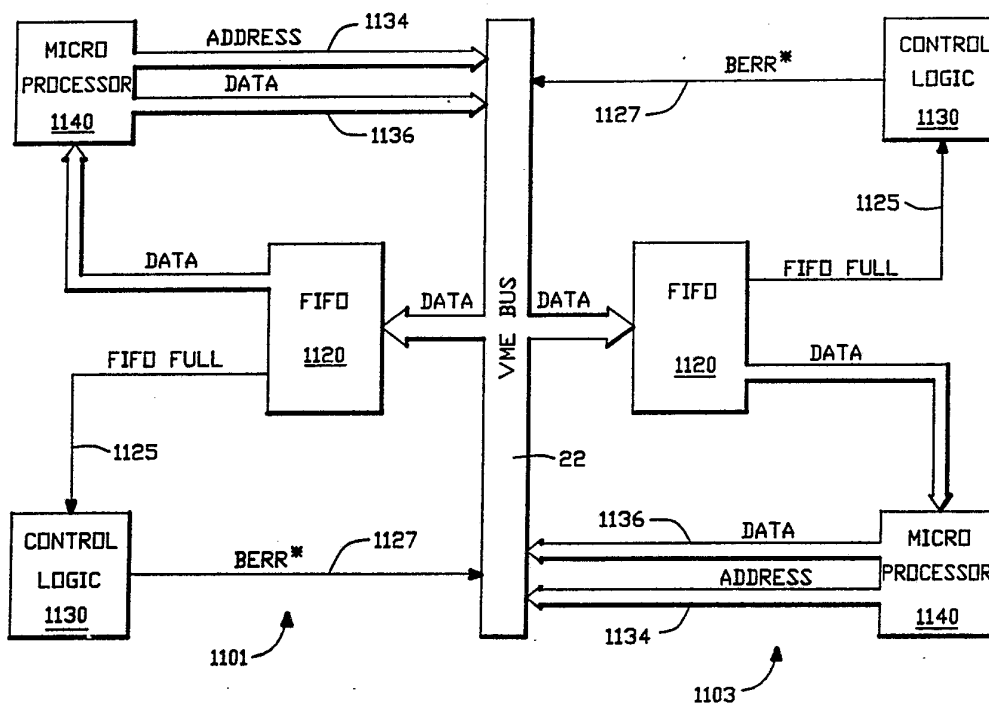




## INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification <sup>5</sup> :  G06F 13/38, 13/00	A1	(11) International Publication Number: WO 91/11768  (43) International Publication Date: 8 August 1991 (08.08.91)
(21) International Application Number: PCT/US90/04697  (22) International Filing Date: 20 August 1990 (20.08.90)  (30) Priority data: 474,350 2 February 1990 (02.02.90) US  (71) Applicant: AUSPEX SYSTEMS, INC. [US/US]; 2952 Bunker Hill Lane, Santa Clara, CA 95054 (US).  (72) Inventors: PITTS, William, M. ; 780 Mora Drive, Los Al- tos, CA 94022 (US). BLIGHTMAN, Stephen, E. ; 775 Salt Lake Drive, San Jose, CA 95133 (US). STARR, Da- ryl, D. ; 446 Folsom Court, Milpitas, CA 95035 (US).		(74) Agents: FLIESLER, Martin, C. et al.; Fliesler, Dubb, Meyer & Lovejoy, 4 Embarcadero Center, Suite 400, San Francisco, CA 94111 (US).  (81) Designated States: AT (European patent), AU, BE (Euro- pean patent), CA, CH (European patent), DE (Euro- pean patent)*, DK (European patent), ES (European pa- tent), FR (European patent), GB (European patent), IT (European patent), JP, KR, LU (European patent), NL (European patent), SE (European patent).  <b>Published</b> <i>With international search report.</i>

## (54) Title: BUS LOCKING FIFO MULTI-PROCESSOR COMMUNICATION SYSTEM



## (57) Abstract

A message transfer system for transferring message data from a master processor (1140) across a VMEbus (22) to a slave processor (1140). The message transfer system includes a FIFO (1120) interconnected to the VMEbus (22) for receiving and storing the message data transferred from the master processor (1140). The FIFO (1120) is unable to store message data, and generates a FIFO FULL signal (1125) to indicate the existence of the FIFO FULL state. The system further includes a means (1130) interconnected to the FIFO (1120) and the VMEbus (22) responsive to the receipt of a FIFO FULL signal (1125) from the FIFO (1125).

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

-1-

BUS LOCKING FIFO MULTI-PROCESSOR COMMUNICATION  
SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is related to the  
following U.S. Patent Applications:

- 5           1.    MULTIPLE FACILITY OPERATING SYSTEM  
ARCHITECTURE, inventors: David Hitz, Allan  
Schwartz, James Lau, and Guy Harris;
2.    PARALLEL I/O NETWORK FILE SERVER  
ARCHITECTURE, inventors: John Row, Larry Boucher,  
10   William Pitts, and Steve Blightman;
3.    ENHANCED VMEBUS PROTOCOL UTILIZING  
SYNCHRONOUS HANDSHAKING AND BLOCK MODE DATA  
TRANSFER, inventor: Daryl D. Starr;
- 15          4.    HIGH SPEED, FLEXIBLE SOURCE/DESTINATION DATA  
BURST DIRECT MEMORY ACCESS CONTROLLER, inventor:  
Daryl D. Starr;

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

20

-2-

Field of the Invention:

The present invention relates generally to the field of microcomputers and, more specifically, to a bus locking FIFO communication system for use in a multiprocessor computer system.

Background of the Invention:

Over the past several years, the computer industry has experienced a remarkable evolution in the architecture of technical and office computing systems. Distributed "smart" workstations have increasingly replaced the simple "dumb" terminal attached to a mainframe or microcomputer. These "smart" workstations are, themselves, computers having local processing ability and local memory storage. Such "smart" workstations comprise part of a larger network, which includes a wide variety of processors, data storage and communication devices, and other peripherals.

A workstation network generally consists of individual user workstations (referred to as "clients") and shared resources for filing, data storage, printing and wide-area communications (referred to individually as "servers"). The clients and servers are inter-connected along a local area network ("LAN"), such as an ethernet. Multiple ethernets may be connected to one another by a backbone ethernet.

Clients along an ethernet are typically connected to a server providing the clients with data and storage facilities. Servers that primarily provide for file storage access are referred to as "file servers". A conventional server might include a central processing unit ("CPU") coupled to the ethernet. The CPU itself is coupled to a primary

-3-

memory device. Both the CPU and the primary memory device are connected to a conventional input-output device ("I/O"), such as a bus. Using the bus, the CPU may communicate with other devices such as disk controllers, for mass storage, or other peripheral devices.

Although processor technology and performance has increased significantly in recent years, input/output performance has not commensurately improved to date. Thus, although the processing performance capabilities of the CPU are considerable, the overall performance of the system is less formidable due to the lower performance threshold of I/O embodied in the bus.

The level of performance of any bus is largely determined by the time required to execute data transfer transactions across the bus. If the transaction time for a given transaction across the bus can be optimized to the shortest period of time possible, the bus will be capable of handling more transactions in a given period of time. Hence, performance is increased as the bus is capable of handling a greater number of transactions during a given period of time.

The VME backplane bus (hereinafter "VMEbus") is one of the most popular I/O bus systems presently in use. The VMEbus is widely implemented and standard throughout the industry. To this end, the Standards Committee of the Institute of Electrical and Electronics Engineers ("IEEE") has formulated and published VMEbus standards in a publication entitled VMEbus Specification Manual, revision D1.2. (hereinafter "the VMEbus standard"), which is hereby incorporated by reference.

-4-

The standard VMEbus interface system consists of backplane interface logic, four groups of signal lines referred to as "buses," and a collection of functional modules which communicate with one another using the signal lines. The four buses are the data transfer bus ("DTB"), arbitration bus, priority interrupt bus and utility bus. The present application is principally concerned with the DTB.

The DTB allows "masters," which are functional modules, such as the CPU or other intelligent controllers connected to the VMEbus, that initiate DTB cycles, to direct the transfer of binary data between themselves and "slaves." A "slave" is a functional module, such as primary memory, which detects DTB cycles initiated by a "master" and, when those cycles specify its participation, transfers data to or receives data from its "master."

There are seven DTB cycles which a "master" may implement on the DTB: READ, WRITE, BLOCK READ, BLOCK WRITE, READ-MODIFY-WRITE, ADDRESS ONLY, and INTERRUPT ACKNOWLEDGE CYCLE.

In a READ cycle, one, two, three or four bytes of parallel data are transferred across the DTB from master to slave. The READ cycle begins when the master broadcasts an address and an address modifier and places data on the DTB. Each slave captures the address and address modifier and determines whether it is to respond to the cycle. The intended slave recipient retrieves the data from its internal storage and places the data on the DTB, acknowledging the data transfer.

In a WRITE cycle, one, two, three or four bytes of parallel data are transferred across the bus from a master to a slave. The cycle commences when the master broadcasts an address and address modifier

**SUBSTITUTE SHEET**

-5-

and places data on the DTB. Each slave on the bus captures the address and address modifier and determines whether it is to respond to the cycle. The intended slave(s) stores the data and acknowledges the transfer.

The BLOCK READ cycle is a DTB cycle used to transfer a block of one to two-hundred fifty-six bytes from a slave to a master. The BLOCK READ transfer is accomplished using a string of one, two or four byte-wide (i.e., 8, 16, or 32 bit-wide data words) data transfers. Once the block transfer is started, the master does not release the DTB until all of the bytes have been transferred. The BLOCK READ cycle differs from a string of READ cycles in that the master broadcasts only one address and address modifier at the beginning of the cycle. The slave increments the address on each transfer in order that the data for the next transfer is retrieved from the next higher location.

The BLOCK WRITE cycle, like the BLOCK READ cycle, is a DTB cycle used to transfer a block of one to two-hundred fifty-six bytes from a master to a slave. The BLOCK WRITE transfer is accomplished using a string of one, two or four byte-wide data transfers. Once the block transfer is started, the master does not release the DTB until all of the bytes have been transferred. The BLOCK WRITE cycle differs from a string of WRITE cycles in that the master broadcasts only one address and address modifier at the beginning of the cycle. The slave increments the address on each transfer in order that the data for the next transfer is retrieved from the next higher location.

-6-

The READ-MODIFY cycle is a DTB cycle used to both read from and write to a slave location without permitting another master access to the slave location.

5       The ADDRESS-ONLY cycle consists only of an address broadcast. Data is not transferred. Slaves do not acknowledge ADDRESS-ONLY cycles and the master terminates the cycle without waiting for an acknowledgement.

10       It should be noted that this differs from "synchronous" systems in that in totally "synchronous" systems the response of the slave is irrelevant. This initiation of a DTB cycle is referred to in the art as "handshaking." After a  
15       master initiates a data transfer cycle it waits for the designated slave to respond before finishing the cycle. The asynchronous nature of the VMEbus allows a slave to take as long as it needs to respond. The VMEbus requires four propagations across the DTB to  
20       complete a single handshake sequence. If a slave fails to respond because of a malfunction or if the master accidentally addresses a location where there is no slave, the bus timer intervenes allowing the cycle to be terminated.

25       The VMEbus standard specifies the use of location monitors, which are on the functional modules, to monitor data transfers over the DTB. Each operates to detect accesses to the locations it has been assigned to watch. When an access to one of these  
30       assigned locations occurs, the location monitor typically signals its on-board processor by means of an interrupt request signal. In such a configuration, if processor A writes into the global VMEbus memory monitored by processor B's location  
35       monitor, processor B will be interrupted.



-7-

The DTB includes three types of lines: addressing lines, data lines and control lines.

5 Masters use address lines numbers 2 through 31, denoted as A02 through A31, to select a four-byte group to be accessed. Four additional lines, data  
10 strobe zero (DS0\*), data strobe one (DS1\*), address line number one (A01) and longword (LWORD\*), are then used to select which byte locations within the four-byte group are accessed during the data  
15 transfer. The asterisk following the abbreviated line designation denotes that these lines are "active low" (i.e., considered "active" when driven low). Using these four lines, a master can access one, two, three or four-byte locations  
simultaneously, depending upon the type of cycle initiated.

The DTB includes six address modifier lines which allow the master to pass additional binary information to the slave during a data transfer.  
20 Sixty-four possible modifier codes exist, which are classified into each of three categories: defined, reserved and user defined. User defined codes may be used for any purpose which the user deems appropriate. Typical uses of the user defined codes  
25 include page switching, memory protection, master or task identification, privileged access to resources and so forth.

30 Thirty-two data lines, D00 through D31, actually transfer data across the bus. The master may simultaneously access up to four byte locations. When the master has selected the byte locations to be accessed, it can transfer binary data between itself and those locations over the data bus.

-8-

The DTB includes six control lines: address  
strobe (AS\*), data strobe zero (DS0\*), data strobe  
one (DS1\*), bus error (BERR\*), data transfer  
acknowledge (DTACK\*), and read/write (WRITE\*). The  
5 VME standard requires that the control lines be  
considered "active" when driven low.

A falling edge on the AS\* line informs all slave  
modules that the broadcasted address is stable and  
can be captured.

10 DS0\* and DS1\*, in addition to their function in  
selecting byte locations for data transfer, also  
serve control functions. On WRITE cycles, the first  
falling edge of the data strobe indicates that the  
master has placed valid data on the data bus. On  
15 READ cycles, the first rising edge tells the slave  
when it can remove valid data from the DTB.

A slave will drive DTACK\* low to indicate that it  
has successfully received the data on a WRITE cycle.

On a READ cycle, the slave drives DTACK\* low to  
20 indicate that it has placed data on the DTB.

The BERR\* line is an open-collector signal driven  
low by the slave or the bus timer to indicate to the  
master that the data transfer was unsuccessful. For  
example, if a master tries to write to a location  
25 which contains Read-Only memory, the responding  
slave might drive BERR\* low. If the master tries to  
access a location that is not provided by any slave,  
the bus timer would drive BERR\* low after waiting a  
specified period of time.

30 WRITE\* is a level significant line strobed by the  
leading edge of the first data strobe. It is used  
by the master to indicate the direction of data  
transfer operations. When WRITE\* is driven low, the

-9-

data transfer direction is from the master to the slave. When WRITE\* is driven high, the data transfer direction is from the slave to the master.

5 The VMEbus standard sets forth a handshake which requires four separate propagations across the VMEbus. The master asserts DS0\* and DS1\* to initiate the data transfer cycle. The slave, in response to the master's assertion of DS0\* and DS1\*, asserts DTACK\*. In response to the assertion of  
10 DTACK\* by the master, the master deasserts DS0\* and DS1\*. The slave, in response, deasserts DTACK\* to complete the handshake. Each four of these propagations is required to accomplish the handshake.

15 The maximum transfer rate across a typical VMEbus is generally in the range of 20 to 30 megabytes per second. However, in situations where a great deal of data must be transferred very quickly from one device on the VMEbus to another device on the VMEbus  
20 or a large number of data transfers need to be made, this transfer rate can oftentimes be slow enough to result in processing delays. Accordingly, in order to maximize data transfer and processing efficiency, the transfer rate of data across the VME backplane  
25 bus should be increased.

Another significant aspect of VMEbus architecture is the "message" passing system implemented to pass "messages" across the bus. A typical "message" might consist of a message from a CPU controller to  
30 a disk controller to read a particular block of data on a disk controlled by the disk controller. As various processes are running on the various functional modules, it is necessary to send "messages" across the bus from one module to  
35 another.

-10-

The most common manner in which messages are delivered is commonly referred to as the "mailbox method" of message transfer. Under the mailbox method, the processor which originates the message to be delivered is designated as the "sender" and the processor to which the message is to be sent is designated as the "recipient." All processors adapted to receive messages are equipped with a "mailbox."

The processors located on the various functional modules allocate memory for the purposes of storing the "messages." The messages are typically of fixed length of 128 bytes. Consequently, the various processors allocate 128 bytes of memory to a "message buffer." To initiate the transmission of a message, the sender inserts a message to be sent to the designated recipient into the message buffer. The message must then be delivered to recipient.

The sender will institute a conventional VMEbus READ operation directed to the recipient's mailbox. If the data read from the recipient's mailbox is a "0," this indicates that a slot is available in the recipient's mailbox. The sender is then free to write a "pointer" into the recipient's mailbox by transmitting the pointer across the VMEbus using a WRITE operation. The "pointer" comprises the address of the sender's message buffer.

If the data read from the mailbox is nonzero, the recipient's mailbox is full and the sender is required to wait a predetermined period of time before attempting once again to ascertain whether it may write a pointer into the recipient's mailbox. After the predetermined period of time has elapsed, the sender will then institute a READ operation directed to the mailbox in order to ascertain

-11-

whether a slot is available in the recipient's mailbox. Upon detecting an available slot, the sender will initiate a WRITE operation to place a pointer in the mailbox.

5       The recipient monitors its mailbox for incoming pointers. Upon receiving a pointer, the recipient may initiate a sequence of READ operations to retrieve the message from the sender's message buffer.

10       The recipient processor must include one mailbox for each potential sender. This one-to-one requirement is mandated by the fact that, unless one mailbox is dedicated to each sender, collisions between two different senders simultaneously writing  
15       into the same mailbox. Thus, in the prior art, the recipient averts such collisions by establishing one mailbox for each potential sender. A recipient processor will typically include a plurality of mailboxes, each corresponding to a particular  
20       potential sender. Because the recipient must establish one mailbox for each potential sender, a great deal of the recipient's memory is used to set up the mailboxes. Moreover, because each recipient has multiple mailboxes capable of simultaneously  
25       receiving messages from a number of different senders, the time required to poll all of the multiple mailboxes can be substantially more than the time required to poll just one mailbox.

30       Where fast transfers are the objective, the mailbox method is most undesirable. The mailbox method requires numerous transaction across the VMEbus. As noted, the sender must initiate a READ operation at least once, if not multiple times, to initially detect the availability of a mailbox slot.  
35       To place the pointer in the recipient's mailbox, the

-12-

sender must initiate a WRITE operation. The recipient, in turn, must initiate a READ operation to retrieve the message. These multiple transactions across the VMEbus consume bus time, which results in a slower overall performance.

Hence, a need exists for a high performance message transfer system, which is compatible with the VMEbus standard. To date, there has been little progress made in increasing the transfer rate beyond the 20 to 30 megabyte per second range. Adherence to the VMEbus standard poses a problem with respect to improving the data transfer rate of the VMEbus.

The popularity of the VMEbus is due, in large part, to both the widespread implementation of the VMEbus standard and the industry-wide compatibility of other systems and peripherals. Indeed, it is this paramount consideration of compatibility which has, to some extent, retarded efforts to increase the performance of the VMEbus. The architecture of a compatible VMEbus must be consistent with the VMEbus standard. Performance problems must be resolved within the context of the VMEbus standard if compatibility is to be maintained.

#### SUMMARY OF THE INVENTION

The present invention is directed to a message transfer system for transferring message data from a first processor ("sender") to a second processor ("receiver") across a bus. The message transfer system includes a station comprising a FIFO interconnected to the bus for receiving and storing the message data transferred from the first processor. The FIFO has a FIFO FULL state, which indicates that the FIFO is unable to store message data, and generates a FIFO FULL signal to indicate

-13-

the existence of the FIFO FULL state. The system further includes a means, interconnected to the FIFO means and the VMEbus, for responding across the VMEbus to a FIFO WRITE cycle from a sender processor with the BUS ERROR signal, instead of the normal DTACK signal, when this WRITE cycle occurs while the FIFO is generating the FIFO FULL signal.

The present invention provides an apparatus and method for efficiently and rapidly transferring message data across a bus from one processor to another. The present invention's efficient and rapid performance is attributable, in large part, to the minimization of transactions across the bus. The number of bus cycles required to accomplish the transfer of message data across the bus is reduced to a single WRITE cycle. This marks a substantial improvement over prior art message transfer systems, which, as noted, require multiple cycles to accomplish the transfer of message data.

Moreover, the present invention requires only one FIFO for each processor to handle message data transfer from all senders. This likewise marks a departure from prior art systems which require that each processor include one mailbox for each potential sender. The use of a single FIFO per recipient processor is advantageous as it substantially reduces the hardware required to accomplish message data transfer and simplifies the polling function. The recipient processor needs only to look at a single FIFO for message data rather than polling multiple mailboxes for message pointers.

It is an object of the present invention to provide a multi-processor communication system which both minimizes hardware requirements necessary to

-14-

implement a message transfer function and significantly simplifies the management of messages transferred.

5

#### BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram representing the preferred embodiment of the hardware support for the present invention.

10

Fig. 2 is a block diagram illustrating the principal signal lines logically connecting the data transfer bus to a master functional unit, as required by the VMEbus standard.

15

Fig. 3 is a block diagram illustrating the principal signal lines logically connecting the data transfer bus to a slave functional unit, as required by the VMEbus standard.

Fig. 4A is a timing diagram illustrating the conventional VMEbus standard handshaking protocol.

20

Fig. 4B is a timing diagram illustrating the fast transfer mode handshake.

Fig. 5A is timing diagram illustrating the standard VMEbus protocol for data referencing during a BLOCK WRITE cycle.

25

Fig. 5B is timing diagram illustrating the fast transfer protocol of the present invention for data referencing during a BLOCK WRITE cycle.

Fig. 6A is timing diagram illustrating the standard VMEbus protocol for data referencing during a BLOCK READ cycle.

30

Fig. 6B is timing diagram illustrating the fast transfer protocol of the present invention for data referencing during a BLOCK READ cycle.

Fig. 7A is a flowchart illustrating the operation of the fast transfer protocol BLOCK WRITE cycle.



-15-

Fig. 7B is a continuation of the flowchart of Fig. 7A.

Fig. 7C is a continuation of the flowchart of Fig. 7B.

5 Fig. 8A is a flowchart illustrating the operation of the fast transfer protocol BLOCK READ cycle.

Fig. 8B is a continuation of the flowchart of Fig. 8A.

10 Fig. 8C is a continuation of the flowchart of Fig. 8B.

Fig. 9 is a timing diagram illustrating the data transfer timing associated with a fast transfer mode BLOCK WRITE operation.

15 Fig. 9A illustrates a data transfer cycle which could be inserted in the location of box 900 in Fig. 9.

Fig. 10 is a timing diagram illustrating the data transfer timing associated with a fast transfer mode BLOCK READ operation.

20 Fig. 10A illustrates a data transfer cycle which could be inserted in the location of box 1000 in Fig. 10.

25 Fig. 11 is a block diagram illustrating the principal signal lines logically connecting the data transfer bus to a modified slave functional unit, as implemented in the preferred embodiment of the present invention.

30 Fig. 12 is a flowchart illustrating a message data transfer operation accomplished under the present invention.

Fig. 13 is a block diagram of the bus-locking FIFO multi-processor communication system of the present invention.

-16-

Fig. 14 is a block diagram of the preferred embodiment of one of the processors employed in the bus-locking FIFO multi-processor communication system of the present invention.

5

## DESCRIPTION OF THE PREFERRED EMBODIMENTS

### I. System Overview

A block diagram representing the preferred embodiment of the hardware support for the present invention, generally indicated by the reference numeral 10, is provided in Fig 1. The architecture of the preferred hardware system 10 is described in the above-identified related application entitled PARALLEL I/O NETWORK FILE SERVER ARCHITECTURE which application is expressly incorporated by reference.

15

The hardware components of the system 10 include multiple instances of network controllers 12, file system controllers 14, and mass storage processors 16 interconnected by a high-bandwidth backplane bus 22. Each of these controllers 12, 14, 16 preferably include a high performance processor and local program store, thereby minimizing their need to access the bus 22. Rather, bus 22 accesses by the controllers 12, 14, 16 are substantially limited to transfer accesses as required to transfer control information and client workstation data between the controllers 12, 14, 16 system memory 18, and a local host processor 20, when necessary.

25

The illustrated system 10 configuration includes four network controllers 12A-C two file controllers 14A-B two mass storage processors 16A-B a bank of four system memory cards 18A-D and a local host processor 20 coupled to the backplane bus 22. Each network controller (NC) 12 preferably includes two independent ethernet network connections, shown as

35

-17-

the network pairs 1, 3, 5 and 7, controlled by a Motorola 68020 processor. Each of the network connections directly support the ten megabit per second data rate specified for a conventional individual Ethernet network connection. The preferred hardware embodiment of the present invention thus realizes a combined maximum data throughput potential of 80 megabits per second.

The file controllers (FC) 14, intended to operate primarily as specialized computer engines, each include a high-performance Motorola 68020 based microprocessor system, two megabytes of local program memory and a smaller half-megabyte high-speed data store.

The mass storage processors (SP) 16 function as intelligent small computer system interface (SCSI) controllers. Each includes a Motorola 68020 based microprocessor system, a local program and data memory, and an array of ten parallel SCSI channels. Drive arrays 24A-B are coupled to the storage processors 16A-B to provide mass storage. Preferably, the drive arrays 24A-B are ten unit wide arrays of SCSI storage devices and from one to three units deep uniformly. In the preferred embodiment of the present invention using conventional 768 megabyte 5¼-inch hard disk drives for each unit of the arrays 24A-B. Thus, each drive array level achieves a storage capacity of approximately 6 gigabytes, with each storage processor readily supporting 18 gigabytes, and a system 10 capable realizing a total combined data storage capacity of 36 gigabytes.

-18-

The local host processor 20, in the preferred embodiments of the present invention, is a Sun 3/40 central processor card, model Sun 3E120, manufactured and distributed by Sun Microsystems, Inc.

Finally, the system memory cards 18 each provide 32 megabytes of 32-bit memory for shared use within the computer system 10. The memory is logically visible to each of the processors of the system 10.

A VMEbus 22 is used in the preferred embodiments of the present invention to interconnect the network controllers 12, file controllers 14, storage processor 16, system memory 18, and local host 20. The hardware control logic for controlling the VMEbus 22, as at least implemented on the network controller 12 and storage processor 16, has been enhanced to support the bus master fast transfer protocol of the present invention. The system memory 18 also implements the modified slave VMEbus control logic, also in accordance with the present invention, to allow the system memory 18 to act as the data transfer data source or destination for the network controller 12 and storage processors 16.

It should be understood that, while the system 10 configuration represents the initially preferred maximum hardware configuration, the present invention is not limited to the preferred number or type of controllers or the preferred size and type of disk drives.

## II. Enhanced VMEbus Overview

Figs. 2 and 3 are, respectively, block diagrams of typical master and slave functional units (respectively hereinafter "master" and "slave"). The signal lines interconnecting the master and

-19-

slave across the data transfer bus (DTB), as shown in Figs. 2 and 3, are the following:

5	A01-A15	ADDRESS bus (bits 1-15) - Three-state driven address lines that are used to broadcast a short, standard, or extended address.
10	A16-A23	ADDRESS bus (bits 16-23) - Three-state driven address lines that are used in conjunction with A01-A15 to broadcast a standard or extended address.
15	A24-A31	ADDRESS bus (bits 24-31) - Three-state driven address lines that are used in conjunction with A01-A23 to broadcast an extended address.
20		
25	AM0-AM5	ADDRESS MODIFIER (bits 0-5) - Three-state driven lines that are used to broadcast information such as address size, cycle type, and/or MASTER identification.
30	AS*	ADDRESS STROBE - A three- state driven signal that indicates when a valid address has been placed on the address bus.
35		
40	BERR*	BUS ERROR - An open-collector driven signal generated by a SLAVE or BUS TIMER. This signal indicates to the MASTER that the data transfer was not completed.
45	D00-D31	DATA BUS - Three-state driven bidirectional data lines used to transfer data between MASTERS and SLAVES.

**SUBSTITUTE SHEET**

-20-

5  10  15	DS0*, DS1*	DATA STROBE ZERO, ONE - Three-state driven signals used in conjunction with LWORD and A01 to indicate how many data bytes are being transferred (1, 2, 3, or 4). During a write cycle, the falling edge of the first data strobe indicates that valid data is available on the data bus. On a read cycle, the rising edge of the first data strobe indicates that data has been accepted from the data bus.
20  25  30	DTACK*	DATA TRANSFER ACKNOWLEDGE - A three-state driven signal generated by a SLAVE. The falling edge of this signal indicates that valid data is available on the data bus during a read cycle, or that data has been accepted from the data bus during a write cycle. The rising edge indicates when the SLAVE has released the data bus at the end of a READ CYCLE.
35	LWORD*	LONGWORD - A three-state driven signal used in conjunction with DS0*, DS1*, and A01 to select which byte location(s) within the 4 byte group are accessed during the data transfer.
40  45	WRITE*	WRITE - A three-state driven signal generated by the MASTER to indicate whether the data transfer cycle is a read or write. A high level indicates a read operation; a low level indicates a write operation.

As shown in Fig. 2, the slave functional module  
200 is logically connected to backplane interface  
logic 210. The backplane interface logic 210 is  
connected to the data transfer bus 10 by signal

-21-

lines 220. The signal flow direction of the signal lines 320 is indicated by the direction of the respective arrows. The DTACK\* signal line originates with the slave and is driven by a conventional 64 mA three-state driver. The data lines are, of course, bidirectional, as shown in Fig. 2.

As shown in Fig. 3, the master functional module 300 is logically connected to backplane interface logic 310. The backplane interface logic 310 is connected to the data transfer bus 10 by signal lines 320. The signal flow direction of the signal lines 320 is indicated by the direction of the respective arrows. The DS0\*, DS1\*, AS\* and AM0 through AM5 signal lines originate with the master. The data lines, D00 through D31, are, of course, bidirectional, as shown in Fig. 3.

### III. Enhanced VMEbus Fast Transfer Protocol

The present invention increases the data transfer rate across the VMEbus by reducing the number of bus propagations required to accomplish handshaking and data transfer.

Fig. 4A illustrates the conventional handshaking protocol defined by the VMEbus standard. Four bus propagations are required to accomplish a handshake using the conventional VMEbus handshaking protocol. A master will initiate a data transfer over the DTB by asserting DS0\* and DS1\*, shown as propagation 1 in Fig. 4A. The addressed slave then asserts DTACK\*, shown as propagation 2 in Figure 4A. The master, upon receiving the assertion of DTACK\* driven by the slave, deasserts DS0\* and DS1\*, shown as propagation 3 in Figure 4A. The slave, upon receiving deassertion of DS0\* and DS1\*, deasserts

**SUBSTITUTE SHEET**

-22-

DTACK\*, shown as propagation 4 in Figure 4A. Upon the deassertion of DTACK by the slave, the handshake is then completed.

Fig. 4B is a timing diagram illustrating the fast transfer mode handshake protocol. Only two bus propagations are used to accomplish a handshake. At the initiation of a data transfer cycle, the master will assert and deassert DS0\* in the form of a pulse of a given duration in the manner shown as propagation 1 in Fig. 4B. The deassertion of DS0\* is accomplished without regard as to whether a response has been received from the slave. Hence, the DS0\* signal is wholly decoupled from the DTACK\* signal.

The master must then wait 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\*, shown as propagation 2 in Fig. 4B, the master can then immediately reassert data strobe, if so desired. The fast transfer mode protocol of the present invention does not require the master to wait for the deassertion of DTACK\* by the slave as a condition precedent to the 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.

It should be noted that the fast transfer protocol of the present invention does not employ the DS1\* line for data strobe purposes. The use of both DS0\* and DS1\* would be undesirable in the present context. Because DS0\* and DS1\* are driven by different drivers, skew between the signals is a



-23-

very common problem. Skew between DS0\* and DS1\* results in delay of the assertion of the data strobe condition required to signal a data transfer. Accordingly, under the present invention, the DS1\* line is not used in the handshake process. Skew problems are eliminated by referencing on DS0\* for data strobe purposes under the fast transfer mode protocol of the present invention.

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 present invention includes both synchronous and asynchronous components, it is most accurately classified as "pseudo-synchronous."

Fig. 5A is a timing diagram illustrating the standard VMEbus protocol for data referencing during a BLOCK WRITE cycle. In a standard VMEbus BLOCK WRITE operation, the data to be transferred is broadcast, as shown in Fig. 5A, and the master asserts DS0\* and DS1\*. The slave receives the data and asserts DTACK\*. Under the standard VMEbus protocol, valid data is guaranteed to be broadcast to the slave for a known period of time after the assertion of DTACK\* by the slave. The master then deasserts DS0\* and DS1\*, although valid data continues to be broadcast. The BLOCK WRITE cycle is completed upon deassertion of DTACK\* by the slave.

**SUBSTITUTE SHEET**

-24-

Fig. 5B is a timing diagram illustrating the fast transfer protocol of the present invention for data referencing during a BLOCK WRITE cycle. The transfer of data during a BLOCK WRITE cycle is referenced only to DS0\*, as shown in Fig. 5B. The master broadcasts valid data to the slave. The master then asserts DS0 to the slave, as shown in Fig. 5B. The slave is given a predetermined period of time,  $t_c$  in Fig. 5B, 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.

Fig. 6A is timing diagram illustrating the standard VMEbus protocol for data referencing during a BLOCK READ cycle. In a standard VMEbus BLOCK READ operation, the master asserts DS0\* and DS1\*, as shown in Fig. 6A. The slave, in response to the assertion of DS0\* and DS1\*, broadcasts the data to be transferred across the bus and asserts DTACK\*. Valid data is guaranteed to be broadcast to the master for a given period of time after the assertion of DTACK\* by the slave. The master then deasserts DS0\* and DS1\*, although valid data continues to be broadcast. The BLOCK READ cycle is completed upon deassertion of DTACK\* by the slave.

Fig. 6B is a timing diagram illustrating the fast transfer protocol of the present invention for data referencing during a BLOCK READ cycle. The transfer of data during a BLOCK READ cycle is referenced only to DTACK\*, as shown in Fig. 6B. The master asserts DS0\*. The slave broadcasts data to the master and then asserts DTACK\*, as shown in Fig. 6B. Under the fast transfer protocol, the master is given a predetermined period of time,  $t_c$  in Fig. 6B, after the assertion of DTACK, in which to capture the

-25-

data. Hence, master modules must be prepared to capture data at any time as DS0 is not referenced during the transfer cycle.

Fig. 7, parts A through C, is a flowchart illustrating the operations involved in accomplishing the fast transfer protocol BLOCK WRITE cycle of the present invention. 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. The IACK\* signal is a standard VMEbus protocol signal used to acknowledge an interrupt request from the priority interrupt bus.

A special address modifier broadcast by the master indicates to the slave module that the fast transfer protocol will be used to accomplish the BLOCK WRITE. In one embodiment of the invention, the hexadecimal address modifier "1f," which is one of the user defined address modifiers under the VMEbus standard, is broadcast to the slave to indicate that the fast transfer protocol will be used. However, it should be understood that any of the user defined address modifiers might be designated as the fast transfer protocol address modifier.

It should also be noted that 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.

**SUBSTITUTE SHEET**

-26-

The slave modules connected to the DTB receive the address and the address modifier broadcast by the master across the bus and receive LWORD\* low and IACK\* high 703. Shortly after broadcasting the address and address modifier 701, the master drives the AS\* signal low 705. The slave modules receive the AS\* low signal 707. Each slave individually determines whether it will participate in the data transfer by determining whether the broadcasted address is valid for the slave in question 709. If the address is not valid, the data transfer does not involve that particular slave and it ignores the remainder of the data transfer cycle.

The master drives WRITE\* low to indicate that the transfer cycle about to occur is a WRITE operation 711. The slave receives the WRITE\* low signal 713 and, knowing that the data transfer operation is a WRITE operation, awaits receipt of a high to low transition on the DS0\* signal line 715. The master will wait until both DTACK\* and BERR\* are high 718, which indicates that the previous slave is no longer driving the DTB.

The master proceeds to place the first segment of the data to be transferred on data lines D00 through D31, 719. After placing data on D00 through D31, the master drives DS0\* low 721 and, after a predetermined interval, drives DS0\* high 723.

In response to the transition of DS0\* from high to low, respectively 721 and 723, the slave latches the data being transmitted by the master over data lines D00 through D31, 725. It should be noted that the latching operation is responsive only to the DS0\* signal. In the fast transfer protocol of the present invention, DTACK\* is not referenced for purposes of latching data placed on the data lines

**SUBSTITUTE SHEET**

-27-

by the master. The master places the next segment of the data to be transferred on data lines D00 through D31, 727, and awaits receipt of a DTACK\* signal in the form of a high to low transition signal, 729 in Fig. 7B.

Referring to Fig. 7B, the slave then drives DTACK\* low, 731, and, after a predetermined period of time, drives DTACK high, 733. The data latched by the slave, 725, is written to a device, which has been selected to store the data, 735. The slave also increments the device address, 735. The slave then waits for another transition of DS0\* from high to low, 737.

To commence the transfer of the next segment of the block of data to be transferred, the master drives DS0\* low, 739 and, after a predetermined period of time, drives DS0\* high, 741. In response to the transition of DS0\* from high to low, respectively 739 and 741, the slave latches the data being broadcast by the master over data lines D00 through D31, 743. The master places the next segment of the data to be transferred on data lines D00 through D31, 745, and awaits receipt of a DTACK\* signal in the form of a high to low transition, 747.

The slave then drives DTACK\* low, 749, and, after a predetermined period of time, drives DTACK\* high, 751. The data latched by the slave, 743, is written to the device selected to store the data and the device address is incremented, 753. The slave waits for another transition of DS0\* from high to low, 737.

The transfer of data will continue in the above-described manner until all of the data has been transferred from the master to the slave. After all of the data has been transferred, the master will

**SUBSTITUTE SHEET**

-28-

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 of the present invention. To initiate a BLOCK READ cycle, the master broadcasts the memory address of the data to be transferred and the address modifier across the DTB bus, 801. The master drives the LWORD\* signal low and the IACK\* signal high, 801. As noted previously, a special address modifier indicates to the slave module that the fast transfer protocol will be used to accomplish the BLOCK READ.

The slave modules connected to the DTB receive the address and the address modifier broadcast by the master across the bus and receive LWORD\* low and IACK\* high, 803. Shortly after broadcasting the address and address modifier, 801, the master drives the AS\* signal low, 805. The slave modules receive the AS\* low signal, 807. Each slave individually determines whether it will participate in the data transfer by determining whether the broadcasted address is valid for the slave in question, 809. If the address is not valid, the data transfer does not involve that particular slave and it ignores the remainder of the data transfer cycle.

**SUBSTITUTE SHEET**

-29-

The master drives WRITE\* high to indicate that the transfer cycle about to occur is a READ operation, 811. The slave receives the WRITE\* high signal, 813, and, knowing that the data transfer operation is a READ operation, places the first segment of the data to be transferred on data lines D00 through D31, 819. The master will wait until both DTACK\* and BERR\* are high, 818, which indicates that the previous slave is no longer driving the DTB.

The master then drives DS0\* low, 821, and, after a predetermined interval, drives DS0\* high, 823. The master then awaits a high to low transition on the DTACK\* signal line, 824. As shown in Fig. 8B, the slave then drives the DTACK\* signal low, 825, and, after a predetermined period of time, drives the DTACK\* signal high, 827.

In response to the transition of DTACK\* from high to low, respectively 825 and 827, the master latches the data being transmitted by the slave over data lines D00 through D31, 831. It should be noted that the latching operation is responsive only to the DTACK\* signal. In the fast transfer protocol of present invention, DS0\* is not referenced for purposes of latching data placed on the data lines by the master. The data latched by the master, 831, is written to a device, which has been selected to store the data the device address is incremented, 833.

The slave places the next segment of the data to be transferred on data lines D00 through D31, 829, and then waits for another transition of DS0\* from high to low, 837.

**SUBSTITUTE SHEET**

-30-

To commence the transfer of the next segment of the block of data to be transferred, the master drives DS0\* low, 839, and, after a predetermined period of time, drives DS0\* high, 841. The master then waits for the DTACK\* line to transition from high to low, 843.

The slave drives DTACK\* low, 845, and, after a predetermined period of time, drives DTACK\* high, 847. In response to the transition of DTACK\* from high to low, respectively 839 and 841, the master latches the data being transmitted by the slave over data lines D00 through D31, 845. The data latched by the master, 845, is written to the device selected to store the data, 851 in Fig. 8C, and the device address is incremented. The slave places the next segment of the data to be transferred on data lines D00 through D31, 849.

The transfer of data will continue in the above-described manner until all of the data to be transferred from the slave to the master has been written into the device selected to store the data. After all of the data to be transferred has been written into the storage device, the master will release the address lines, address modifier lines, data lines, the IACK\* line, the LWORD line and DS0\* line, 852. The master will then wait for receipt of a DTACK\* high to low transition, 853. The slave will drive DTACK\* low, 855, and, after a predetermined period of time, drive DTACK\* high, 857. In response to the receipt of the DTACK\* high to low transition, the master will drive AS\* high, 859, and release the AS\* line, 861.

Fig. 9 is a timing diagram illustrating the data transfer timing associated with a fast transfer mode BLOCK WRITE operation.

**SUBSTITUTE SHEET**



-31-

As shown in Fig. 9, the address of the location to which data is to be transferred is broadcast on lines A01 through A31. The address modifier, which would include the fast transfer mode address modifier code, is broadcast by the master on lines AM0 through AM5. After the address and address modifier have been set up on their respective lines, the master drives AS\* low. The WRITE\* line is driven low by the master to indicate, as noted previously, that the operation to follow is a WRITE operation.

Because the DS1\* line is not used during a fast transfer mode operation, the line may or may not be asserted throughout the operation.

After driving the WRITE\* line low, the master broadcasts the first segment of the data to be transferred on lines D00 through D31.

DS0\* is driven low and the signal subsequently deasserted by being driven high, as shown in Fig. 9. The data being broadcast by the master as DS0\* is driven low and is latched by the slave, in response to the DS0\* signal being driven low. After DS0\* is driven low, the master broadcasts the next segment of data to be transferred to the slave over lines D00 through D31, as shown in Fig. 9. The slave, in response to DS0\* being driven low, subsequently acknowledges the data transfer by driving DTACK\* low for a given period and then deasserting the signal by driving the DTACK\* line high. As shown in Fig. 9, DS0\* is not reasserted until the slave acknowledges the data transfer by driving the DTACK\* line low.

As noted previously, the data transfer cycles will continue until all of the data to be transferred has been broadcast to the slave. The number of cycles required to complete the transfer

**SUBSTITUTE SHEET**

-32-

would occur at box 900 in Fig. 9. Box 900 is merely exemplary and not drawn to a particular time scale.

Fig. 9A illustrates a data transfer cycle which could be inserted in the location of box 900 in Fig. 9. As shown in Fig. 9A,  $DS0^*$  is driven low. In response to the  $DS0^*$  low signal, the slave latches the data broadcast at the time  $DS0^*$  went low. The master broadcasts the next segment of the data to be transferred. The slave, acknowledging the data transfer, drives  $DTACK^*$  low. This operation would continue until all data has been transferred.

Referring again to Fig. 9, after the data transfer operation has been completed, the slave drives  $DTACK^*$  low. In response, the master deasserts  $AS^*$  by driving the  $AS^*$  line high. The master likewise releases the  $WRITE^*$  line by driving the line high.

The duration of the respective  $DS0^*$  and  $DTACK^*$  signals can vary depending upon the application and the system being used. Likewise, the period between the assertion of  $DS0^*$  and the assertion of  $DTACK^*$  may also vary depending upon the application and the system being used. Obviously, the data transfer rate will be increased if the duration of the  $DS0^*$  and  $DTACK^*$  signals and the period between the assertion of  $DS0^*$  and  $DTACK^*$  are minimized.

Fig. 10 is a timing diagram illustrating the data transfer timing associated with a fast transfer mode BLOCK READ operation.

As shown in Fig. 10, the address of the location to which data is to be transferred is broadcast on lines A01 through A31. The address modifier, which would include the fast transfer mode address modifier code, is broadcast by the master on line AM0 through AM5. After the address and address

-33-

modifier have been set up on their respective lines, the master drives AS\* low. The WRITE\* line is driven high by the master to indicate, as noted previously, that the operation to follow is a READ operation.

5       Because the DS1\* line is not used during a fast transfer mode operation, the line remains high throughout the entire operation.

10       In response to the WRITE\* line being driven high, data is broadcast by the slave on lines D00 through D31.

15       DS0\* is driven low and the signal subsequently deasserted by being driven high, as shown in Fig. 10. The slave, in response to DS0\* being driven low, subsequently acknowledges the data transfer by driving DTACK\* low for a given period and then deasserting the signal by driving the DTACK\* line high. The data being broadcast by the slave as DTACK\* is driven low is latched by the master, in response to the DTACK\* signal being driven low. After DTACK\* is driven low, the slave broadcasts the next segment of data to be transferred to the master over lines D00 through D31, as shown in Fig. 10. DS0\* is not reasserted until the slave acknowledges the data transfer by driving the DTACK\* line low.

25       As noted previously, the data transfer cycles will continue until all of the data to be transferred has been broadcast to the master. The number of cycles required to complete the transfer would occur at box 1000 in Fig. 10. Box 1000 is merely exemplary and not drawn to a particular time scale. Fig. 10A illustrates a data transfer cycle which could be inserted in the location of box 1000 in Fig. 10. As shown in Fig. 10A, DS0\* is driven low. In response to the DS0\* low signal, the slave acknowledges the data transfer by driving DTACK\* low.

35

-34-

In response to the DTACK\* low signal, the master latches the data broadcast at the time DTACK\* went low. The slave broadcasts the next segment of the data to be transferred. This operation would  
5 continue until all data has been transferred.

Referring again to Fig. 10, after the data transfer operation has been completed the slave drives DTACK\* low. In response, the master deasserts AS\* by driving the AS\* line high. The master  
10 likewise releases the WRITE\* line.

As already noted with regard to the WRITE operation, the duration of the respective DS0\* and DTACK\* signals can vary depending upon the application and the system being used. Likewise,  
15 the period between the assertion of DS0\* and the assertion of DTACK\* may also vary depending upon the application and the system being used. Obviously, the data transfer rate will be increased if the duration of the DS0\* and DTACK\* signals and the  
20 period between the assertion of DS0\* and DTACK\* are minimized.

Because the fast transfer protocol requires that data signals remain on the DTB for a very short period of time, the amount of skew between the  
25 control signals and the data signals must be minimized. For example, the DTACK\* signal, which references data transfer on a BLOCK READ cycle, must go from high to low in the shortest possible interval so as to enable the master the maximum  
30 amount of time, under the circumstances, to capture the data to be transferred.

To implement the fast transfer protocol, a conventional 64 mA tri-state driver 245 is substituted for the 48 mA open collector driver  
35 conventionally used in the slave module to drive

**SUBSTITUTE SHEET**

-35-

DTACK\* as shown in Fig. 11. This substitution is necessary because the 48 mA open collector DTACK\* driver does not drive DTACK\* from high to low quickly enough for purposes of the present invention. Implementation of the 64 mA tri-state driver 245 provides a means of quickly changing the state of the DTACK\* signal so as to reduce skew between the DTACK\* and data signals sufficient for purposes of the present invention.

It should likewise be noted that the data drivers on master and slave modules have been modified. To implement fast transfer protocol in the preferred embodiment of the present invention, the conventional VMEbus data drivers should be replaced with 64 mA tri-state drivers in SO-type packages. This 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\*.

Further, in order to maximize performance, signal return inductance along the bus backplane should be reduced to a level permitting the short signal activation times necessary to implement the fast transfer mode protocol. In the preferred embodiment of the present invention, signal return inductance is reduced 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**

-36-

IV. Bus Locking FIFO Message Passing Protocol

While the enhanced fast transfer protocol provides for the expedient block transfer of message data, the present invention further and in combination provides for a similarly expedient delivery of message descriptors between the processors. These message descriptors are used, in the preferred embodiments of the present invention, to deliver the information necessary for the processor that receives a message descriptor to locate a data block and initiate an enhanced fast transfer protocol block transfer.

Figure 12 is a flowchart illustrating a message descriptor transfer operation accomplished under the present invention. A sender processor (hereinafter "sender"), not shown in Figure 12, may initiate a message descriptor transfer, 1010, by initiating a WRITE cycle, as described above, to a FIFO, not shown in Figure 12, associated with a recipient processor (hereinafter "recipient") over the VMEbus. Message descriptors may contain data of any type. Typically, the message descriptor consists of a 128 byte wide data packet. In the preferred embodiments of the present invention, a message descriptor contains a shared memory address of a data block that is to be transferred to the message descriptor recipient processor. This recipient processor will act as the master processor in initiating an enhanced fast transfer protocol block transfer of the data block from the shared memory address provided by the message descriptor to a memory location of the recipient processor's own choosing.

In the transfer of a message descriptor from a sender to a receiver, if the recipient processor's FIFO is not FULL and a message descriptor WRITE

**SUBSTITUTE SHEET**

-37-

operation is successful, the recipient's control logic will acknowledge the message descriptor transfer by transmitting a DTACK\* signal to the sender, 1030.

5        If the FIFO being written to by the sender is full, 1020, the recipient's control logic will drive BERR\* low, indicating a BUS ERROR, i.e., a bus-lock on the message descriptor write operation. This informs the sender that the WRITE operation was  
10        unsuccessful. The sender will then wait for a designated period of time, 1050, before attempting to retry the WRITE operation. The waiting time is determined by the application. After waiting the designated interval, 1050, the sender will retry the  
15        WRITE operation, 1010. This process will continue until the message descriptor transfer operation is successful (i.e., the sender receives DTACK\* rather than BERR\*).

20        Figure 13 is a block diagram of the bus locking multi-processor communication system. A processor 1101 and a processor 1103 are each interconnected to VMEbus 22. It should be understood that either of the processors 1101 and 1103 may send or, alternatively, receive message descriptors across  
25        VMEbus 22.

      A message descriptor is transmitted by a sender, 1101 for example, across VMEbus 22 to FIFO 1120 of a recipient, 1103 for example. To initiate a transfer, the sender's microprocessor 1140  
30        broadcasts, one word at a time typically the descriptor and its address across the VMEbus 22. The address broadcast by the sender corresponds to the bus address of recipient's FIFO. If the FIFO 1120 is either empty or not full, the message  
35        descriptor is received by and stored in FIFO 1120.

**SUBSTITUTE SHEET**

-38-

The message descriptor can then be read from FIFO 1120 by the microprocessor 1140 when it is ready to process the next message descriptor.

If a recipient FIFO 1120 is full, the FIFO 1120 transmits a FIFO FULL signal along the FIFO FULL line 1125 to the recipient processor's control logic 1130. In response to receiving a FIFO FULL signal, the recipient's control logic 1130 generates a BERR\* signal by driving BERR\* line 1127 low,

Figure 14 is a block diagram of the preferred embodiment of one of the message transfer units of the bus-locking multi-processor communication system of the present invention. A message descriptor is transmitted by the sender processor, not shown in Figure 14, across VMEbus 22 and is received at the recipient processor by a data receiver 1210.

Similarly, the address and address modifier, also transmitted by the sender processor across VMEbus 22, are received by address receiver 1240 and forwarded to an address detect circuit 1250. The address detect circuit detects the address and address modifier and enables the control logic 1130. The address and address modifier transmitted by the sender processor, not shown in Figure 14, is supplied to the control logic 1130 by lines 1247 and 1245 respectively. The transmitted address designates the FIFO 1120 as the intended recipient.

FIFO 1120 is a conventional FIFO having at least three data storage states: a "FIFO full" state, indicating that all FIFO storage locations are full and cannot store any further data, a "FIFO not full" state, indicating that some, but not all, of the memory storage locations are available for the storage of a message descriptor, and a "FIFO empty"



-39-

state, which indicates that all memory storage locations are available for the storage of message descriptors.

5 The FIFO 1120 has a FIFO FULL signal output line 1125 interconnected to the control logic 1130. The FIFO FULL line 1125 is activated by the FIFO 1120 when a "FIFO full" state exists and a sender processor is attempting to write descriptor data into a full FIFO 1120. A FIFO EMPTY signal line 10 1235 is interconnected to the control microprocessor 1220 and signals the microprocessor 1220 in the event FIFO 1120 is empty. A FIFO NOT EMPTY signal line 1236 is interconnected to control microprocessor 1220 to signal the control 15 microprocessor 1220 that a message descriptor is resident in some, but not all, of the FIFO's 1120 storage locations.

Data input lines 1215 interconnect the data receiver 1210 to the FIFO 1120. Data is output from 20 the FIFO 1120 to the control microprocessor 1220 across data lines 1233. Two control lines, FIFO WRITE 1231 interconnected to the control logic 1130 and FIFO READ 1237 interconnected to the microprocessor 1220, control the flow of data to and 25 from the FIFO 1120.

WRITE\* signal line 1127 interconnects microprocessor 1220 to control logic 1130 and is driven low in the event the microprocessor 1220 desires to initiate a data WRITE operation. Data 30 lines 1129 interconnect microprocessor 1220 to data transceiver 1212. Similarly, address lines 1123 and address modifier lines 1122 interconnect microprocessor 1220 and address transceiver 1213. Data transceiver 1212 transmits and receives data 35 directed, respectively, to and from VMEbus 22.

-40-

Likewise, address transceiver 1213 transmits and receives address and address modifier signals directed, respectively, to and from VMEbus 22.

Control logic 1130 is interconnected to VMEbus 22  
5 by WRITE\* line 1250, BERR\* line 1262, DTACK\* line 1264 and DSO\* 1266. The WRITE\* line 1250, DTACK\* line 1264 and DSO\* line operate in the manner described above. If the WRITE operation is successfully accomplished, the recipient processor  
10 acknowledges the transfer by having control logic 1130 drive DTACK\* low.

If the sender's processor, not shown in Figure 14, is attempting to write a message descriptor into the FIFO 1120 when a FIFO full state condition  
15 exists, the FIFO activates FIFO FULL line 1125. Upon receipt of the FIFO FULL signal across FIFO FULL line 1125, control logic 1130 drives BERR\* line 1250 low indicating to the sender that the immediate WRITE operation was unsuccessful. The sender may,  
20 upon receipt of the BERR\* signal, choose to retransmit the descriptor to the recipient.

If the FIFO is not full and the WRITE operation was successful, the message descriptor is stored in the FIFO 1120 for subsequent access and use by the  
25 recipient microprocessor 1220. Successful message descriptor transfers conclude with a DTACK\* signal generated by the recipient processor's control logic 1130.

The processor system shown in FIGURE 14 can also  
30 be used to transmit message descriptors. To transmit message descriptors to a recipient processor, not shown in FIGURE 14, along VMEbus 22, the microprocessor 1220 compiles the message data to be transferred in a "message buffer". The address  
35 of the intended recipient FIFO and address modifier

-41-

are transmitted to the address transceiver 1213.  
The microprocessor 1220 drives WRITE\* line 1127 low.  
In response, control logic 1130 drives the WRITE\*  
line 1260 low to indicate that the processor is  
5 instituting a WRITE operation. A message  
descriptor, which is a pointer to the "message  
buffer", is transmitted by microprocessor 1220 to  
the data transceiver 1212. The data, address, and  
address modifier are broadcast by respective  
10 transceivers across VMEbus 22. As noted above, if  
the addressed FIFO is full, the processor will  
receive a BERR\* signal, indicating the WRITE  
operation was unsuccessful.

The foregoing description of the present  
15 invention merely sets forth the preferred  
embodiment. Modifications and variations of the  
invention as described and set forth above may be  
made which are consistent with the scope and spirit  
of the present invention and the appended claims.  
20 Other aspects, objects and advantages of this  
invention can be obtained from a study of the  
drawing, the disclosure and the appended claims.

-42-

CLAIMS

What is claimed is:

1. A message transfer system for transferring  
5 message data from a master processor across a VMEbus  
to a slave processor, the apparatus comprising:

FIFO means interconnected to the VMEbus for  
receiving and storing the message data transferred  
from the master processor, said FIFO means having a  
10 FIFO FULL state indicative that said FIFO means is  
unable to store message data, said FIFO means  
generating a FIFO FULL signal to indicate the  
existence of said FIFO FULL state; and

means interconnected to said FIFO means and the  
15 VMEbus for transmitting a BUS ERROR signal across  
the VMEbus responsive to the receipt of a FIFO FULL  
signal from said FIFO means.

2. A message transfer system for transferring  
message data from a master processor across a VMEbus  
20 to a slave processor, the apparatus comprising:

a message data channel means for receiving  
message data transmitted from the master  
processor, said message data channel means  
interconnected to the VMEbus;

25 a FIFO means for storing message data  
interconnected to said message channel means,  
said FIFO means having a FIFO FULL state  
indicative that said FIFO means is unable to  
store message data, said FIFO means generating a  
30 FIFO FULL signal to indicate the existence of  
said FIFO FULL state; and

means for transmitting a BUS ERROR signal  
across the VMEbus responsive to the receipt of a  
FIFO FULL signal from said FIFO means.

35 3. A method for terminating the transfer of message

-43-

data across a VMEbus from a master processor to a slave processor in the event that the slave processor is unable to store the message data to be transferred, the method comprising the steps of:

5           initiating a WRITE cycle so as to write the message data across the VMEbus to a FIFO means interconnected to a slave processor; and

10           transmitting a BUS ERROR signal in response to the initiation of the WRITE cycle to indicate that the FIFO means is unable to capture the message data transferred.

4. The method of Claim 3 wherein the method comprises the additional step of reinitiating the WRITE cycle after predetermined amount of time has  
15           lapsed.

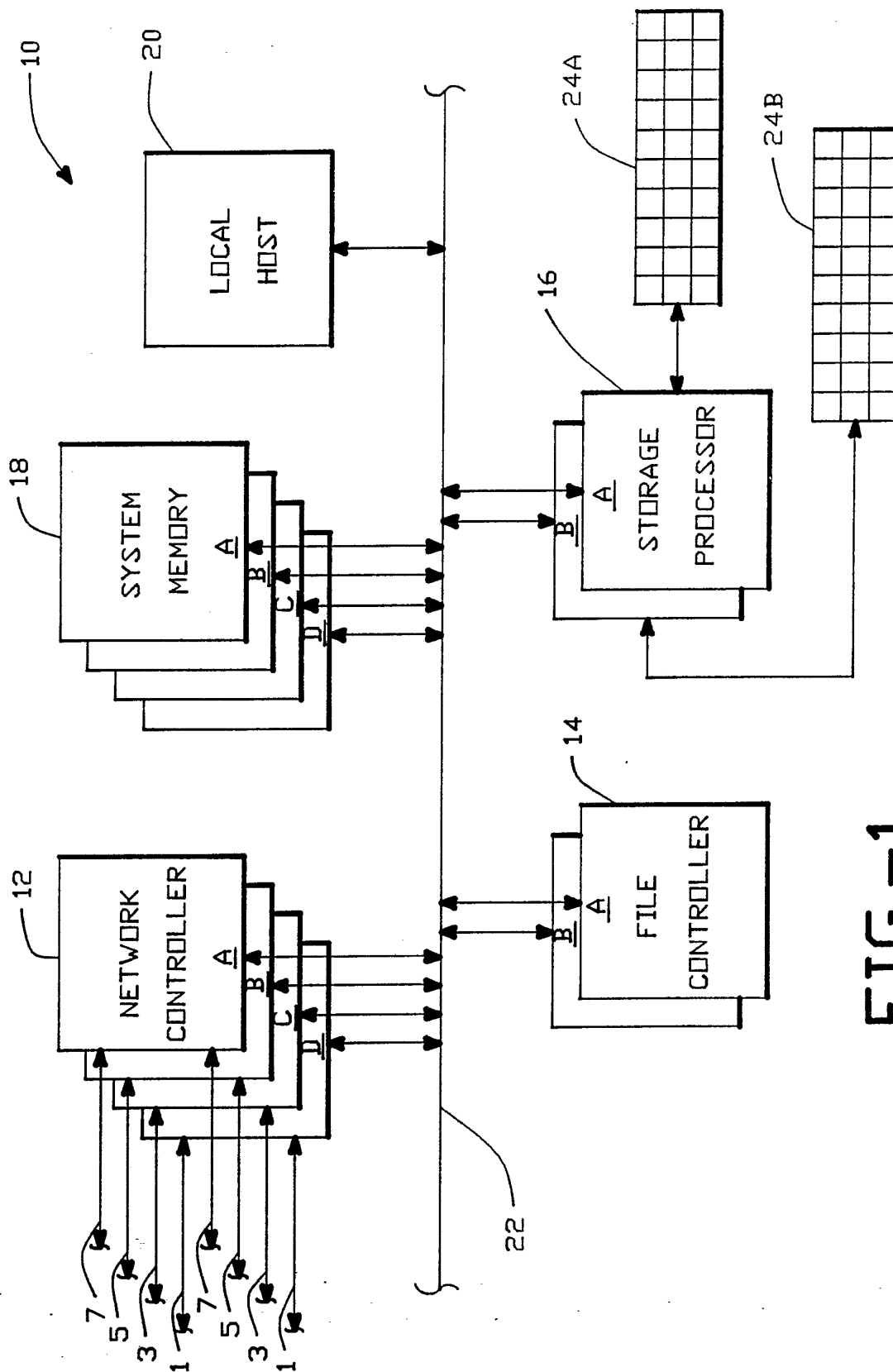


FIG.-1

2/19

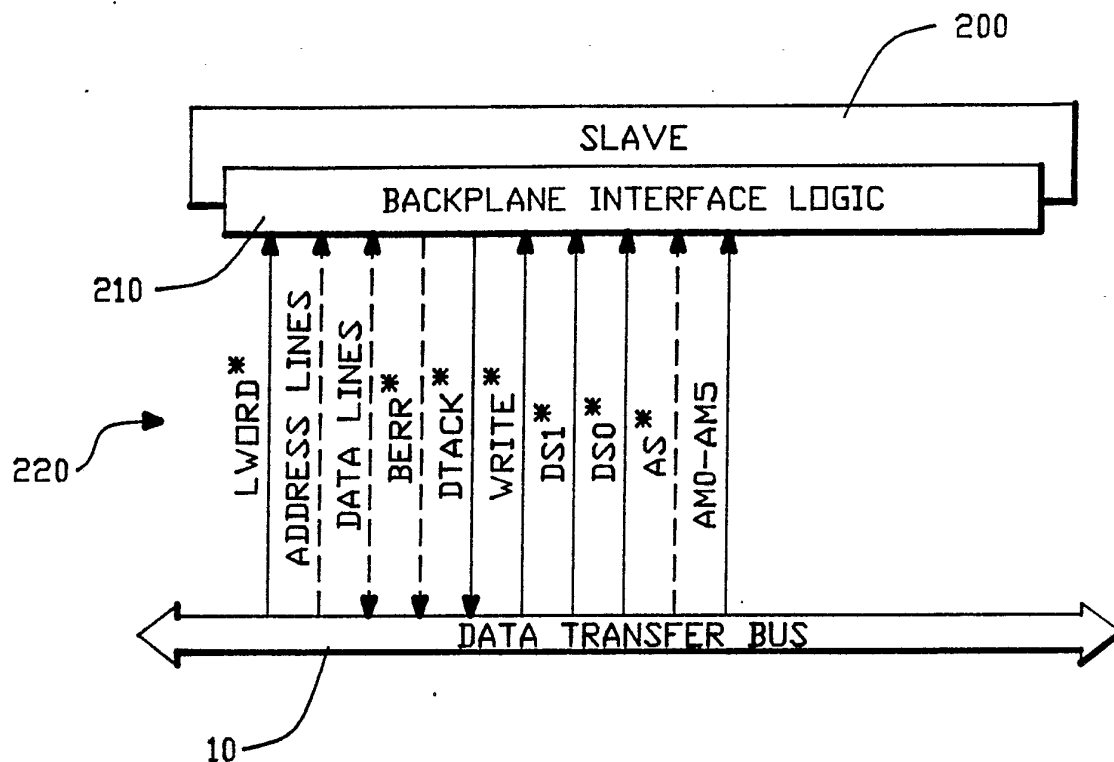


FIG.-2

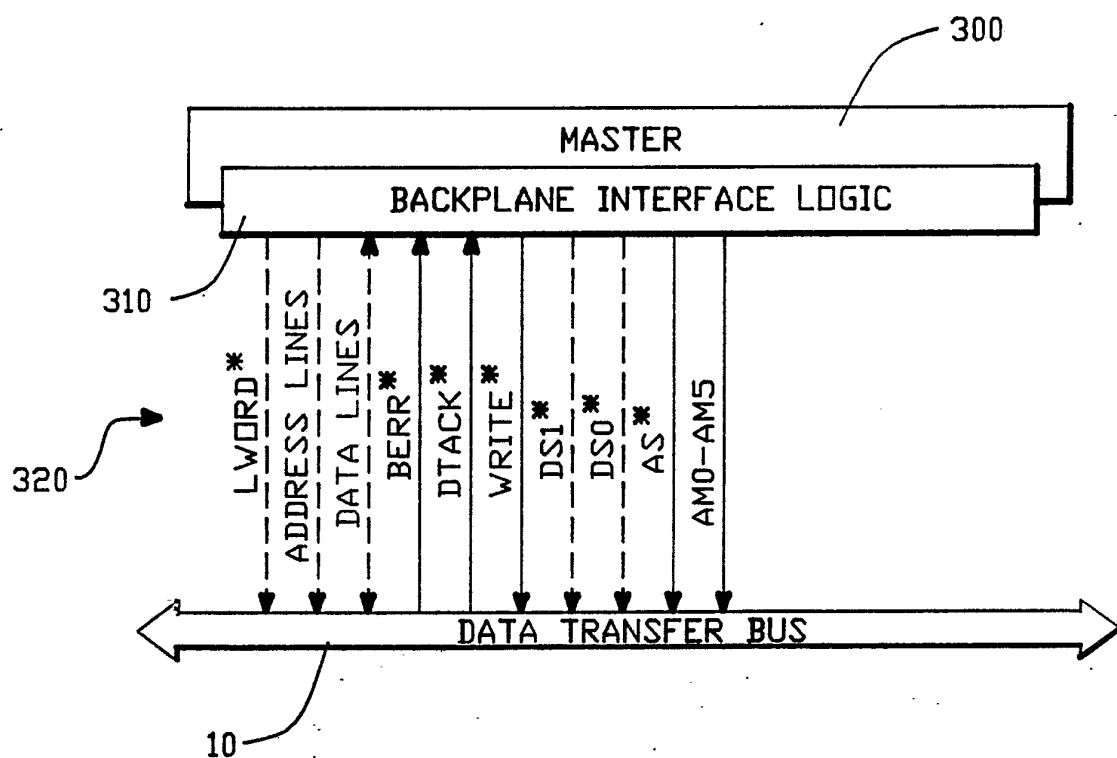


FIG.-3

SUBSTITUTE SHEET

3/19

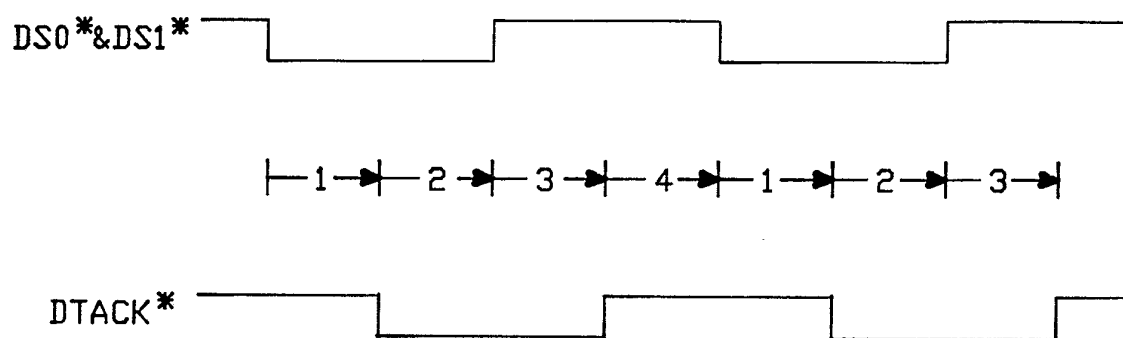


FIG.-4A

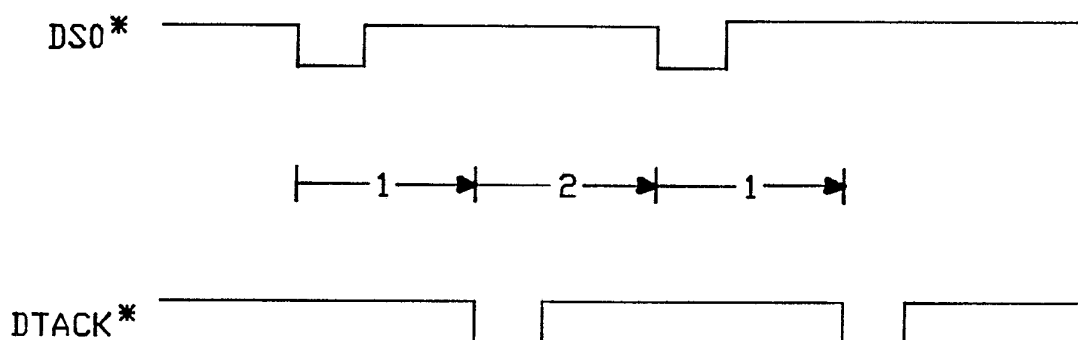


FIG.-4B

SUBSTITUTE SHEET



4/19

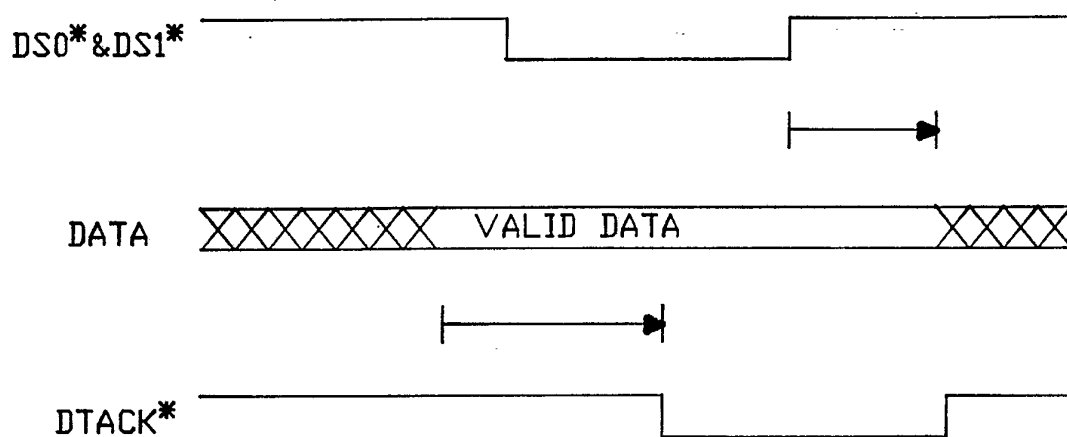


FIG.-5A

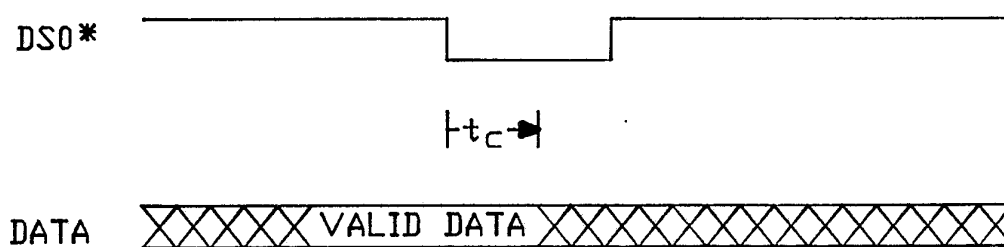


FIG.-5B

SUBSTITUTE SHEET

5/19

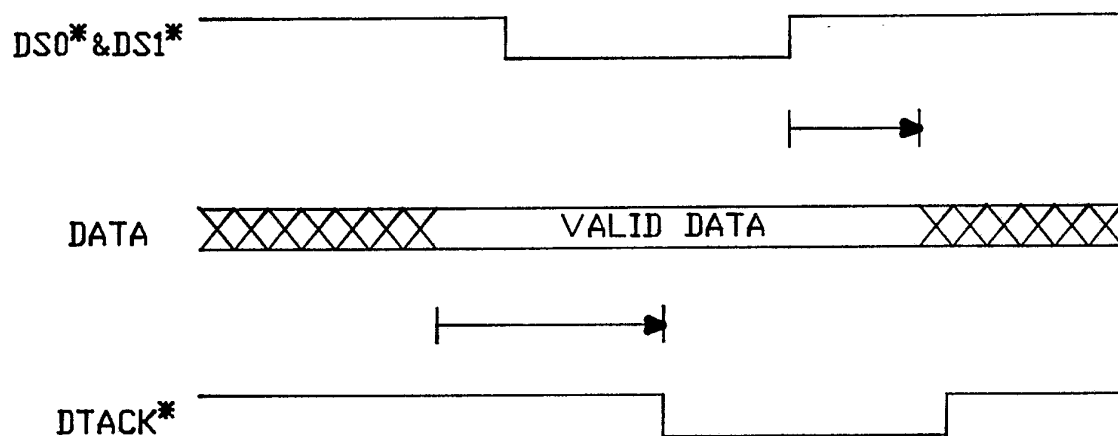


FIG.-6A

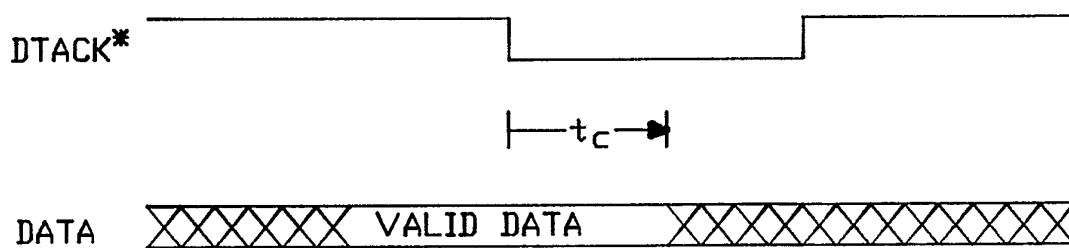


FIG.-6B

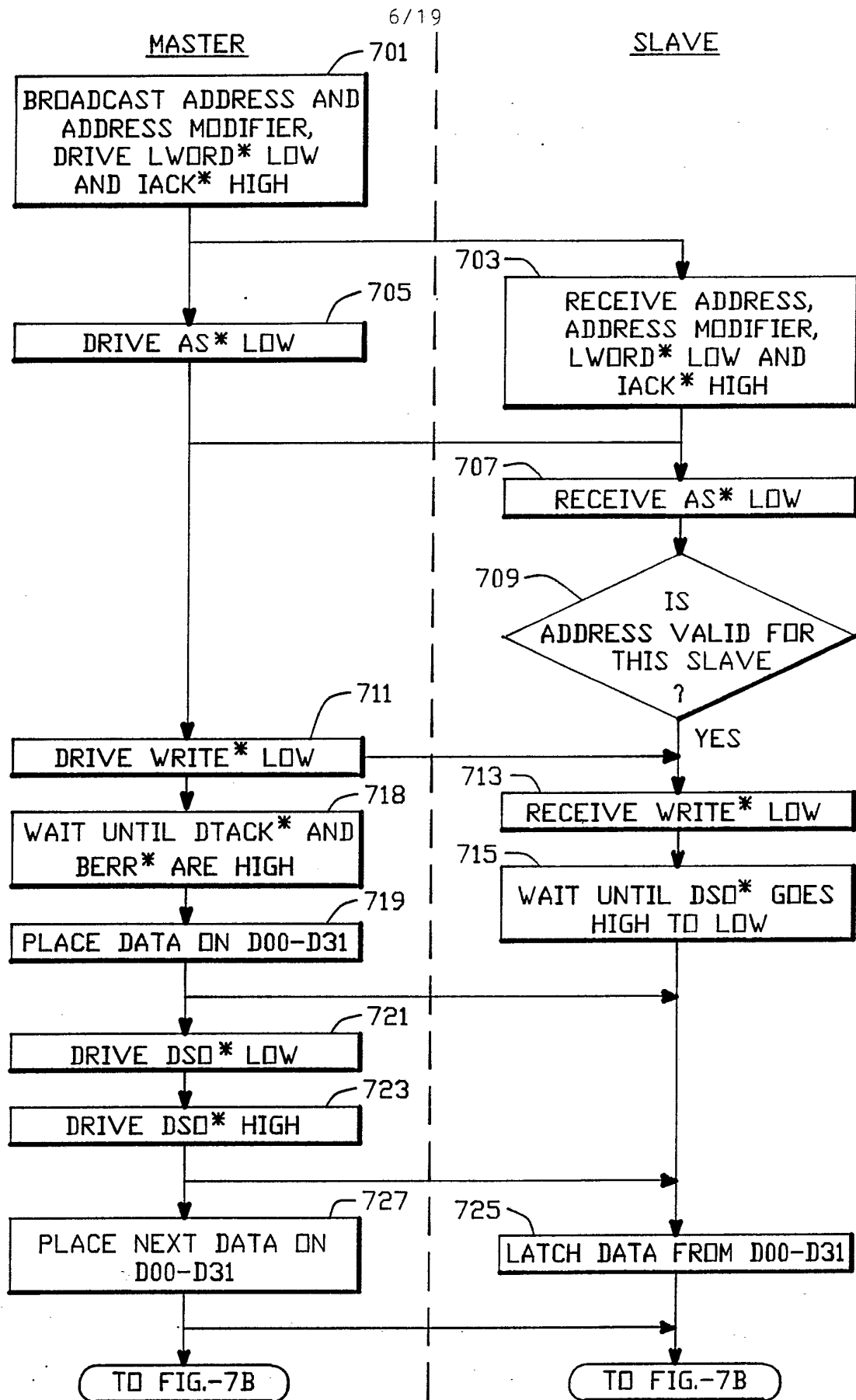


FIG.-7A

SUBSTITUTE SHEET

7/19

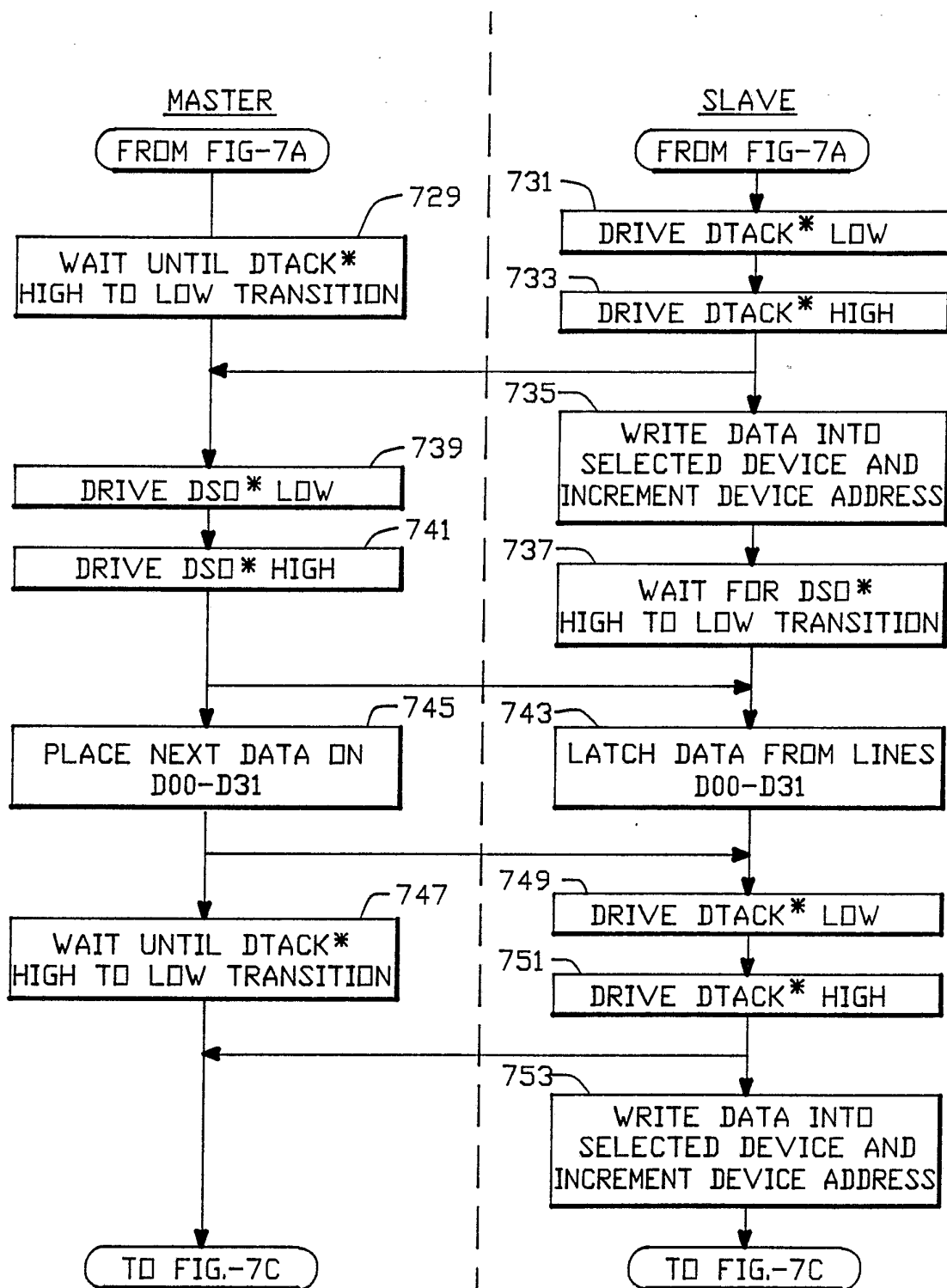


FIG.-7B

8/19

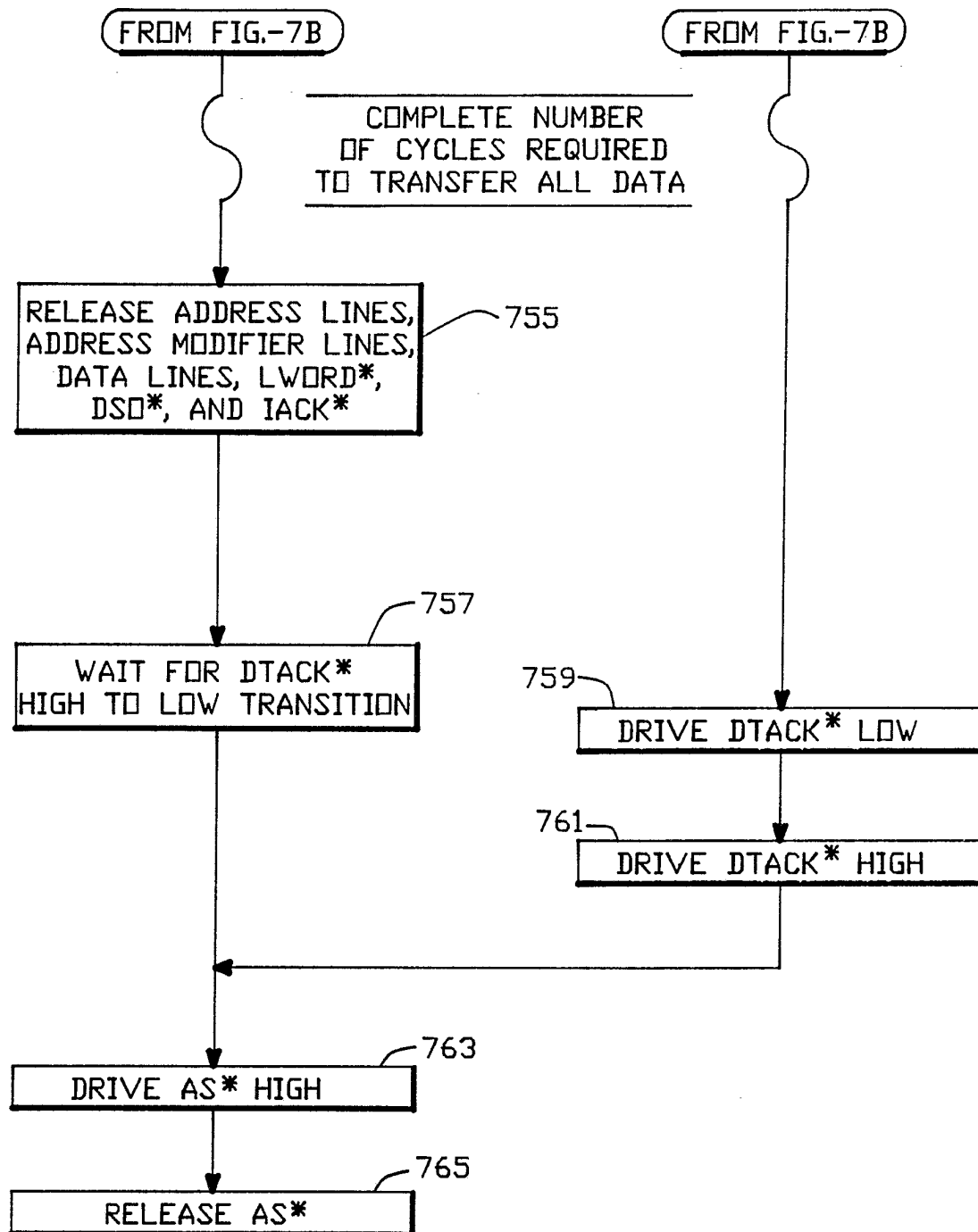


FIG.-7C

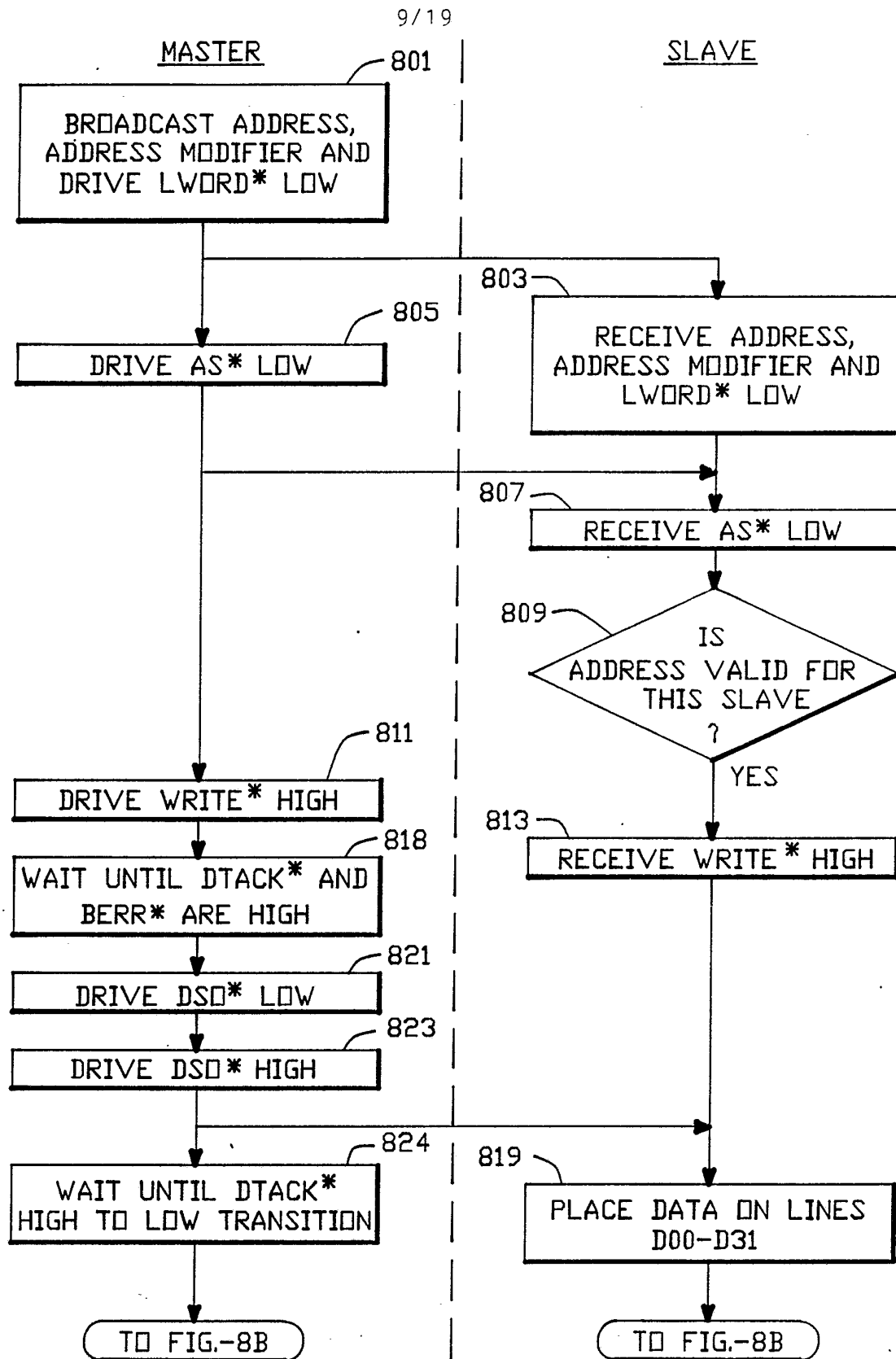


FIG.-8A

10/19

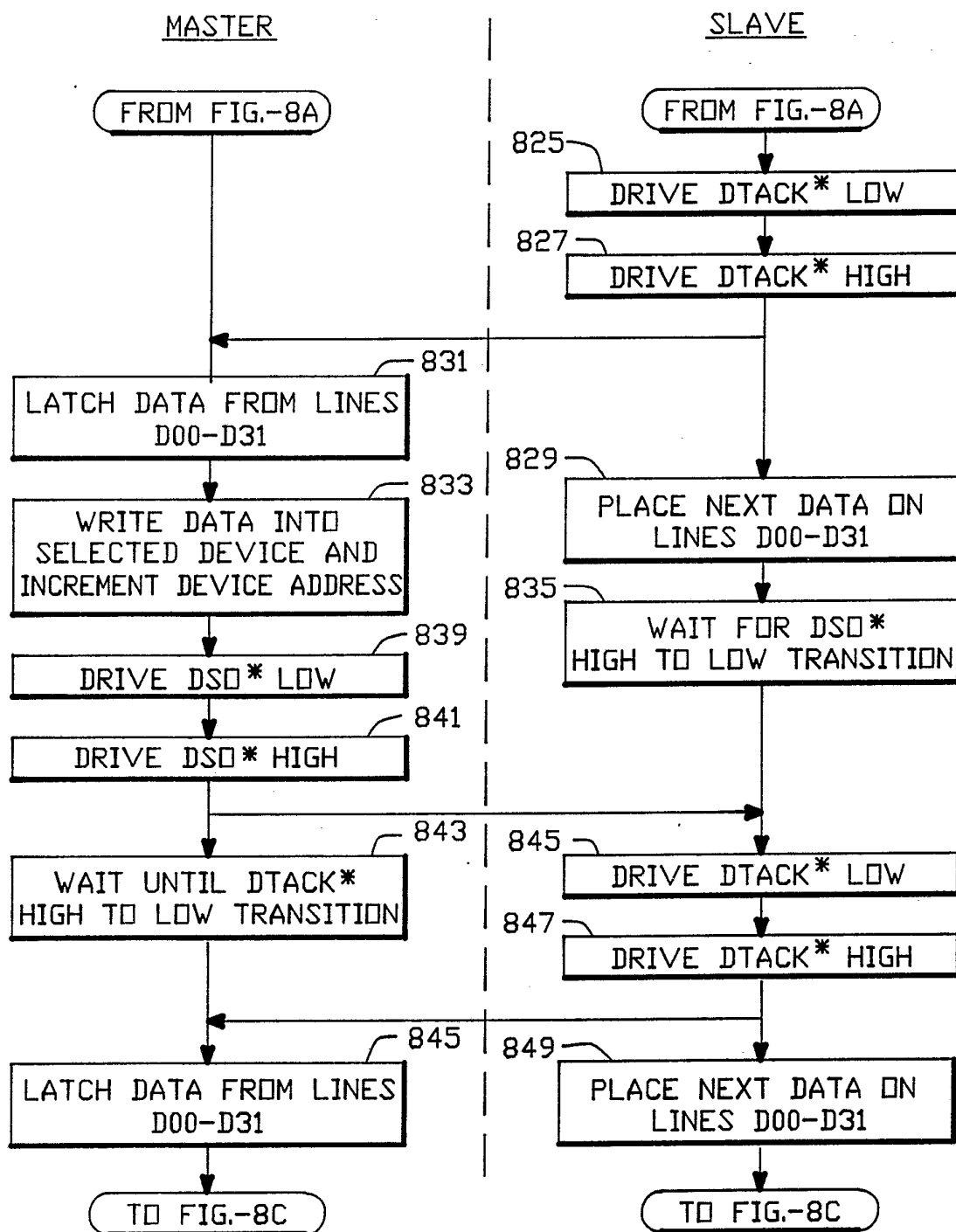


FIG.-8B

11/19

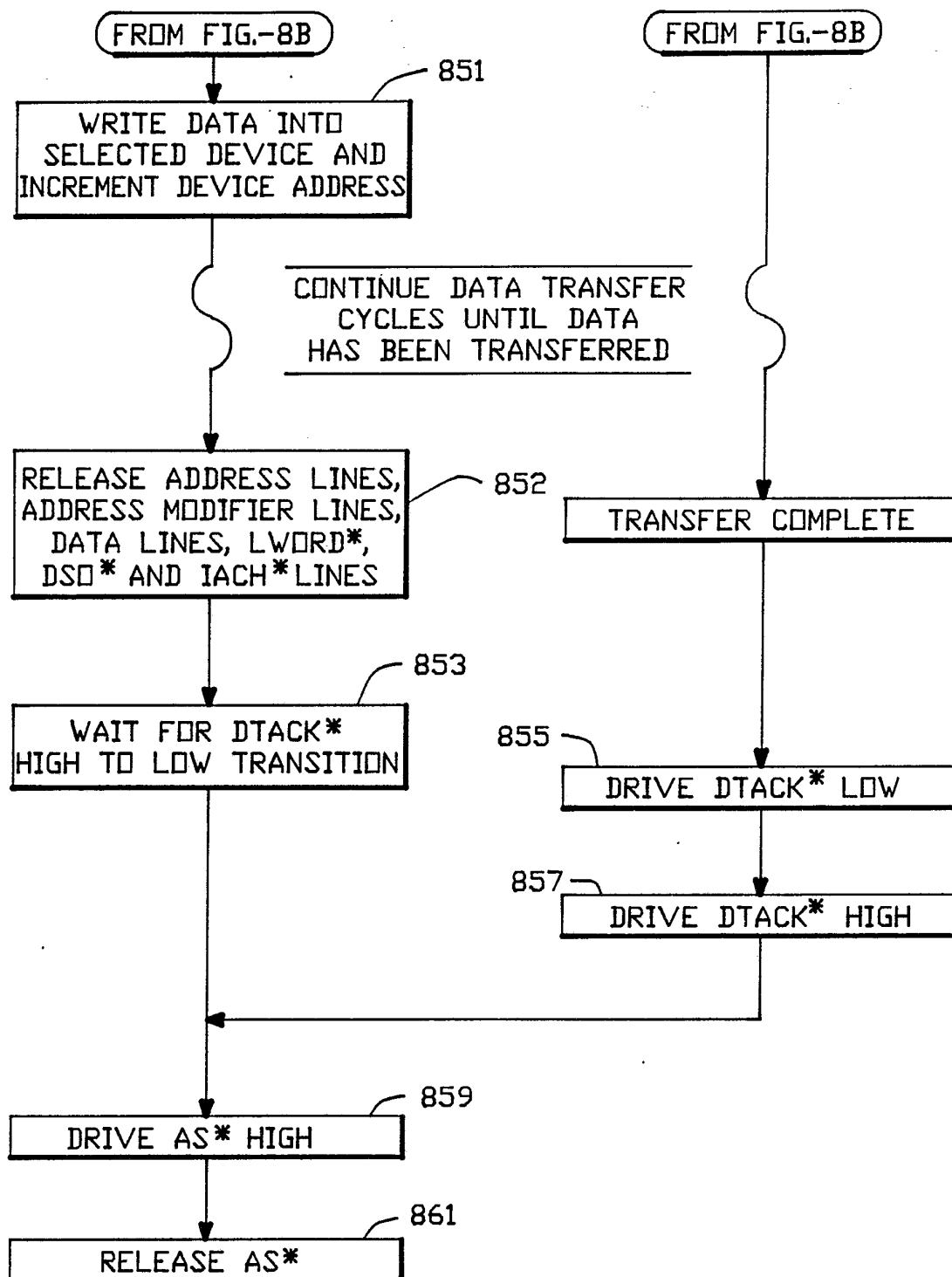


FIG.-8C



12/19

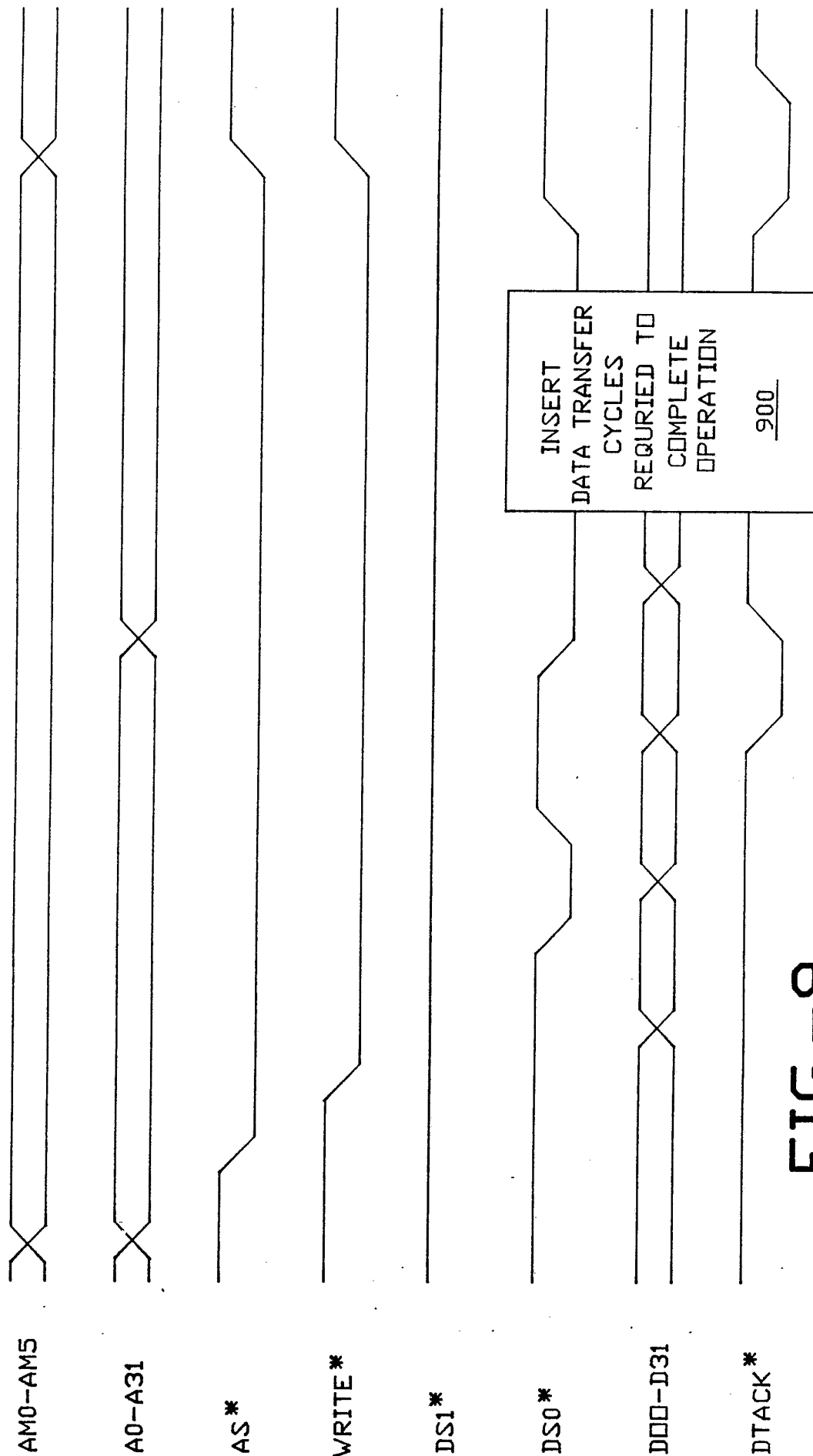
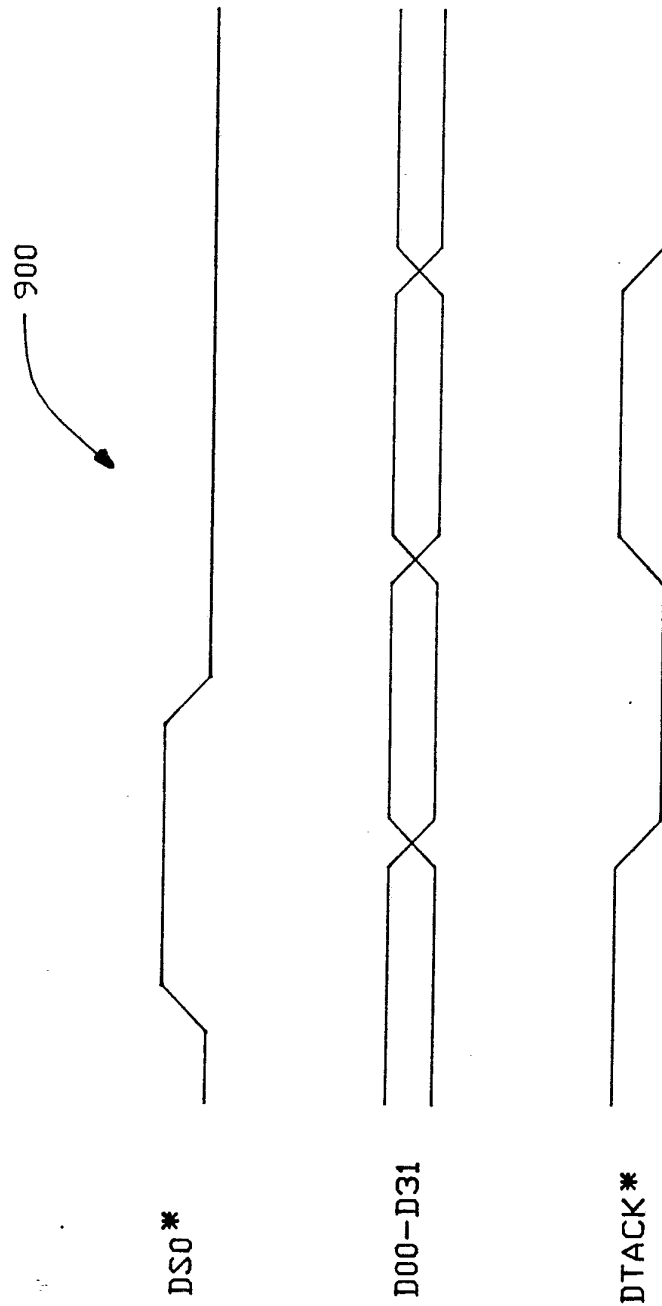


FIG.-9



**FIG.-9A**

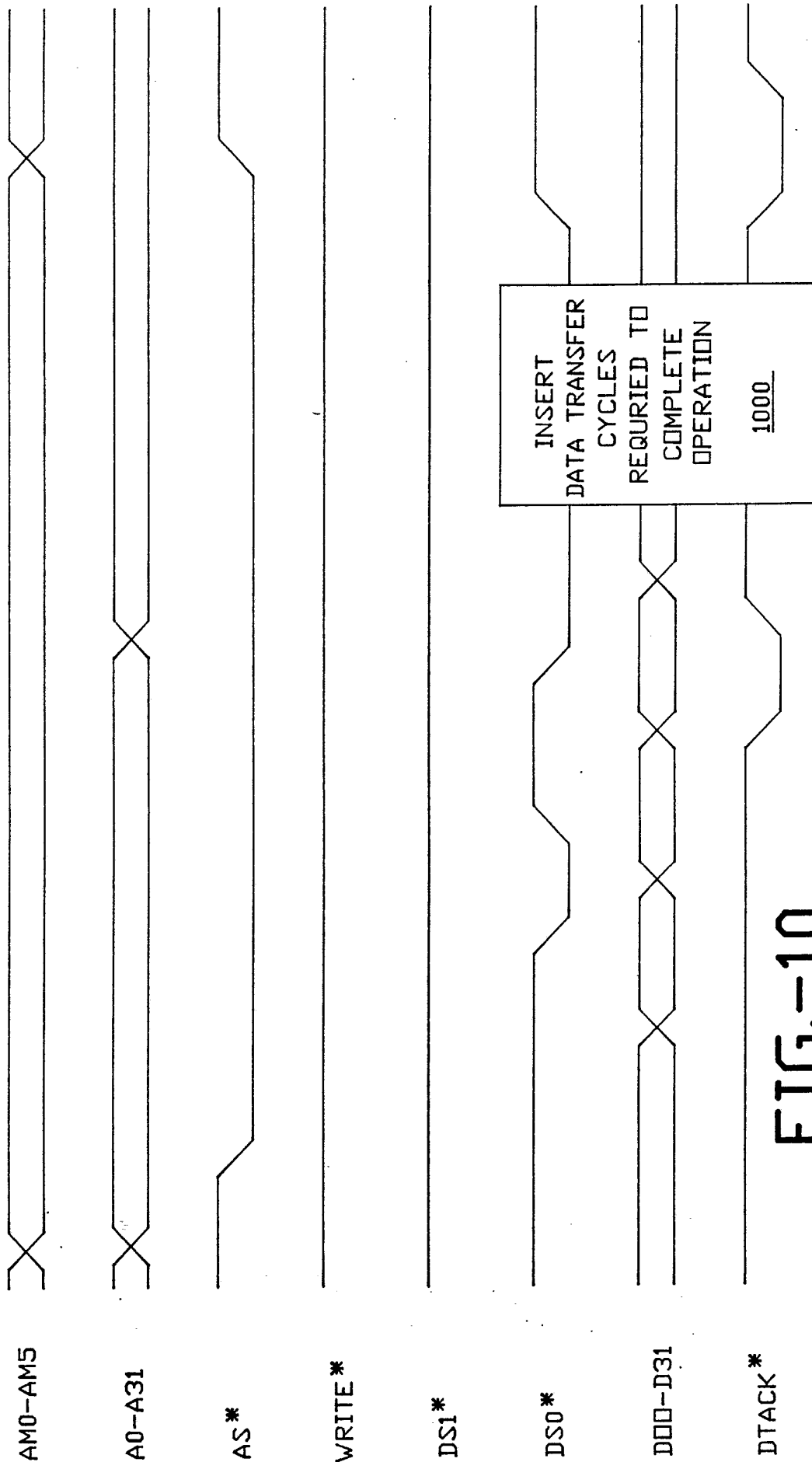
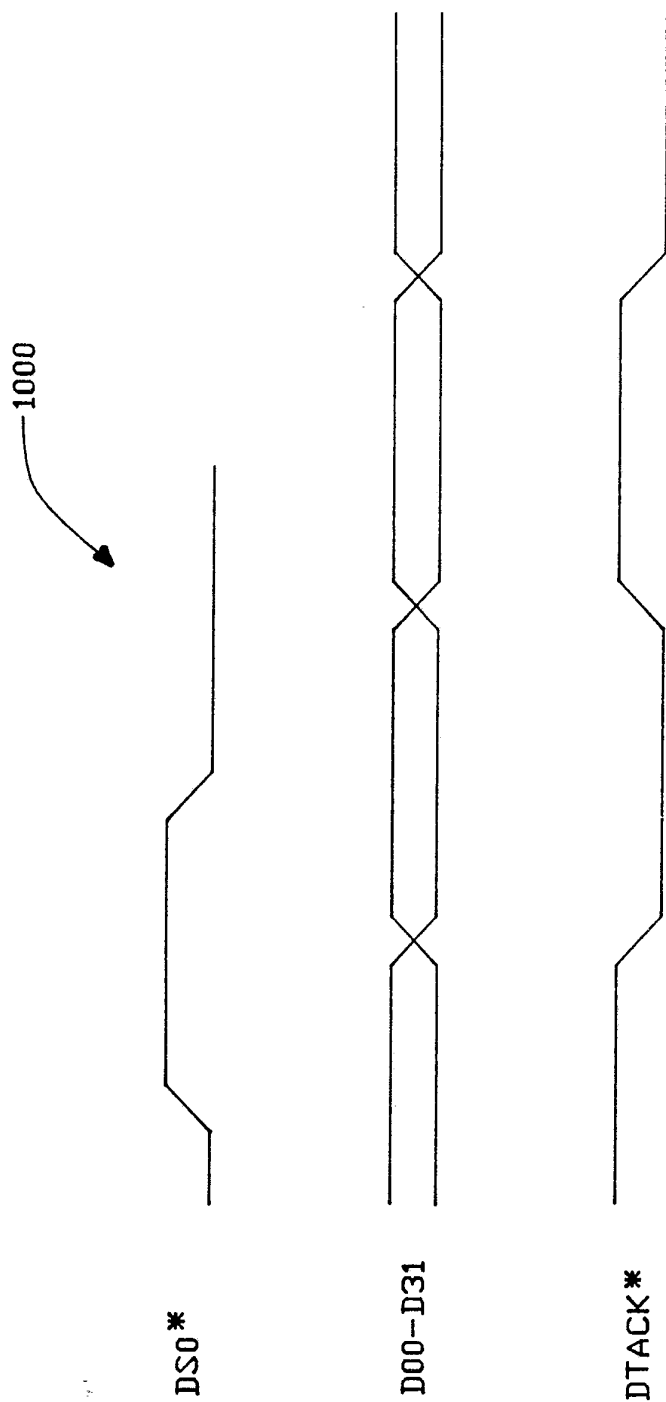


FIG.-10



**FIG.-10A**

16/19

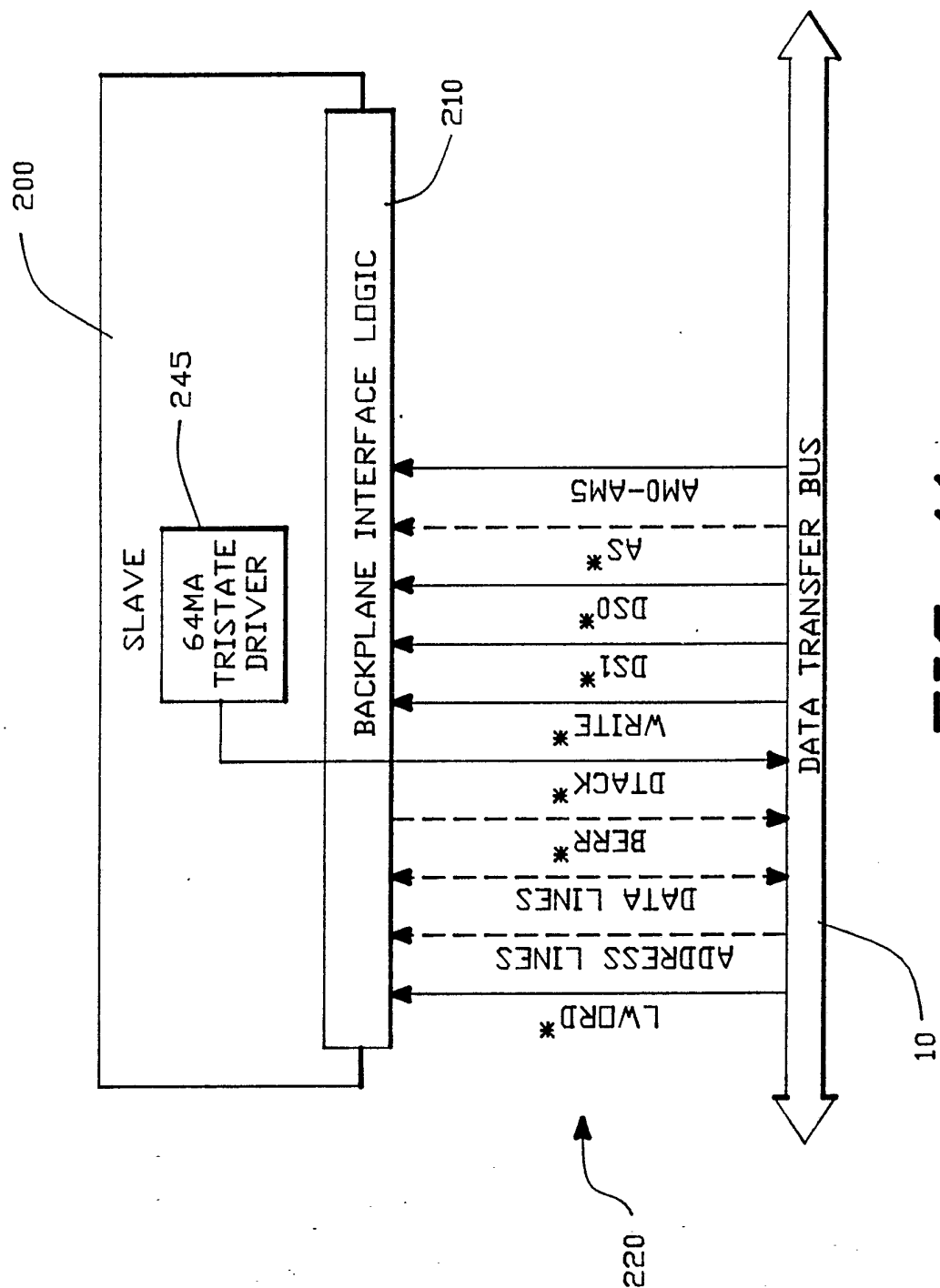


FIG.-11

17/19

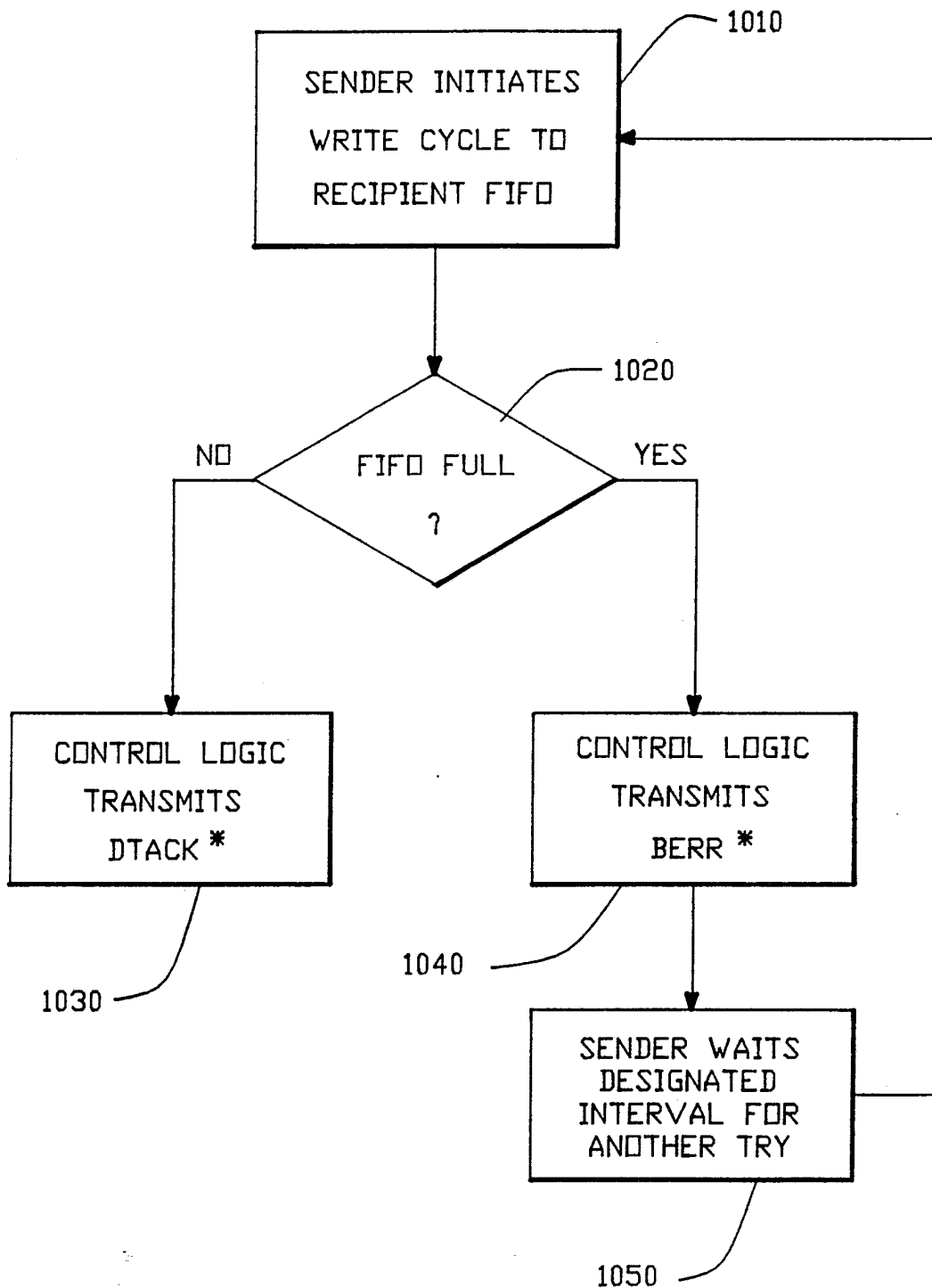


FIG.-12

SUBSTITUTE SHEET

18/19

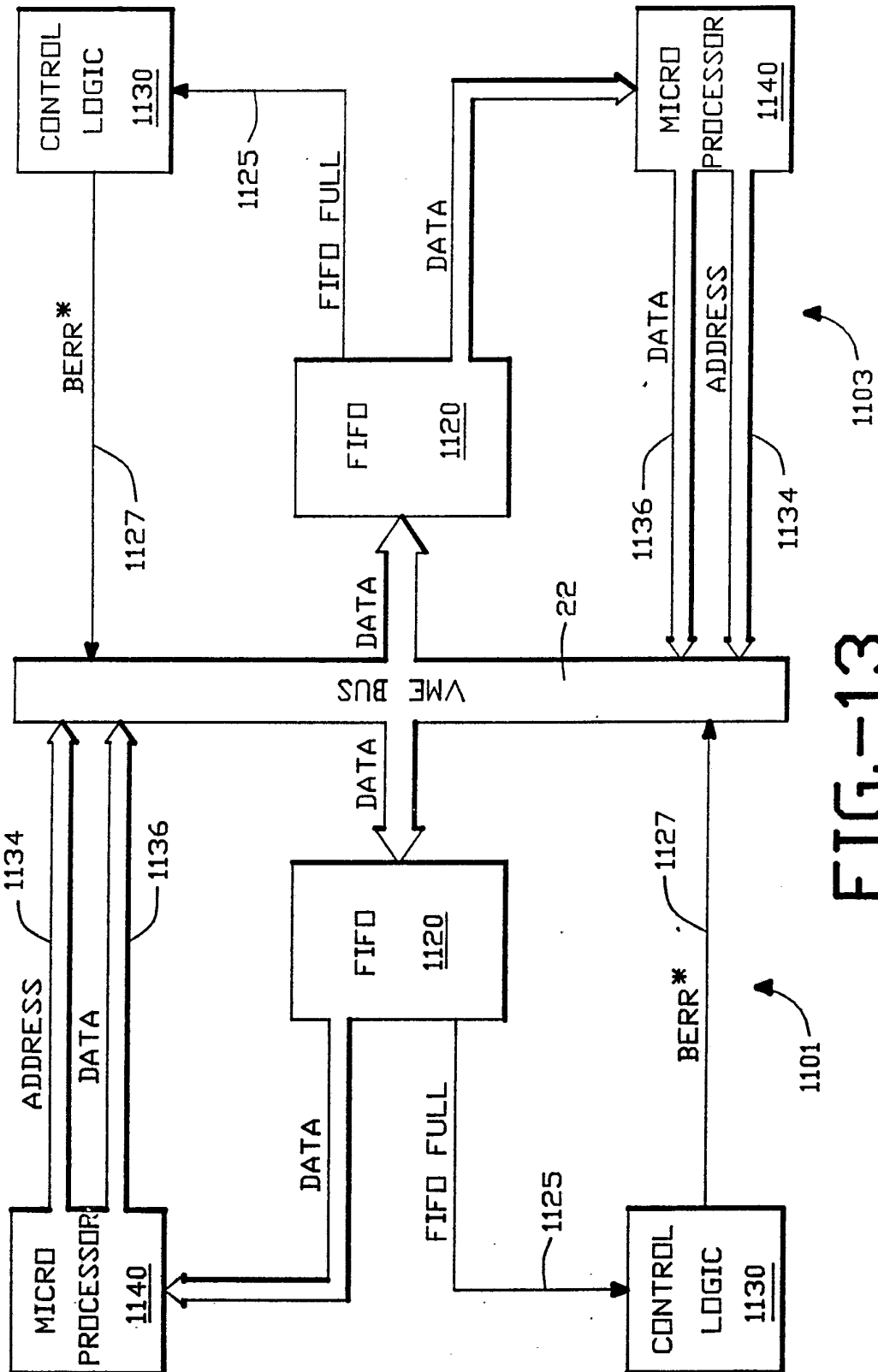


FIG.-13

19/19

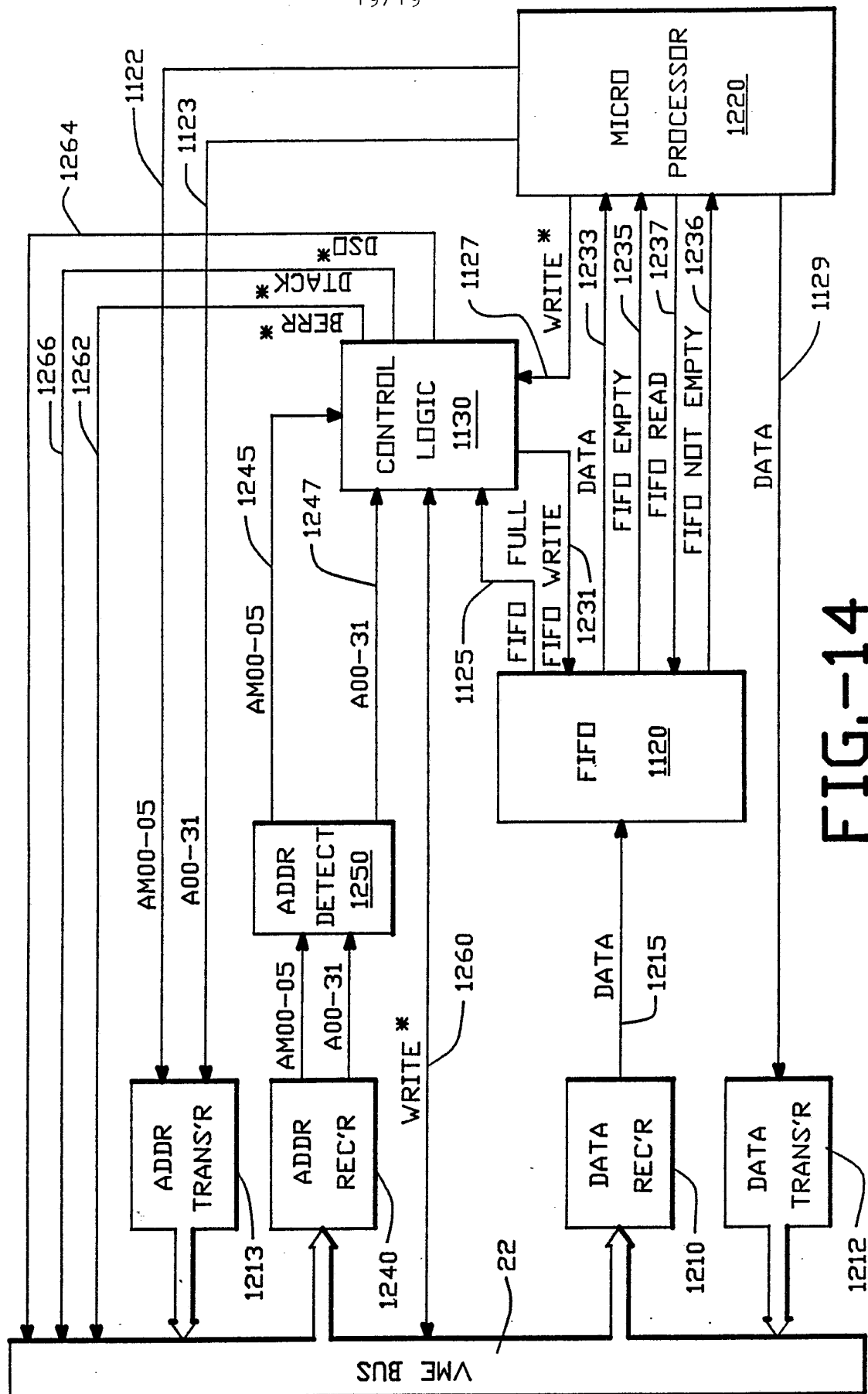
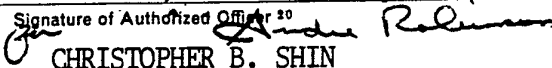


FIG.-14



# INTERNATIONAL SEARCH REPORT

International Application No PCT/US90/04697

<b>I. CLASSIFICATION OF SUBJECT MATTER</b> (if several classification symbols apply, indicate all) <sup>3</sup>		
According to International Patent Classification (IPC) or to both National Classification and IPC		
IPC(5) G06F 13/38, 13/00		
US 364/200		
<b>II. FIELDS SEARCHED</b>		
Minimum Documentation Searched <sup>4</sup>		
Classification System	Classification Symbols	
US	364/200	
Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched <sup>5</sup>		
<b>III. DOCUMENTS CONSIDERED TO BE RELEVANT</b> <sup>14</sup>		
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>
X	US, A, 4,062,059 (SUZUKI ET AL) 06 DECEMBER 1977	1-4
X	US, A, 4,423,482 (KARGROVE ET AL) 27 DECEMBER 1983	1-4
X	US, A, 4,285,038 (SUZUKI ET AL) 18 AUGUST 1981	1-4
<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>2</sup>	Date of Mailing of this International Search Report <sup>2</sup>	
06 NOVEMBER 1990	<div style="font-size: 1.5em; font-weight: bold;">22 FEB 1991</div>	
International Searching Authority <sup>1</sup>	Signature of Authorized Officer <sup>20</sup>	
ISA/US	 <b>CHRISTOPHER B. SHIN</b>	