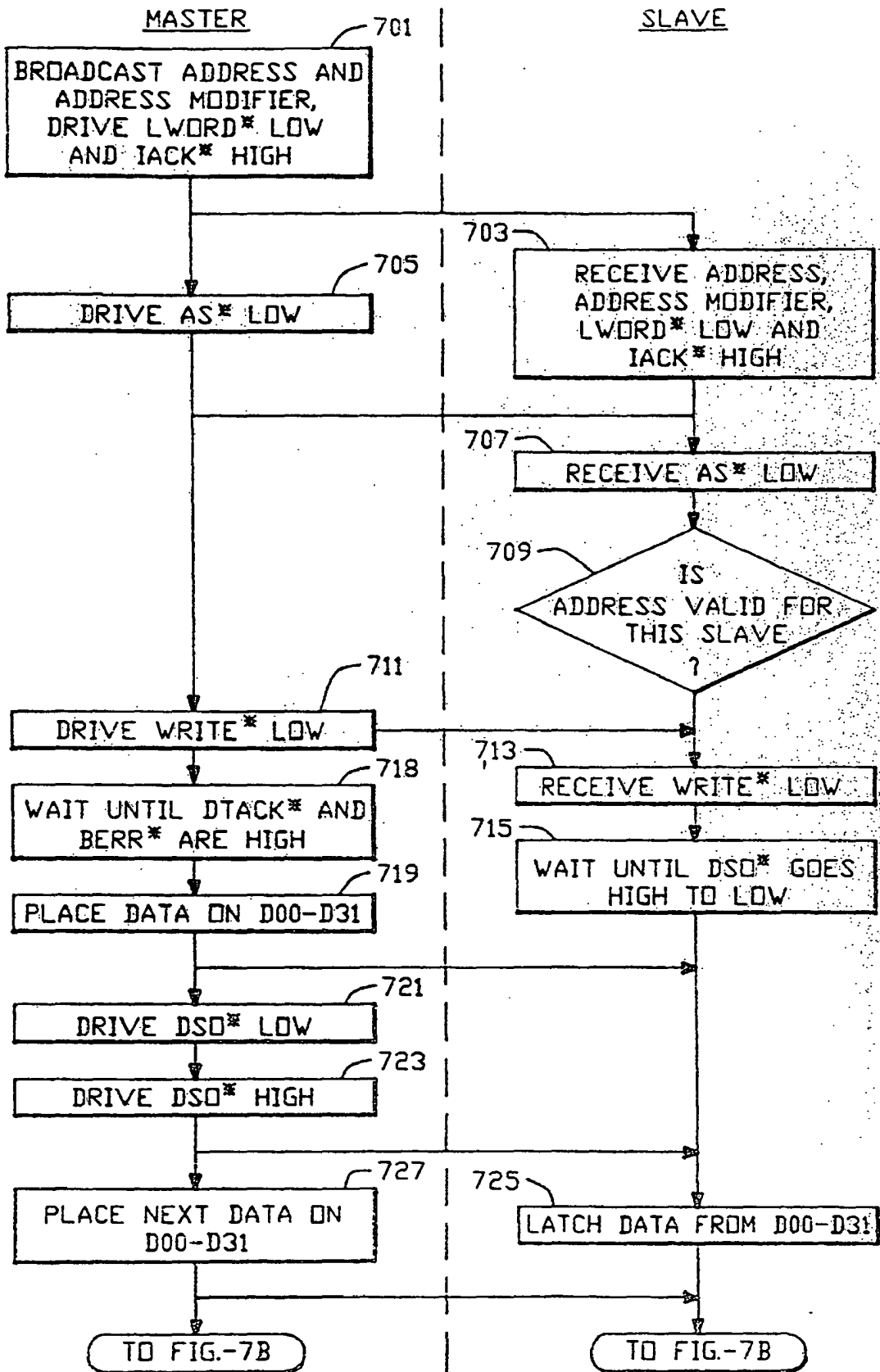


FIG.-7A



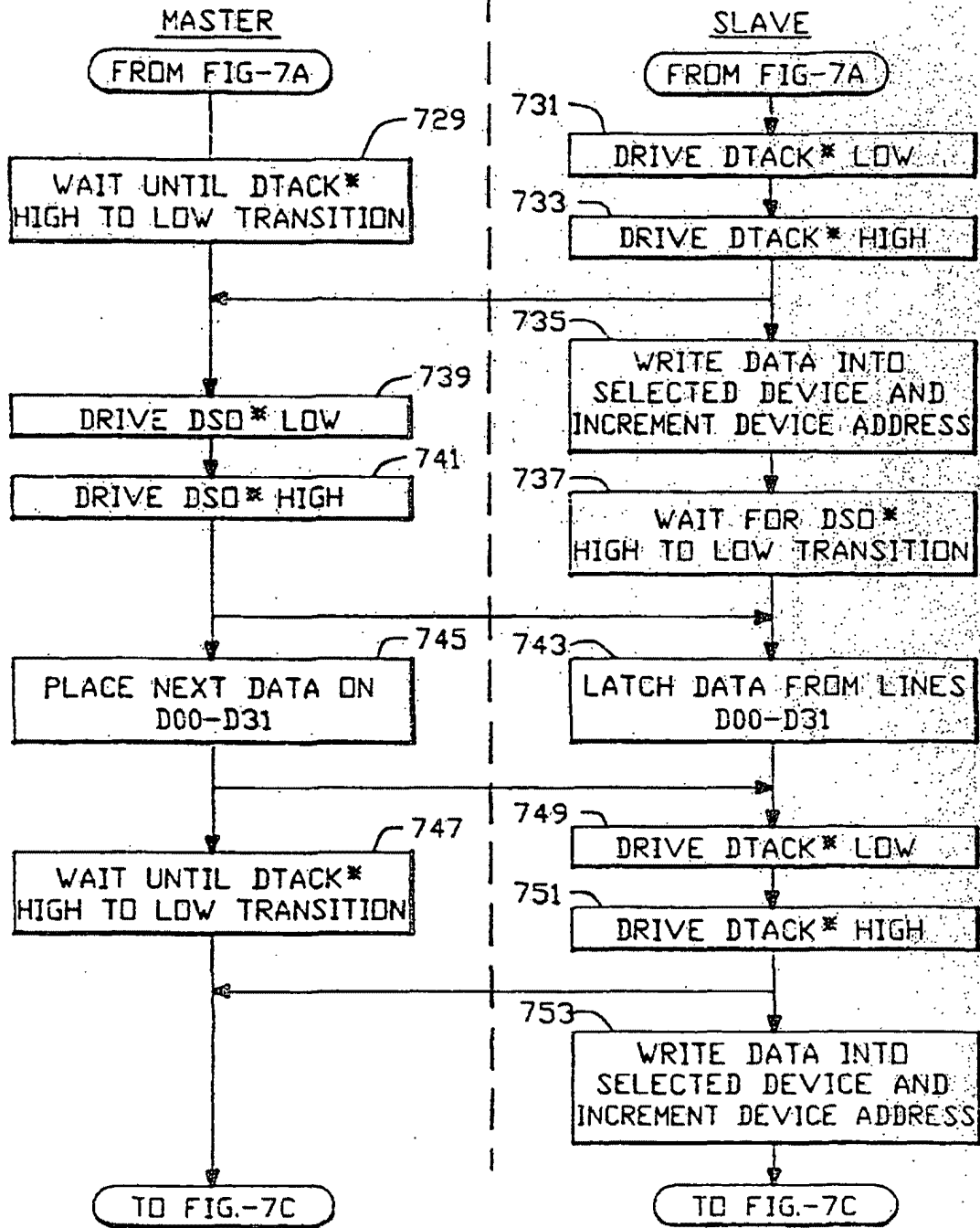


FIG.-7B

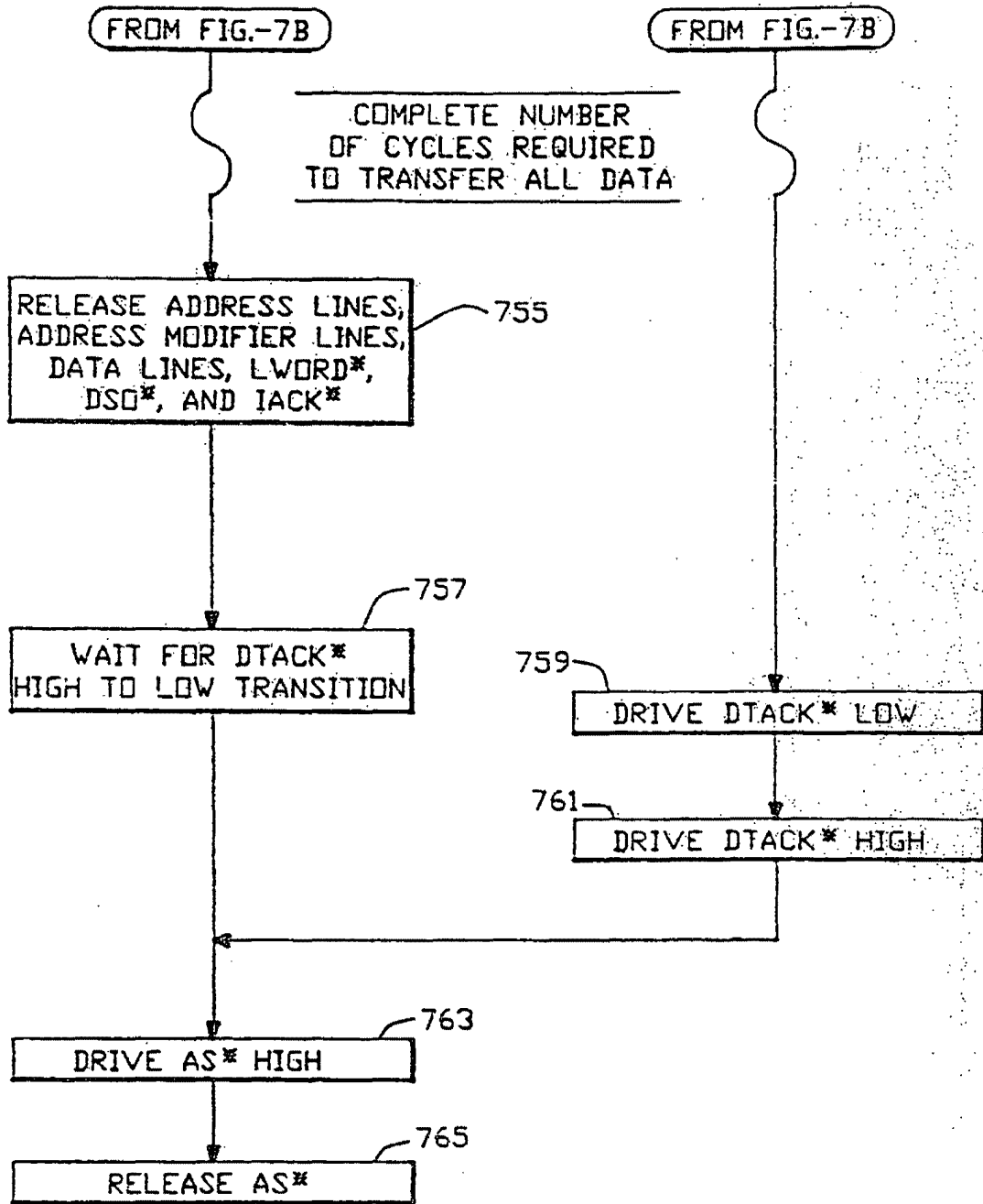


FIG.-7C

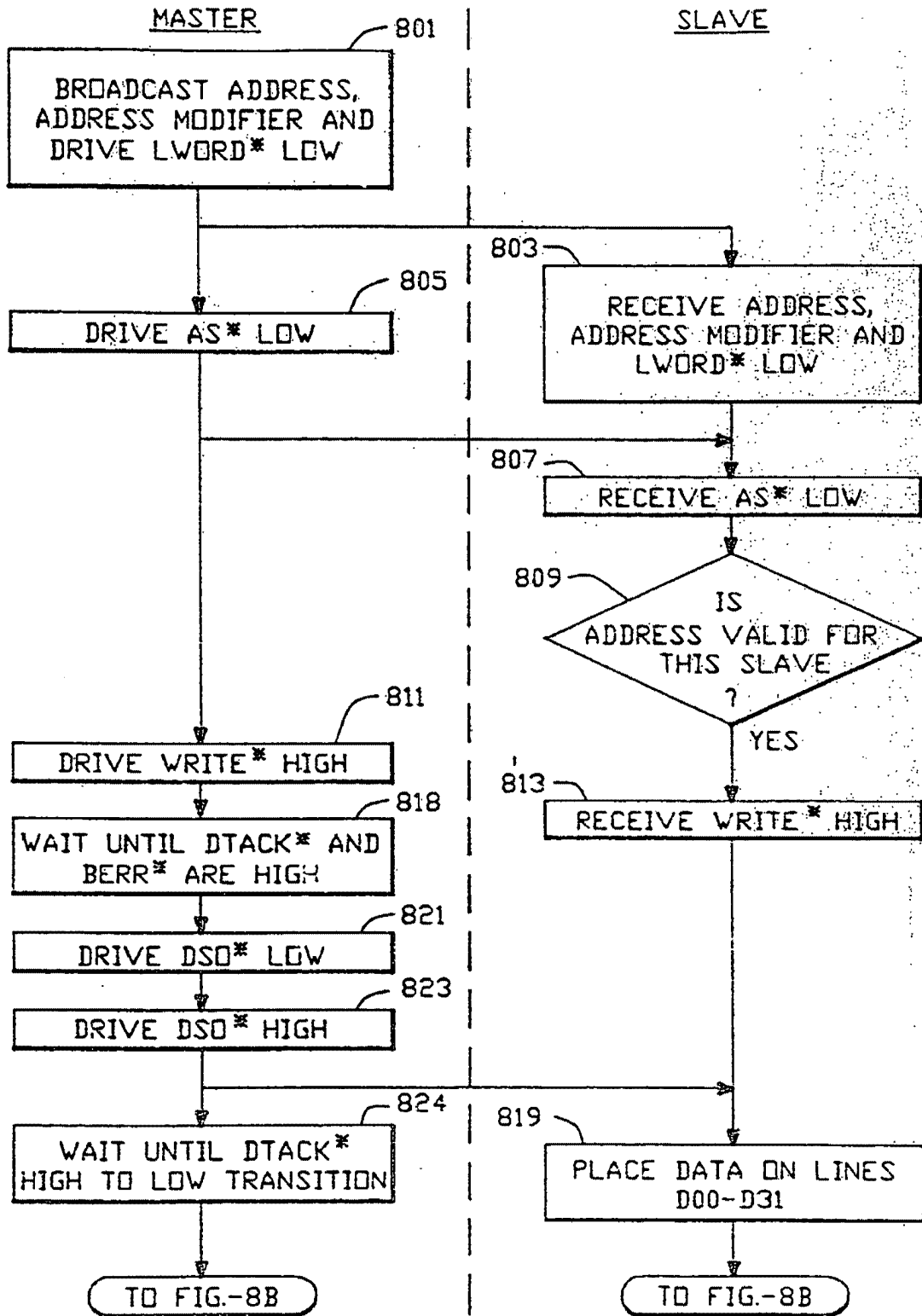


FIG.-8A

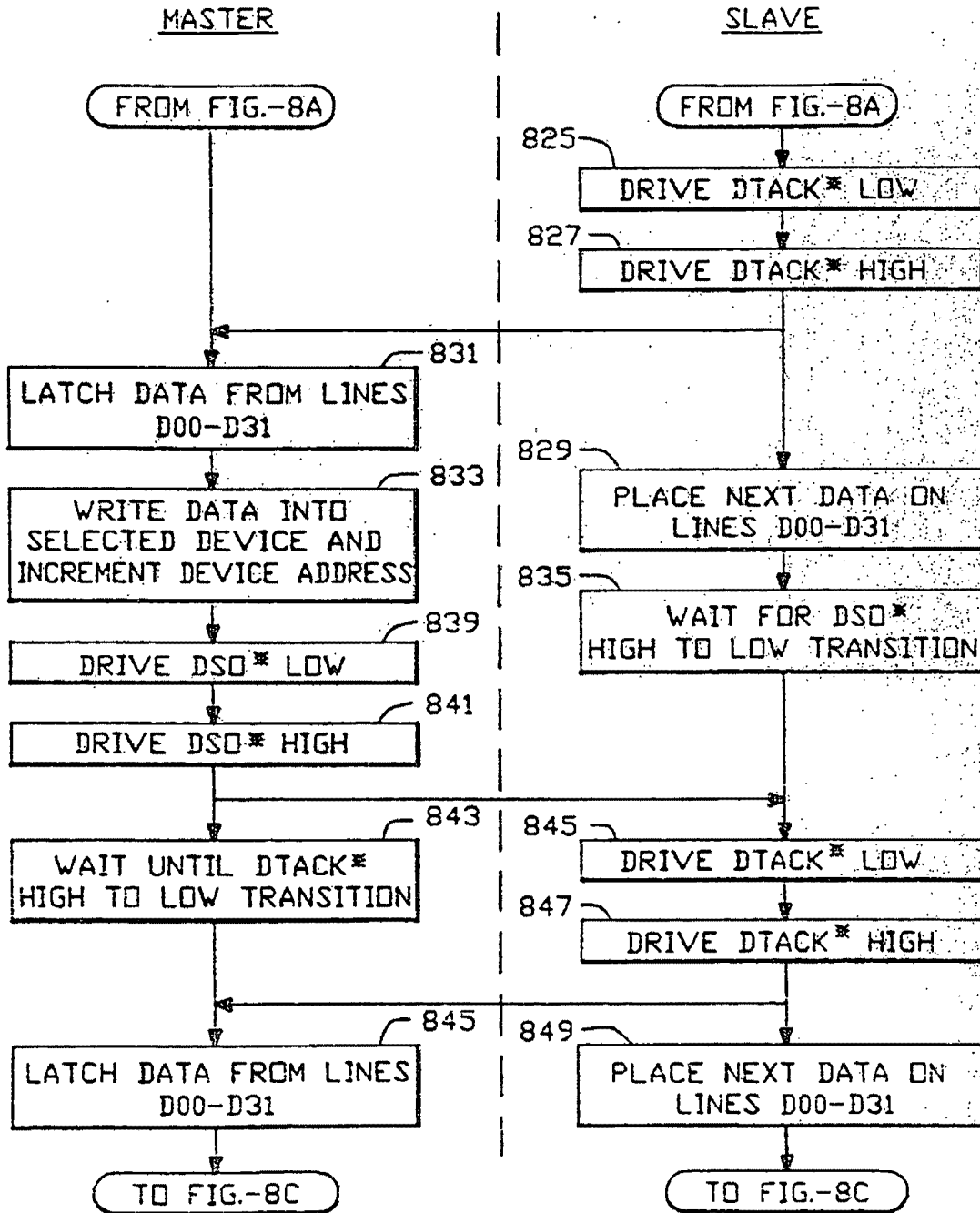


FIG.-8B

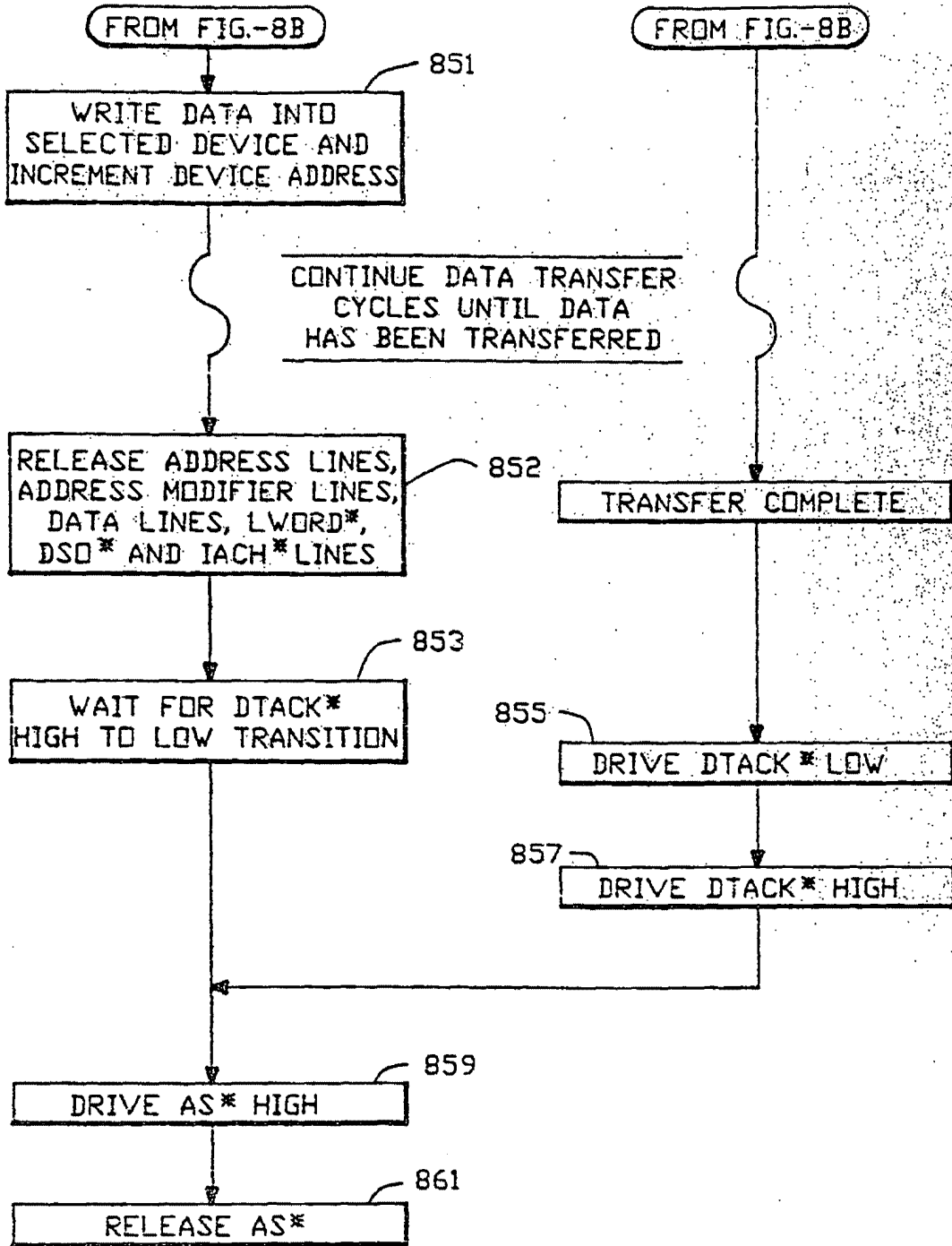


FIG.-8C