(12)

PATENT SPECIFICATION (11) 1588804

1588804

(21) Application No. 3057/80 (22) Filed 17 June 1977 (19)
(62) Divided out of No. 1588803
(31) Convention Application No. 721043 (32) Filed 7 Sept. 1976 in
(33) United States of America (US)
(44) Complete Specification published 29 April 1981
(51) INT. CL.³ G06F 13/00
(52) Index at acceptance
G4A 17P 1C MT1



(54) A PROCESSOR MODULE FOR A MULTIPROCESSOR SYSTEM

(71) We, TANDEM COMPUTERS INCORPORATED, a Corporation of the State of California, United States of America, of 20605 Valley Green Drive, Cupertino, California 95014, United States of America, do hereby declare the invention, for which we pray that a patent may be granted to us, and the method by which it is to be performed, to be particularly described in and by the following statement:—

This invention relates to a processor module for a system in which separate processor modules are inter-connected for parallel processing.

The present invention relates particularly to a processor module for a multiprocessor system which can support high transaction rates to large on-line data bases and in which no single component failure can stop or contaminate the operation of the system.

There are many applications which require on-line processing of large volumes of data at high transaction rates. For example, such processing is required in retail applications for automated point of sale, inventory and credit transactions and in financial institutions for automated funds transfer and credit transactions.

In computing applications of this kind it is important, and often critical, that the data processing not be interrupted. A failure of an on-line computer system can shut down a portion of the related business and can cause considerable loss of data and money.

Thus, an on-line system of this kind must provide not only sufficient computing power to permit multiple computations to be done simultaneously, but it must also provide a mode of operation which permits data processing to be continued without interruption in the event some component of the system fails.

The system should operate either in a fail-safe mode (in which no loss of throughput occurs as a result of failure) or in a fail-safe mode (in which some slowdown occurs but full processing capabilities are maintained) in the event of a failure.

Furthermore, the system should also operate in a way such that a failure of a single component cannot contaminate the operation of the system. The system should provide fault-tolerant computing. For fault-tolerant computing all errors and failures in the system should either be corrected automatically, or if the failure or error cannot be corrected automatically, it should be detected, or if it cannot be detected, it should be contained and should not be permitted to contaminate the rest of the system.

Since a single processor module can fail, it is obvious that a system which will operate without interruption in an on-line application must have more than one processor module.

Sytems which have more than one processor can therefore meet one of the necessary conditions for non-interruptible operation. However, the use of more than one processor module in a system does not by itself provide all the sufficient conditions for maintaining the required processing capabilities in the event of component failure, as will become more apparent from the description to follow.

Computing systems for on-line, high volume, transaction oriented, computing applications which must operate without interruption therefore require multiprocessors as a starting point. But the use of multiprocessors does not guarantee that all of the sufficient conditions will be met, and fulfilling the additional sufficient conditions for on-line systems of this kind has presented a number of problems in the prior art.

The prior art approach to uninterrupted data processing has proceeded generally along two lines — either adapting two or more large, monolithic, general purpose computers for joint operation or interconnecting a plurality of minicomputers to provide multiprocessing capabilities.

In the first case, adapting two large monolithic general purpose computers for joint operation, one conventional prior art approach has been to have the two computers share a common memory. Now in this type of multiprocessing system a failure in the shared memory can stop the entire system. Shared memory also presents a number of

SPECIFICATION AMENDED - SEE ATTACHED SLIP

other problems including sequencing accesses to the common memory. This system, while meeting some of the necessary conditions for uninterruptible processing, does not meet all of the sufficient conditions.

Furthermore, multiprocessing systems using large general purpose computers are quite expensive because each computer is constructed as a monolithic unit in which all components (including the packaging, the cooling system, etc.) must be duplicated each time another processor is added to the system even though many of the duplicated components are not required.

The other prior art approach of using a plurality of minicomputers has (in common with the approach of using large general purpose computers) suffered from the drawback of having to adapt a communications link between computers that were never originally constructed to provide such a link. The required links were, as a result, usually made through the input/output channel. Connections through the input/output channel are necessarily slower than internal transfers within the processor itself, and such interprocessor links have therefore provided relatively slow interprocessor communication.

Furthermore, the interprocessor connections required special adapter cards that added substantially to the cost of the overall system and that introduced the possibility of single component failures which could stop the system. Adding dual interprocessor links and adapter cards to avoid problems of critical single components failures increased the overall system cost even more substantially.

Providing dual links and adapter cards between all processors generally became very cumbersome and quite complex from the standpoint of operation.

Another problem of the prior art arose out of the way in which connections were made to peripheral devices.

If a number of peripheral devices are connected to a single input/output bus of one processor in a multiprocessor system and that processor fails, then the peripheral devices will be unavailable to the system even though the failed processor is linked through an interprocessor connection to another processor or processors in the system.

To avoid this problem, the prior art has provided an input/output bus switch for interconnecting input/output busses for continued access to peripheral devices when a processor associated with the peripheral devices on a particular input/output bus fails. The bus switches have been expensive and also have presented the possibility of single component failure which could down a substantial part of the overall system.

Providing software for the prior art multiprocessor systems has also been a major problem.

Operating systems software for such multiprocessing systems has tended to be nonexistent. Where software had been developed for such multiprocessor systems, it quite often was restricted to a small number of processors and was not adapted for the inclusion of additional processors. In many cases it was necessary either to modify the operating system or to put some of the operating system functions into the user's own program — an expensive, time-consuming operation.

The prior art lacked a satisfactory standard operating system for linking processors. It also did not provide an operating system for automatically accommodating additional processors in a multiprocessing system constructed to accommodate the modular addition of processors as increased computering power was required.

According to the present invention there is provided a processor module for a multiprocessor system of the kind in which separate processor modules are interconnected for parallel processing, said processor module comprising; a central processing unit having a processor clock; a memory; and interprocessor control means for connecting the processor module to any other processor module in the multiprocessor system across an interprocessor bus operated at a bus clock that is not synchronous with any of the processor clocks, said interprocessor control means including queue means for buffering data between the interprocessor bus and the processor memory, said queue means having a first logic state machine operated in synchronism with the bus clock and a second logic state machine operated in synchronism with the processor clock, and interlock means operatively associated with the first and second logic state machines for qualifying certain transitions of one of the state machines in dependence upon the state of the other state machine so that the interprocessor bus can be operated at the bus clock rate and the central processing unit and memory of the processor module can be operated at the processor clock rate without loss of the data being transferred or loss of information about the state of the transfer sequence.

An embodiment of the present invention will now be described, by way of example, with reference to the accompanying drawings, in which:

FIGURE 1 is an isometric, block diagram view of a multiprocessor system incorporating one embodiment of the processor module of the present invention. Figure 1 shows several processor modules connected by two interprocessor buses (an X bus and a Y bus) with each bus controlled by a bus controller.

Figure 1 also shows several dual-port device controllers with each device controller connected to the input/output (I/O) buses of two processor modules.

FIGURE 2 is a block diagram view showing details of the connections of the X bus controller and the Y bus controller to the individual processor modules. Figure 2 shows, in diagrammatic form, the connections between each bus controller and the interprocessor control of an individual processor module.

FIGURE 3 is a detailed diagrammatic view of the logic of one of the bus controllers shown in Figure 2.

FIGURE 4 is a detailed diagrammatic view of the logic for the shared output buffer and control in the interprocessor control of a processor module as illustrated in Figure 2.

FIGURE 5 is a view like Figure 4 but showing the logic for an inqueue buffer and control of the interprocessor control for a processor module.

FIGURE 6 is a state diagram of the logic for a bus controller and illustrates how the logic responds to the protocol lines going into the bus controller and generates the protocol lines going out of the bus controller to the processor modules.

FIGURE 7 is a state diagram like Figure 6 but showing the logic for the shared outqueue buffer and control of Figure 4.

FIGURE 8 is a state diagram like Figures 6 and 7 but showing the logic for the inqueue buffer and control of Figure 5.

FIGURE 9 is a diagrammatic view showing the time sequence for the transmission of a given packet between a sender processor module and a receiver processor module.

FIGURE 10 is a logic diagram of the bus empty state logic section and the processor fill state logic section of the outqueue buffer and the control 67 shown in Figure 4.

FIGURE 11 is a listing of logic equations for the logic diagram shown in Figure 10.

FIGURE 12 is a block diagram of the input/output (I/O) system of the multiprocessor system shown in Figure 1.

FIGURE 13 is a block diagram of the input/output (I/O) channel of a processor module. Figure 13 shows the major components of the I/O channel and the data path relating those component parts.

FIGURE 14 is a detailed view showing the individual lines in the I/O bus of Figure 1.

FIGURE 15 is an I/O channel protocol diagram showing the state changes of the T bus for an execute input/output (EIO) caused by the microprogram in the CPU. The sequence illustrated is initiated by the CPU and is transmitted through the I/O channel of the processor module and on the T bus to a device controller as shown in Figure 1.

FIGURE 16 is an I/O channel protocol diagram showing the state changes of the T bus for a reconnect and data transfer sequence initiated by the I/O channel microprogram in response to a request signal from a device controller.

FIGURE 17 is an I/O channel protocol diagram showing the state changes of the T bus for an interrogate I/O (IIO) instruction or an interrogate high priority I/O (HIIO) instruction initiated by the CPU microprogram. The sequence illustrated is transmitted over the T bus to a device controller.

FIGURE 18 is a table identifying the functions referred to by the mnemonics in Figures 15 to 17.

FIGURE 19 is a block diagram showing the general structure of the ports and a device controller as illustrated in Figure 1.

FIGURE 20 is a block diagram of a port 43 shown in Figure 19. This Figure 20 shows primarily the data paths within a port.

FIGURE 21 is a block diagram showing the data path details of the interface common logic of the device controller shown in Figure 19.

FIGURE 22 is a block diagram showing the component parts of a data buffer in the control part of a device controller as illustrated in Figure 19.

FIGURE 23 is a graph illustrating the operation of the data buffer illustrated in Figures 22 and 19.

FIGURE 24 is a timing diagram illustrating the relationship of SERVICE OUT (SVO) from the channel to the loading of data into the port data register (Figure 21) and illustrates how the parity check is started before data is loaded into the register and is continued until after the data has been fully loaded into the register.

FIGURE 25 is a schematic view showing details of the power on circuit (PON) shown in Figures 19 and 21.

FIGURE 26 is a logic diagram of the buffer control logic of the data buffer (shown in Figure 22) of a device controller. Figure 26 shows how the buffer control logic controls the hand-shakes on the data bus and controls the input and output pointers.

FIGURE 27 is a listing of the logic equations for the select register shown in Figure 20. These logic equations are implemented by the port control logic shown in Figure 20.

FIGURE 28 is a timing diagram showing the operation of the two line handshake between the I/O channel and the ports.

FIGURE 29 is a logic diagram showing the logic for the general case of the handshake shown in Figure 28. The logic shown in Figure 29 is part of the T bus machine of the input/output channel shown in Figure 13.

FIGURE 30 is a block diagram of a power distribution system. Figure 30 shows how a

plurality of independent and separate power supplies are distributed and associated with the dual port device controllers for insuring that each device controller has both a primary and an alternate power supply.

FIGURE 31 is a block diagram of the memory system and shows details of the memory 107 of a processor module 33 shown in Figure 1.

FIGURE 32 shows the various states of a two processor system running an application program which is required to be running continuously. The diagrams illustrate the two processors successively failing and being repaired and the application program changing its mode of operation accordingly.

Fig. 1 is an isometric diagrammatic view of part of a multiprocessor system constructed in accordance with one embodiment of the present invention. In Fig. 1 the multiprocessor system is indicated generally by the reference numeral 31.

The multiprocessor system 31 includes individual processor modules 33. Each processor module 33 comprises a central processing unit 105, a memory 107, an input/output channel 109 and an interprocessor control 55.

The individual processor modules are interconnected by interprocessor buses 35 for interprocessor communications.

In a specific embodiment of the multiprocessor system 31, up to sixteen processor modules 33 are interconnected by two interprocessor buses 35 (indicated as the X bus and the Y bus in Fig. 1).

Each interprocessor bus has a bus controller 37 associated with that bus.

The bus controllers 37, interprocessor buses 35 and interprocessor controls 55 (Fig. 1), together with associated microprocessors 113, microprograms 115 and bus receive tables 150 (Fig. 2) provide an interprocessor bus system. The construction and operation of this interprocessor bus system are illustrated in Figs. 2—11 and 42 and are described in more detail below under the subtitle The Interprocessor Bus System.

The multiprocessor system 31 has an input/output (I/O) system for transferring data between the processor modules 33 and peripheral devices, such as the discs 45, terminals 47, magnetic tape drives 49, card readers 51, and line printers 53 shown in Fig. 1.

The I/O system includes one I/O bus 39 associated with each I/O channel 109 of a processor module and one or more multiport device controllers 41 may be connected to each I/O bus 39.

In the specific embodiment illustrated, each device controller 41 has two ports 43 for connection to two different processor modules 33 so that each device controller is connected for access by two processor modules.

The I/O system includes a microprocessor 119 and a microprogram 121 in the I/O channel 109 (See Fig. 12.) which are dedicated to input/output transfers.

As also diagrammatically illustrated in Fig. 12, the microprocessor 113 and microprogram 115 of the central processing unit 105 and an input/output control table 140 in the main memory 107 of each processor module 33 are operatively associated with the I/O channel 109.

The construction and operation of these and other components of the I/O system are illustrated in Figs. 12—29 and are described in detail below under the subtitle The Input/Output System and Dual Port Device Controller.

The multiprocessor system includes a power distribution system 301 which distributes power from separate power supplies to the processor modules 33 and to the device controllers 41 in a way that permits on-line maintenance and also provides redundancy of power on each device controller.

As illustrated in Fig. 30, the power distribution system includes separate and independent power supplies 303.

A separate power supply 303 is provided for each processor module 33, and a bus 305 supplies the power from the power supply 303 to the central processing unit 105 and memory 105 of a related processor module 33.

As also illustrated in Fig. 30, each device controller 41 is connected for supply of power from two separate power supplies 303 through an automatic switch 311. If one power supply 303 for a particular device controller 41 fails, that device controller is supplied with power from the other power supply 303; and the changeover is accomplished smoothly and without any interruption or pulsation in the power supplied to the device controller.

The power distribution system coacts with the dual port system of the device controller to provide nonstop operation and access to the peripheral devices in the event of a failure of either a single port 43 or a single power supply 303.

The multiprocessor system includes a power on (PON) circuit 182 in several components of the system to establish that the power to that particular component is within certain acceptable limits.

For example, the PON circuit 182 is located in each CPU 105, in each device controller 41, and in each bus controller 37.

The purpose of the PON circuit is to present a signal establishing the level of power applied to that particular component; and if the power is not within certain predetermined acceptable limits, then the signal output is used to directly disable the appropriate bus signal of the component in

which the PON is located.

The power-on circuit functions in four states — power off; power going from off to on; power on; and power going from on to off.

The power-on circuit initializes all of the logic states of the system as the power is brought up; and in the present invention, the power-on circuit provides an additional and very important function of providing for a fail-safe system with on line maintenance. To do this, the power-on circuit in the present invention is used in a unique way to control the interface circuits which drive all of the intercommunication buses in the system.

The multiprocessor system includes a memory system in which the physical memory is divided into four logical address areas — user data, system data, user code and system code.

The memory system includes a map and control logic for translating all logical addresses to physical addresses and for indicating pages absent from primary storage bit present in secondary storage as required to implement a virtual memory system in which the physical page addresses are invisible to users.

The memory system incorporates a dual port access to the memory by the central processing unit 105 and the I/O channel 109. The I/O channel 109 can therefore access the memory 107 directly (without having to go through the central processing unit 105) for data transfers to and from a device controller 41.

An error detection system is incorporated in the memory system for correcting all single bit and detecting all double bit errors when semiconductor memory is used in the memory system. This error detection system utilizes a 16 bit data field and a 6 bit check field and includes a data bit complementer for correcting single bit errors.

Figs. 37 through 41 and the related disclosure illustrate and describe details of the error detection system.

Before going into the detailed description of the systems and components noted generally above, it should be noted that certain terminology will have the following meanings as used in this application.

The term "software" will refer to an operating system or a user program instructions; the term "firmware" will refer to a microprogram in read only memory; and the term "hardware" will refer to actual electronic logic and data storage.

The operating system is a master control program executing in each processor module which has primary control of the allocation of all system resources accessible to that processor module. The operating system provides a scheduling function and determines what process has use of that processor module. The operating system also allocates the use of primary memory (memory management), and it operates the file system for secondary memory management. The operating system also manages the message system. This provides a facility for information transfer over the interprocessor bus.

The operating system arrangement parallels the modular arrangement of the multiprocessor system components described above, in that there are no "global" components.

At the lowest level of the software system, two fundamental entities are implemented — processes and messages.

A process is the fundamental entity of control within a system.

Each process consists of a private data space and register values, and a possibly shared code set. A process may also access a common data space.

A number of processes coexist in a processor module 33.

The processes may be user written programs, or the processes may have dedicated functions, such as, for example, control of an I/O device or the creation and deletion of other processes.

A process may request services from another process, and this other process may be located in the same processor module 33 as a process making the request, or the other process may be located in some other processor module 33.

The processes work in an asynchronous manner, and the processes therefore need a method of communication that will allow a request for services to be queued without "races" (a condition in which the outcome depends upon the sequence of which process started first)—thus the need for "messages" (an orderly system of interprocessor module communication described in more detail below).

Also, all interprocessor module communication should appear the same to the processes, regardless of whether the processes are in the same or in different processor modules.

As will become more clear from the description to follow, the software structure parallels the hardware; and different processes can be considered equivalent to certain components of the hardware in arrangement and function.

For example, just as the I/O channel 109 communicates over the I/O bus 39 to the device controller 41, a user process can make a request (using the message system) to the process associated with that device controller 41; and then the device process returns status back similar to the way the device controller 41 returns information back to the I/O channel 109 over the I/O bus 39.

The other fundamental entity of the soft-

ware system, the message, consists of a request for service as well as any required data. When the request is completed, any required values will be returned to the requesting process.

When a message is to be sent between processes in two different processor modules 33, the interprocessor buses 35 are used. However, as noted above, all communication between processes appears the same to the processes, regardless of whether they are in the same or in different processor modules 33.

This software organization provides a number of benefits.

This method of structuring the software also provides for significantly more reliable software. By being able to compartmentalize the software structure, smaller module sizes can be obtained, and the interfaces between modules are well defined.

The system is also more maintainable because of the compartmentalization of function.

The well defined modules and the well defined interfaces in the software system also provide advantages in being able to make it easily expandible—as in the case of adding additional processor modules 33 or device controllers 41 to the multiprocessor system.

Furthermord, there is a benefit to the user of the multiprocessor system and software system in that the user, writing his program, need not be aware of either the actual machine configuration or the physical location of other processes.

Just as the hardware provides multiple functionally equivalent modules with redundant interconnects, so does the software.

For example, messages going between processes in different processor modules 33 may use either interprocssor bus 35. Also, device controllers 41 may be operated by processes in either of the processor modules 33 connected to the device controller 41.

The multiprocessor hardware system and software system described above enable the user to develop a fault tolerant application system by virtue of its replicated modules with redundant interconnects.

THE INTERPROCESSOR BUS SYSTEM:

As pointed out above, the individual processor modules 33 are interconnected by two interprocessor buses 35 (an X bus and a Y bus) with each bus controlled by a related bus controller 37. Each interprocessor bus 35, in combination with its bus controller 37 and a related interprocessor control 55 in each processor module 33, provides a multi-module communication path from any one processor module to any other processor module in the system. The use of two buses assures that two independent paths exist between all processor modules in the system. Therefore, a failure in one path (one bus) does not prevent communication between the processor modules.

The bus controller 37 for each interprocessor bus 35 is a controller which is, in a preferred form of the invention, separate and distinct from the processor modules 33.

Each interprocessor bus 35 is a synchronous bus with the time synchronization provided by a bus clock generator in the bus controllers 37. The interprocessor control portions 55 of all of the modules associated with the bus make state changes in synchronism with that bus clock during transfers over the bus.

As will be described in more detail below, the CPU 105 operates on a different clock from the interprocessor bus clock. During the filling of an outqueue or the emptying of an inqueue in the interprocessor control 55 by the CPU, the operation takes place at the CPU clock rate. However, transmission of packets over the interprocessor bus always takes place at the bus clock rate.

It is an important feature of the present invention that the information transmitted over the interprocessor bus is transferred at high transmission rates without any required correspondence to the clock rates of the various CPUs 105. The information transfer rate over the interprocessor bus is also substantially faster than would be permitted by direct memory accesses into and out of the memory sections 107 at memory speed. This ensures that there is adequate bus bandwidth even when a large number of processor modules is connected in a multiprocessor system.

A benefit of using separate clocks for each CPU 105 is that a master system clock is not required, and this eliminates a potential source of single component failure which could stop the entire system.

The interprocessor control 55 incorporates logic interlocks which make it possible to operate the interprocessor buses 35 at one clock rate and each CPU 105 at its own independent clock rate without loss of data.

The information transmitted over the bus is transmitted in multiword packets. In a preferred form of the present invention each packet is a sixteen word packet in which fifteen of the words are data words and one word is a check word.

The control logic within the bus controller 37 and the interprocessor controls 55 of the individual modules 33 follows a detailed protocol. The protocol provides for establishing a sender-receiver pair and a time frame for the data packet transfer. At the end of the time frame for the transmission of the data packet, the bus controller 37 is released for another such sequence. The specific manner in which these functions are carried out will

become more apparent after a description of the structural features of Figs. 3—9 below.

X bus 35 is identical in structure to the Y bus 35, so the structure of only one bus will be described in detail.

As illustrated in Fig. 2, each bus 35 comprises sixteen individual bus data lines 57, five individual bus protocol lines 59, and one clock line 61, and one select line 63 for each processor module 33.

As also illustrated in Fig. 2, the interprocessor control 55 of each processor module 33 includes two inqueue sections 65 (shown as an X inqueue section and a Y inqueue section in Fig. 2) and a shared outqueue section 67.

With the specific reference to Fig. 4, the shared outqueue section 67 includes an outqueue buffer 69 which performs a storage function. In a preferred form the buffer 69 has sixteen words of sixteen bits each. The buffer 69 is loaded by the CPU and holds the data until the packet transmission time, at which time the data is gated out to the bus, as will be described in more detail below.

The outqueue section 67 also includes a receive register 71, which in a preferred form of the invention is a four bit register. This register is loaded by the CPU with the number of the processor module to which the data will be sent.

The control part of the outqueue section 67 includes a processor fill state logic section 73 which operates in synchronism with the CPU clock, a bus empty state logic section 75 which operates in synchronism with the X or Y bus clock, and an outqueue counter 77. During filling of the outqueue buffer 69 by the CPU, the outqueue counter 77 scans the buffer 69 to direct the data input into each of the sixteen words of the buffer; and, as the sixteenth word is stored into the outqueue buffer 69, the outqueue counter 77 terminates the fill state.

The outqueue section 67 also includes an outqueue pointer 79 which connects the entire outqueue section to either the X bus or the Y bus 35. The outqueue pointer 79 allows the logic sections 73 and 75 and the buffer 69 to be shared by the X and Y interprocessor buses 35.

As illustrated in Fig. 3, the bus controller 37 comprises a bus control state logic section 81, a sender counter 83, a processor select logic section 85, a receive register 87, a packet counter 89 and a bus clock generator 91.

With reference to Fig. 5, each inqueue section 65 comprises a bus fill state logic section 93 which operates in synchronism with the bus clock, a sender register 95, an inqueue buffer 97, an inqueue counter 99, and a processor empty state logic section 101 which operates in synchronism with the CPU clock.

Fig. 6 is a state diagram of the bus control logic 81 of the bus controller 37.

Fig. 7 is a state diagram of the logic sections 73 and 75 of the outqueue section 67.

Fig. 8 is a state diagram of the logic sections 93 and 101 of the inqueue sections 65.

With reference to Fig. 7, the processor fill state logic section 73 has basically four states—EMPTY, FILL, FULL and WAIT —as indicated by the respective legends. The bus empty state logic section 75 has basically four states—IDLE, SYNC, SEND and DONE—as illustrated by the legends.

Continuing with a description of the notation in Fig. 7, the solid lines with arrows indicate transitions from the present state to the next state. Dashed arrows ending on the solid arrows indicate conditions which must be satisfied for the indicated transition to take place.

The synchronization of state machines running off relatively asynchronous clocks requires a careful construction of an interlock system. These important interlocks are noted by the dashed arrows in the state diagrams. These interlocks perform a synchronization of two relatively asynchronous state machines. The dashed arrows in Fig. 7 and Fig. 8 running between the state machines thus indicate signals which synchronize (qualify) the indicated transistions of the state machines.

With reference to the FILL state for the logic section 73, it should be noted that the store outqueue condition will not cause an exit from the FILL state until the outqueue counter 77 has advanced to count 15 (on a count which starts with zero) at which time the FILL state will advance to the FULL state.

Similarly, it should be noted that the SEND state of the logic section 75 will not terminate on the select and send command condition until the outqueue counter 77 reaches count 15, at which time the SEND state advances to the DONE state.

The asterisk in the notation of Fig. 7 indicates an increment of the outqueue counter 77.

Fig. 6 shows the state diagram for logic 81 of the bus controller and illustrates that the logic has basically four states—IDLE, POLL, RECEIVE and SEND.

The notation in Fig. 6 is the same as that described above for Fig. 7. A solid arrow line indicates a state transition from one state to another and a dotted arrow line to that solid arrow line indicates a condition which must occur to allow the indicated (solid line arrow) transition to occur. An asterisk on a state transition in this case indicates that simultaneously with the indicated transition the sender counter 83 is incremented by one.

The dashed arrow output lines in Fig. 6

indicate protocol commands issued from the bus controller to the interprocessor bus.

In both Fig. 7 and Fig. 6 a dashed arrow leaving a state indicates a logic output from that state such as a logic output signal to a protocol line (in the case of the bus empty state logic 75) or to a status line of the processor module (in the case of the processor fill state logic 73).

Fig. 8 shows the state diagrams for the bus fill state logic section 93 and the processor empty state logic section 101.

The state diagram for the logic section 93 includes four states—SYNC, ACKNOWLEDGE, RECEIVE and FULL.

The state diagram for the logic section 101 includes four states—RESET, READY, INTERRUPT and DUMP.

The notation (solid line arrows and dashed line arrows) is the same as described above for Fig. 7 and Fig. 6.

The asterisk in Fig. 8 indicates an increment in the inqueue counter 99.

Fig. 9 is a timing diagram showing the time sequence in which the state changes given in Figs. 6, 7 and 8 occur.

The sequence shown in Fig. 9 accomplishes the transmission of a packet from one processor module to another processor module at the bus clock rate (assuming that the intended receiver module is ready to receive the packet).

Fig. 9 shows the time sequences for a successful packet transfer with individual signal representations listed from top-to-bottom in Fig. 9 and with time periods of one bus clock each shown from left-to-right in the order of increasing time in Fig. 9.

The top line in Fig. 9 indicates the state of the bus controller, and each division mark represents a clock period or cycle of the bus clock generator 91 shown in Fig. 3. Each time division of the top line carries down vertically through the various signal representations listed by the legends at the left side of the figure.

Taking the signals in the sequence presented from top-to-bottom in Fig. 9, the first signal (below the bus controller state line) is the SEND REQUEST signal (one of the protocol group indicated by the reference numeral 59 in Fig. 3) and specifically is the signal which may be asserted by the outqueue control logic section 67 of any processor module 33. The signal is transmitted to the bus control state logic section 81 of the bus controller 37 (see Fig. 3).

The next signal shown in Fig. 9 (the SELECT signal) represents a signal which originates from the processor select logic section 85 of the bus controller 37 and which is transmitted on only one at a time of the select lines 63 to a related processor module 33.

The next signal represented in Fig. 9, the SEND ACKNOWLEDGE signal, may be asserted only by a particular processor 33 when that processor is selected and when its bus empty state logic section 75 is in the SEND state (as illustrated in the third state of Fig. 7). This SEND ACKNOWLEDGE signal is used by the bus controller 37 to establish the identity of a processor module 33 wishing to send a packet.

The next signal, the RECEIVE COMMAND signal, represents a signal from the bus controller 37 transmitted on one of the protocol lines 59. This signal does two things.

First of all, this signal in combination with receiver SELECT interrogates the receiver processor module 33 to find out whether this receiver module is ready to receive (as indicated by the ACKNOWLEDGE state in Fig. 8).

Secondly, this signal has a secondary function of disabling the bus empty state logic section 75 of the receiving module so that the receiving module cannot gate an intended receiver number to the data bus should the outqueue section of the intended receiver module 33 also have a data packet of its own ready to send.

In this regard, during the time that the sender processor is asserting the SEND ACKNOWLEDGE signal it is also gating the receiver number to the bus for use by the bus controller 37. The bus 35 itself is, of course, a non-directional bus so that the information can be gated to the data bus 57 by any module for use by either the bus controller 37 for a control function or for use by another processor for an information transfer function. It should be noted that a module 33 may gate data to the bus only when its SELECT line is asserted and the RECEIVE COMMAND signal is not asserted.

During the time that the RECEIVE COMMAND signal is asserted the bus controller 37 is gating the sender number to the data bus 57 for capture by the selected receiver processor module.

The next signal line (the RECEIVE ACKNOWLEDGE line in Fig. 9) represents a signal which is transmitted from the selected receiving module's bus fill state logic section 93 to the bus control state logic section 81 of the bus controller 37 (over one of the protocol lines 59) to indicate that the selected receiver module is in the ACKNOWLEDGE state (as indicated by the legend in Fig. 8) and thus ready to receive the packet which the sender module has ready to transmit.

If the RECEIVE ACKNOWLEDGE signal is not asserted by the receiver module, the sender SELECT, the SEND COMMAND and the time frame transmission of the data packet itself will not occur.

If the RECEIVE ACKNOWLEDGE signal is asserted, then the sequence indicated

by the SEND COMMAND line will occur.

The SEND COMMAND line represents a signal which originates from the bus control state logic section 81 of the bus controller 37 and which is transmitted to the bus empty state logic section 75 of the sender processor module 33 over one of the protocol lines 59.

In combination with a SELECT of the sender processor module the SEND COMMAND signal enables the sender processor module to send a packet to the receiver module during the sixteen clock cycles bracketed by the SEND COMMAND signal.

The final line (the data/16 line) represents the information present on the data lines 57 during the above-described sequence.

The data is gated to the bus by the selected sender processor module and is transmitted to the receiver processor module into the inqueue buffer 97 (see Fig. 5) during this sixteen clock cycle time frame. This assumes that the RECEIVE ACKNOWLEDGE signal was received by the bus controller in response to the RECEIVE COMMAND signal.

If the RECEIVE ACKNOWLEDGE signal had not been received by the bus controller, then the SEND COMMAND signal would not have been asserted and the bus controller 37 would have resumed the POLL state as shown in Fig. 6.

With reference to Figs. 2, 7, 10 and 11, a typical operation of the outqueue buffer and control 67 of one processor module 33 will now be described.

As illustrated in Fig. 10, the processor fill state logic section 73 includes two flip-flops A and B, and the bus empty state logic section 75 includes two flip-flops C and D.

Summarizing the state assignments as shown by the AB and CD tables in Fig. 10, the EMPTY state is defined as $A=0$, $B=0$. The FILL state is defined as $A=1$, $B=0$. The FULL state is defined as $A=1$, $B=1$; and the WAIT state is defined as $A=0$, $B=1$.

Similarly, the corresponding combinations of the C and D state variables are defined to be the IDLE, SYNC, SEND and DONE states respectively. State assignments previously listed could also be given in form of logic equations. For example, EMPTY = $\bar{A}\cdot\bar{B}$, and this notation is utilized in the Fig. 11 logic equation listings.

In operation and with specific reference to Fig. 7, the initial state reached through power on initialization or manual reset is the EMPTY state shown in the top left part of Fig. 7.

The EMPTY state of the processor fill state logic 73 provides a ready signal to the central processor unit (CPU) 105 to indicate the presence of that state, as indicated by the dashed arrow RDY shown as leaving the empty state in Fig. 7.

The CPU firmware (microprogram) in response to that ready signal, when a transmission over the interprocessor bus is required, will provide a store receive signal (shown by the dashed arrow incoming to the diagram in Fig. 7). This store receive signal qualifies (synchronizes) the transition which advances the EMPTY state to the FILL state.

The CPU firmware, to transfer data into the outqueue buffer 69, will provide a store outqueue signal (the dashed arrow entering the diagram in Fig. 7) for each word to be stored in the buffer 69.

Each occurence of this store outqueue signal will advance the outqueue counter 77, commencing with a count of zero, until a count of 15 is reached.

On the sixteenth occurrence of the store outqueue signal a transition from the FILL to the FULL state, as illustrated by the solid line arrow in Fig. 7, is allowed.

The FULL state of the processor FILL state logic provides a synchronization condition to the bus empty state logic denoted by the dashed arrow leaving the FULL state of logic 73 and going down to the logic 75 in Fig. 7.

The processor fill state logic 73 will remain in the FULL state until the bus empty state logic 75 has subsequently reached the DONE state.

Now, referring specifically to the bus empty state logic denoted by 75 in Fig. 7, the initial state, IDLE, for the logic section 75 in Fig. 7 is again provided by power on initialization or manual reset.

The bus empty state logic 75 will remain in the IDLE state until the transistion to the SYNC state is allowed as shown by the dashed arrow from the FULL state of the processor fill 73.

The empty state logic 75 will proceed with no qualification required from the SYNC state to the SEND state.

It is in the SEND state that the SEND REQUEST signal to the bus and to the bus controller is asserted (as indicated by the dashed arrow going down and leaving the diagram 75 from the SEND state).

In response to this SEND REQUEST signal, the bus controller logic 81 (Fig. 6) will poll processor modules successively until the sender is identified (as discussed earlier with reference to Fig. 9).

The bus controller will issue a RECEIVE COMMAND and SELECT to the intended receiver processor module; and upon receipt of the RECEIVE ACKNOWLEDGE signal will proceed to the packet time frame (also identified in Fig. 9).

During the packet time frame the bus controller asserts SELECT of the sender processor module and also asserts the SEND COMMAND signal to the sender processor module.

This SELECT signal and SEND COM-

MAND signal is shown as entering the diagram and qualifying (synchronizing) transitions leaving and entering the SEND state as noted in Fig. 7.

Each bus clock while SELECT and SEND COMMAND are asserted will advance the outqueue counter 77 commencing with a count of zero.

On the sixteenth clock period of SELECT and SEND COMMAND the transition terminating the SEND state and advancing to the DONE state is qualified (synchronized as shown by the dashed arrow allowing that transition).

When the empty state logic 75 has reached the DONE state, a transition of the processor fill state logic 73 from FULL to WAIT is qualified (as denoted by the dashed arrow leaving the done state).

Next, the WAIT state of the processor fill state logic 73 qualifies a transition of the bus empty state logic 75 from the DONE state to the IDLE state (as denoted by a dashed arrow leaving the WAIT state and qualifying the indicated transition).

Finally, the bus empty state logic 75, being in the IDLE state, qualifies the transition of the processor fill state logic 73 from the WAIT state to the EMPTY state (as denoted by the dashed arrow leaving the IDLE state).

At this point a packet has been loaded into the outqueue buffer 69 by the processor module and transmitted over the bus 35 to the receiver processor module; and the outqueue control processor fill state logic 73 and bus empty state logic 75 have returned to their initial states.

The above description relates to the transitions and qualifications indicated in Fig. 7. The action of the logic sections 73 and 75 involved in the above description of operation of Fig. 7 will now be noted with reference to the logic diagram of Fig. 10 and the logic equation listing of Fig. 11.

With reference to Fig. 10, as noted above, the flip-flops A and B are JK flip-flops and are edge triggered flip-flops in that state changes occur only on clock transitions (as indicated by the small triangular symbols and legends on the lefthand sides of the flip-flops A and B in Fig. 10).

The primary significance of the logic diagram in Fig. 10 is to illustrate the transition from one state to another in the state machines shown in Fig. 7. Thus, to illustrate the transition from IDLE to SYNC in the empty state logic 75, the operation proceeds as follows.

To implement a change from the IDLE state to the SYNC state, the state variable C must be set.

The logic equation for the J input of state variable C is as shown in Fig. 11 and is indicated by the reference numeral 103. In this equation the interlock (shown by the dashed arrow from the full state of the fill state logic 73 in Fig. 7 to the transition) corresponds to the quantity (A·B) or (FULL) in the equation indicated by the reference number 103. The D or (IDLE) in the equation indicated by reference numeral 103 in Fig. 11 corresponds to the IDLE state shown by the legend in Fig. 7. The J in the equation corresponds to the J input of the C flip-flop in Fig. 10. And the (C) corresponds to the true output of the C flip-flop in Fig. 10.

Other state transitions of the Fig. 7 diagram will not be described in further detail with reference to Figs. 10 and 11 since it is believed that these transitions as carried out by the logic diagram in Fig. 10 and the logic equations in Fig. 11 are clear from the above examples of the transition from IDLE state to SYNC state as described in detail above.

Figs. 10 and 11 show the logic diagram and logic equations for the state diagram of the outqueue buffer and control 67. Corresponding logic diagrams and logic equations have not been illustrated for the inqueue buffer and control 65 or the bus controller 37 because such logic diagrams and equations are similar to those shown in Fig. 10 and Fig. 11 and are easily obtainable from the state diagrams shown in Figs. 6 and 8.

Each processor module 33 (Fig. 1) in the multiprocessor system is connected to both interprocessor buses 35 (Fig. 1) and is capable of communicating with any processor module including itself over either bus. For each block data transfer, one processor module is the source or sender and another is the destination or receiver.

Transmission of data by a processor module over one of the interprocessor buses is initiated and accomplished under software control by means of the SEND instruction.

In the SEND instruction the microprogram 115 (Fig. 2) and the CPU microprocessor 113 (Fig. 2) interacts with the shared outqueue section 67 of the interprocessor control 55 to read a data block from memory 101, to break it up into packets, to calculate packet check sum words, and to transmit the block one packet at a time over a bus to the receiving processor module. Parameters supplied to the SEND instruction specify the number of words in the block, the starting address of the block, which bus to use, the destination processor, and a maximum initial timeout value to wait for the outqueue 67 (Fig. 2) to become available.

The SEND instruction terminates only after the entire block has been transmitted; thus sending a block is a single event from the software viewpoint. However, the SEND instruction is interruptable and resumable, so that response of the operating system to other events is not impaired by the length of the time required to complete a SEND instruction.

(20)

Receiving of data by a processor module over the interprocessor buses is not done by means of a software instruction, since the arrival times and sources of data packets cannot be predicted. The receiving of data is enabled but cannot be initiated by the receiver.

The CPU microprocessor 113 takes time out from software instruction processing as required to execute the BUS RECEIVE microprogram 115. This microprogram takes the received data packet from one of the inqueue sections 65 (Fig. 2) of the interprocessor control 55, stores the data into a memory buffer, and verifies correct packet check sum.

Reassembly of received packets into blocks is accomplished using the Bus Receive Table 150 (BRT) in memory. The BRT contains 32 two-word entries, corresponding to the two buses from each of the sixteen processor modules possible in one specific implementation of the multiprocessor system. Each BRT entry corresponding to a bus and a sender contains an address word and a count word. The address word specifies into which buffer in the System Data area incoming data from that sender is to be stored. The count word specifies how many data words remain to complete the block transfer from that sender.

As each data packet is received, the CPU microprocessor 113 suspends processing of software instructions, and the bus receive microprogram 115 is activated. This microprogram reads the address and count words from the sender's BRT entry, stores the data packet into the specified area, verifies correct packet check sum, and restores adjusted values of the address and count words into the BRT entry. If the packet caused the count to reach zero or if the packet contained incorrect check sum, the bus receive microprogram sets a completion interrupt flag to signal termination of the data block to the software. The CPU microprogram then resumes software instruction processing at the point where it left off with no disturbance except delay to the currently executing program.

It is an important feature that data blocks from several senders can all be assembled concurrently by a receiving processor module from data packets received in any sequence. This interleaved assembly of blocks from packets is carried on transparently to the software executing in the receiver processor. Only successful block completions or erroneous transmissions cause the software to be interrupted.

It is also important that a time-sharing or time-slicing of the interprocessor bus hardware has been achieved in two areas.

First, each interprocessor bus and associated bus controller allow packets to be transmitted between any sender and receiver as required. The circular polling by a bus controller to identify a requesting sender ensures that all processor modules have an equal opportunity to send over that bus. Each bus provides a communication path which is shared in time in an unbiased way by all processor modules.

Secondly, each inqueue section 65 of the interprocessor control 55 of a processor module is shared in time by incoming packets from several senders. That is, the inqueue logic and storage of a processor is not dedicated to a single sender for the duration of a block transfer. Instead, each packet received is correctly directed into memory by the BRT entry corresponding to its sender and bus. Data blocks from several senders are assembled correctly in the receiver's memory independently of the order in which the senders make use of the bus.

A processor module has two ways of controlling its ability to receive packets over the X bus or the Y bus.

First, there is a bit in the CPU's interrupt MASK register corresponding to each interprocessor bus. When the MASK bit is on, micro-interrupts for that bus are allowed. Micro-interrupts (activation of the BUS RECEIVE microprogram) occur when the Processor Empty state logic 101 (Fig. 5) of an inqueue section 65 reaches the MICRO-INT state after a packet has been received into an inqueue buffer. If the MASK bit is off when a packet is received, the micro-interrupt and subsequent processing of the packet into memory will be deferred until the MASK bit is set on by a software instruction.

Software operations such as changing a BRT entry are performed with micro-interrupts disabled to avoid unpredictable results. No packets are lost while micro-interrupts are disabled. The first packet received will be held in the inqueue buffer until the micro-interrupt is enabled. Subsequent packet transfers while the inqueue buffer is full are rejected since the Bus Fill state 93 logic will be in the FULL state and thus unable to assert RECEIVE ACKNOWLEDGE in response to SELECT.

A second means of controlling its ability to receive packets over the bus is the action taken by a processor module after an X bus or Y bus receive completion interrupt (activation of an operating system interrupt handler).

When a check sum error is detected in a received packet or when the BRT word count remaining in a data block reaches zero as a packet is stored into memory, the BUS RECEIVE microprogram sets the X bus or Y bus completion interrupt flag. Otherwise, the microprogram issues the RINT signal (see Fig. 8) to the inqueue Processor Empty state logic 101 to allow another packet to be

received. When the completion flag is set, however, the RINT signal is not issued.

It is thus the responsibility of the bus receive completion software interrupt handler to issue the RINT signal (by means of an RIR software instruction) to reenable the inqueue 65. Until this occurs, the inqueue Bus Fill state logic 93 remains in the FULL state and no additional packets will be received.

The completion interrupt signal can therefore designate either a block data transfer that has been sent and received without error, or it can designate a partial transfer in which a check sum error is detected, and in which partial transfer of the completion interrupt is generated as a result of the check sum error detected. In the latter case, the sender continues to send the data block but the receiver discards the data block after the check sum error has been detected. This error shows up in the bus receive table (BRT) count word as a negative value. This will become more apparent from the description of the operation which follows.

The SEND instruction is an instruction that requires four parameter words in the CPU register stack.

The first of the four parameter words is a count of the number of words to be transferred. This value must match the number expected by the BRT in the receiver processor module if the transfer is to complete successfully.

The second parameter word is the address, minus one, in the System Data area in the sender processor's memory where the data to be transferred is located.

The third parameter word is a timeout value allotted to completing a single packet (fifteen data word) transfer. The timeout period is restarted for each packet transferred by the SEND instruction.

The fourth parameter word specifies the bus (whether the X bus or the Y bus) to be used and specifies the receiver processor module. The high order bit of the parameter specifies the bus and the low order four bits, in one specific implementation of the invention, specify the number of the receiver processor module.

At the completion of a SEND instruction, there are two possible conditions.

The first condition is that a packet timeout occurred and the remaining packets were not transmitted and the instruction was terminated at that point. In this event the remaining packets of the block are not transmitted.

The second condition is an indication that a successful data block transfer has been completed.

Thus, in initial summary of the SEND operation, the SEND instruction fills the outqueue buffer 69 (Fig. 4) with fifteen data words, appends an odd-parity check sum, and signals the bus controller 37 that it has a packet ready for transmission. After each sixteen word packet is transmitted, execution of the SEND instruction resumes at the point where it left off. If the last packet of the block has less than fifteen words, the remaining words are filled in with zeros. The instruction terminates when the last packet is transmitted.

Fig. 5 shows the logic diagram and Fig. 7 shows the state diagram for the send hardware.

The first action of the SEND instruction sequence is to issue the S/RECEIVE signal to the processor fill state logic 73 (Fig. 4) and to supply on the M Bus (Fig. 4) the receiver processor number to the receive register 71. Simultaneously, the pointer of the outqueue pointer 79 is set in accordance with the high order bit of the M Bus to connect the outqueue 67 to either the X bus or the Y bus.

The store receive (S/RECEIVE) signal causes the processor fill state logic 73 (which is initially in the empty state as shown in Fig. 7) to advance to the FILL state as shown in Fig. 7. This state transition causes the receive register 71 (Fig. 4) to be loaded with the receiver processor number.

At this point the outqueue section 67 is ready for the data packet to be loaded into the outqueue buffer 69. Now, up to fifteen words are read from memory and are stored, by means of the M bus (Fig. 4), into the outqueue buffer 69. The store outqueue signal causes each word on the M bus to be written into the outqueue buffer 69 in a location specified by the outqueue counter 77. Each store outqueue signal also causes the outqueue counter 77 to be advanced by one.

As the words are being read from memory, the address word is being incremented by one, and the count of the words to be sent is being decremented by one. If the count reaches zero before fifteen words are read from memory, the remainder of the outqueue buffer is filled with zeros to pad out the data packet.

In addition, as the words are being loaded into the outqueue buffer 69, the microprogram 115 (Fig. 2) is calculating a modulo-two sum of the data words. After the fifteenth data word has been loaded, this odd checksum word is loaded into the sixteenth location of the outqueue buffer 69.

At this time the outqueue counter 77 has a value of count 15 and this value, in combination with the store outqueue signal, causes the processor fill state logic 73 to advance from the FILL state to the FULL state as shown in Fig. 7.

At this point the microprogram 115 has completed loading of the data into the outqueue 69. The microprogram now waits for the packet to be transmitted by testing for

22

occurrence of the ready (RDY) signal shown in Fig. 7.

While waiting for the packet to be transmitted, the microprogram 115 increments a timer; and if the timer runs out or expires before the ready (RDY) signal is asserted, the microprogram issues the clear outqueue (CLOQ) signal to the processor fill state logic 73 (see Fig. 4). This causes the processor fill state logic 73 to return to the empty state as shown in Fig. 7, and the microprogram then terminates the SEND instruction with the time out indication.

In normal operation, the FULL state of the processor fill state logic 73 qualifies the bus empty state logic 75 to advance from the IDLE state to the SYNC state shown in Fig. 7. Next, the SYNC state automatically advances to the SEND state, and this state causes the SEND REQUEST signal to be issued to the bus controller 37. The SEND REQUEST signal initiates a packet transfer sequence described earlier.

As described earlier, when the sender processor module has been identified by the bus controller 37 by polling, and when the receiver processor module has accepted the packet transfer by means of the RECEIVE ACKNOWLEDGE signal, the data packet is gated from the outqueue buffer 69 through the outqueue pointer 79 to one of the data buses 57 for loading into the inqueue of the receiver processor module.

As the sixteenth word is gated to the bus, the value of the outqueue counter count 15, in combination with the SEND COMMAND signal and the SENDER SELECT signal causes the SEND state of the bus empty state logic 75 to advance to the DONE state.

The DONE state qualifies the FULL state of the processor fill state logic 73 (as shown by the dashed line arrow going from the DONE state to the indicated transition from the FULL state in Fig. 7) to advance to the WAIT state.

Next, the WAIT state qualifies the DONE state to advance to the IDLE state as illustrated by the state diagram in Fig. 7.

Finally, the IDLE state qualifies the WAIT state to advance to the EMPTY state as also indicated in the state diagram of Fig. 7.

The EMPTY state, of the processor fill state logic 73, provides the READY indication to the microprogram 115.

If the packet just transmitted was the last packet in the specified data block, the SEND instruction is terminated and the successful block transfer indication is given.

If the packet transmitted is not the last packet in a data block, then the sequence described above is repeated until all words in the block have been transmitted, or until a timeout error has occurred.

The SEND instruction is interruptable and resumable; however, the SEND instruction is only interruptable between packets; and the interruption of the SEND instructions has no effect on the data transmitted.

Thus, by means of a single software instruction (the SEND instruction) a data block of up to 32,767 words is transmittable from a sender processor module to a receiver processor module, and accuracy of the transmission is checked by the packet check-sum. Also, the transmission occurs at a high data transfer rate, because the buffering provided by the outqueue buffer 69 of the sender processor module enables the transfer to be made at interprocessor bus speed independent of the memory speed of the sender processor module. This allows efficient use of this communication path between a number of processor modules on a time slicing basis.

As noted above, there is no instruction for receive.

For a processor module to receive data over an interprocessor bus, the operating system in that processor module must first configure an entry in the bus receive table (BRT). Each BRT entry contains the address where the incoming data is stored and the number of words expected.

While the sender processor module is executing the send instruction and sending data over the bus, the bus receive hardware and the microprogram 115 in the receiver processor module are storing the data away according to the appropriate BRT entry (this occurs interleaved with software program execution).

When the receiver processor module receives the expected number of words from a given sender, the currently executing program is interrupted, and that particular bus transfer is completed.

Fig. 5 shows the logic diagram and Fig. 8 shows the state diagram for the bus receive hardware.

As previously pointed out, there are identical X and Y inqueue sections 65 in each processor module for the X bus and the Y bus. Only one of the inqueue sections will therefore be referred to the description which follows.

After initial reset of a processor module, or after a previous receive operation, the RESET state of the processor empty state logic 101 advances to the READY state. The READY state qualifies the SYNC state of the bus fill state logic 93 to advance the logic to the ACKNOWLEDGE state.

In this ACKNOWLEDGE state the inqueue section 65 returns RECEIVE ACKNOWLEDGE to the bus controller 37 in response to a SELECT 63 (see Fig. 2) of that processor module 33. This indicates the readiness of the X inqueue section 65 to receive the data packet.

In the packet transfer sequence (described in detail above) the combination of the SELECT of that processor module and the RECEIVE COMMAND signal qualify the ACKNOWLEDGE state of the bus fill state logic 93 and to advance to the RECEIVE state.

At this state transition the sender register 95 (Fig. 5) is loaded with the number of the sending processor module.

In the RECEIVE state the data packet is loaded from the data bus to the inqueue buffer 97 under control of the inqueue counter 99.

As the sixteenth word of the packet is loaded, it causes the RECEIVE state to advance to the FULL state (see Fig. 8).

Now the FULL state qualifies the READY state of the processor empty state logic 101 to advance to the MICROINTERRUPT state as shown in Fig. 8. The MICROINTERRUPT state presents an INQUEUE FULL state to the CPU interrupt logic. This INQUEUE FULL signal causes a microinterrupt to occur at the end of the next software instruction if the MASK bit corresponding to that bus is on.

The bus receive microprogram 115 activated by the interrupt first of all issues a LOCK signal (see Fig. 5) to the processor empty state logic 101. This causes the MICROINTERRUPT state of the processor empty state logic 101 to advance to the DUMP state.

The LOCK signal also selects either the X inqueue or the Y inqueue; subject, however, to the condition if both inqueues are full and enabled, the X queue is selected.

Next, the microprogram 115 issues the K/SEND signal which causes the sender register 95 contents to be gated to the K bus (as shown in Fig. 5) to obtain the packet sender's processor number.

Using this processor number, the microprogram 115 reads the sender processor's BRT entry to obtain the address and count words.

If the count word is zero or negative, the packet is discarded; and in this case, the microprogram 115 issues a RINT signal which causes the processor empty state logic 101 to advance from the DUMP state to the RESET state as shown in Fig. 8. In this event there is no further action. The microinterrupt is terminated, and software instruction processing is resumed.

If the count is positive, the microprogram 115 reads words from the inqueue buffer 97 to the K bus by means of the K/INQUEUE signal as shown in Fig. 5.

With each occurrence of the K/INQUEUE signal, the inqueue counter 99 is incremented to scan through the inqueue buffer 97.

As each data word is read from the inqueue buffer 97, the count word is decremented, the memory address word is incremented, and the data word is stored into memory.

If the count word reaches zero, no more words are stored in memory, a completion interrupt flag is set, and the sender processor number is saved in a memory location. In that event the fill state bus logic 93 stays in the FULL state until cleared by a software RIR instruction.

Thus, when a data block has been completely received, the count word will contain a value between minus 14 and zero. After the completion interrupt occurs, no further transfers to the processor over the bus which cause the interrupt are permitted until the inqueue is cleared with an RIR instruction.

As the data words are stored into the memory, a modulo-two sum of packet data is calculated.

If the check sum is bad, the word count in the BRT entry is set to minus 256, a completion interrupt flag is set, and the sender processor number is saved in memory. As above, the bus fill state logic 93 stays in the FULL state until cleared by an RIR instruction.

If the count word does not reach zero, and the check sum is good, the bus receive microprogram 115 issues the RINT signal to the processor empty state logic as shown in Fig. 5 which causes the DUMP state of the processor empty state logic 101 to advance to the RESET state as shown in Fig. 8.

The RESET state of the logic 101 qualifies the bus fill state logic 93 to advance from the FULL state to the SYNC state as also shown in Fig. 8.

At this point, the logic has been returned to the state it was in before the packet was received, thus enabling the receipt of more packets.

These packets may be from the same sender, completing that data block, or the packets may be from some other sender.

This completes the action of the bus receive microprogram 115 and the microprocessor 113 resumes processing of software instructions.

When a bus receive completion interrupt has occurred, the software interrupt handler obtains the sender processor number from the memory location where that number was saved, and the software interrupt handler can then detect if a check sum error occurred by examining that sender processor's bus receive table count word.

In the case of a transmission error, the count word has been set to minus 256. Otherwise, the count word will contain a value between minus fourteen and zero.

As mentioned above, it is thus the responsibility of the bus receive completion software interrupt handler to issue the RINT

signal (by means of an RIR software instruction) to reenable the inqueue 65.

In summary on the receive operation, just as the sending of a data block by a sender processor module is viewed by software as a single event, the receiving of data by a receiver processor does not cause a software interrupt of the receiver processor module until the entire data block has been received or until an error has has occurred. Also, the inqueues 65 serve as buffers to allow the transmission of data to occur at bus transmission rates while allowing the storing of data into memory and the checking of the data to occur at memory speed. This ability to use the high transmission rate on the bus insures adequate bus bandwidth to service a number of processor modules on a time slicing basis. Finally, the provision of a check sum word in each data packet provides a means in the receiver processor module for checking the accuracy of the data received over the multiprocessor communication path.

Information sent over the interprocessor bus is sent under the control of the operating system and is sent from one process in one processor module 33 to another process in another processor module 33. A process (as described in detail above in the description of the Multiprocessor System) is a fundamental entity of control in the software system; and a number of processes coexist in a processor module 33. The information sent over the interprocessor bus between processes in different processor modules consists of two types of elements, control packets and data.

The control packets are used to inform the receiving processor module 33 about message initiations, cancellations, and data transfers.

In this regard it should be noted that, while the interprocessor buses 35 interconnect the processor modules 33, a process within a particular processor module 33 communicates with another process or with other processes within another processor module 33 through a method of multiplexing the interprocessor bus 35. The bus traffic between two processor modules 33 will therefore contain pieces of interprocess communications that are in various states of completion. Many interprocess communications are therefore being interleaved on an apparently simultaneous basis.

The hardware is time slicing the use of the interprocessor bus 35 on a packet level, and multiple processes are intercommunicating both within the processor modules 33 and to the extent necessary over the interprocessor buses 35 in message transactions which occur interleaved with each other. Under no circumstances is an interprocessor bus 35 allocated to any specific process-to-process communication.

Data information is sent over the interprocessor bus in one or more paekets and is always preceded by a control packet and is always followed by a trailer packet.

The control packet preceding the data packets is needed because a bus is never dedicated to a specific message, and the control packet is therefore needed to correctly identify the message and to indicate how much data is to be received in the message.

This information transfer (control packet, data information, trailer packet) is made as an indivisible unit once it is started. The sender processor module sends the data block as an individual transmission (consisting of some number of data packets) and sends the trailer packet as an individual transmission; and only then is the sender processor module able to send information relating to another message.

The trailer packet serves two purposes.

First of all, if there is an error during a data transmission (and therefore the rest of the data block must be discarded), the trailer packet indicates the end of the block.

Secondly, if the sender attempts to send too much data (and again the block must be discarded), the trailer packet provides a means for recognizing data has been transmitted and the data transmission has completed.

The information transmitted is either duplicated over different paths (so that it is insured that the information will get to the receiver) or a receiver acknowledgment is required (so that the information is repeated if necessary). Any single bus error therefore cannot cause information to be lost, and any single bus error will not be seen by the two processes involved.

The bus receive software interlocks with the bus receive hardware (the inqueue section 65 shown in Fig. 2) by controlling the transfer of information from the inqueue into the memory 107.

This allows such operations as changing the bus receive table information to be done without race conditions (synchronization problems).

Once bus receive table information has been updated, the interlock is removed by clearing the previous completion interrupt and by reenabling the bus receive microinterrupts by setting on the bus mask bit in the mask register.

This does two things. It allows the inqueue hardware to accept a packet into the inqueue, and it also enables the bus receive microprogram to transfer that information from the inqueue into memory.

The hardware/software system is so constructed that no information is lost on a system power failure (such as a complete failure of AC power from the mains) or on a line transient that causes a

momentary power failure for part of the system.

This hardware/software system coaction includes a power warn signal (see line 337 of Fig. 30) supplied to the inqueue section 65 (see Fig. 2) so that, at most, one further packet of information can be loaded into the inqueue after the receipt of the power warn signal.

The software action in this event includes a SEND instruction to force the inqueues to be full. The net effect is to insure that no transmissions are completed after the processor module 33 has received its power warn signal, so that the state of every transfer is known when logic power is removed.

The interprocessor buses 35 are used by the operating system to ascertain that other processor modules in the system are operating. Every N seconds, each of the processor modules 33 sends a control packet to each processor module 33 in the system on each interprocessor bus 35. Every two N seconds, each processor module 33 must have recived such a packet from each processor module 33 in the system. A processor module that does not respond is considered down. If a processor module does not get its own message, then that processor module 33 knows that something is wrong with it, and it will no longer take over I/O device controllers 41.

Fig. 42 diagrammatically illustrates how a particular application program can run continuously even though various parts of the multiprocessor system can become inoperative.

Each of the separate views shown in Fig. 42 illustrates a multiprocessor system configuration which consists of two processor modules 33 connected by dual interprocessor buses 35 (indicated as an X bus and a Y bus), a device controller 41 controls a number of keyboard terminals, and another device controller 41 which controls a disc.

The individual views of Fig. 42 indicate various parts of the multiprocessor system rendered unserviceable and then reintroduced into the multiprocessor system in a serviceable state.

The sequence starts with the upper left hand view and then proceeds in the order indicated by the broad line arrows between the views. The sequence thus goes from the condition indicated as (1) Initial State to (2) CPU 0 Down to (3) CPU 0 Restored to (4) CPU 1 Down to (5) CPU 1 Restored (as indicated by the legends above each individual view).

In the initial state of the multiprocessor system shown in the view entitled "Initial State" at the upper left hand corner of Fig. 42, one copy (PA) of the application program is active. This copy makes a system call to create the copy PB as a backup to which the application program PA then passes information. All of the I/O is taking place by way of the processor module 0. In this initial state either interprocessor bus 35 may fail or be brought down (as indicated by the bars on the X bus) and can be then reintroduced into the multiprocessor system without producing any effect on the application program PA.

In the next view (the view intitled "CPU 0 Down") the processor module 0 is rendered unserviceable. The multiprocessor system informs the application program PA that this has happened, and the application program PA no longer tries to communicate with the program PB. All of the I/O is switched by the multiprocessor system to take place by way of the processor module 1, and the application program continues to service the terminals nonstop without interruption over the I/O bus 39 connecting the processor module 1 with the device controllers 41 (as indicated by the solid line arrow on the right hand I/O bus 39).

In the next state of operation of the multiprocessor system, as illustrated in the center top view of Fig. 42 and entitled "CPU 0 Restored", the processor module 0 is now brought back into service by way of a console command. The processor module 0 is reloaded with the multiprocessor system from the disc by way of the processor module 1. The application program PA is informed that processor module 0 is now serviceable and the application program PA tells the multiprocessor system to create another copy of the application program in the processor module 0. This other copy is designated as PC. The terminals continue in operation without interruption.

Next, the processor module 1 is rendered inoperative, as illustrated in the view entitled "CPU 1 Down". The application program PC is informed of this fact by the multiprocessor system and the application program PC takes over the application. The multiprocessor system automatically performs all of the I/O by way of the processor module 0. The terminals continue in operation without interruption.

Finally, as indicated by the top right hand view of Fig. 42 entitled "CPU 1 Restored", the processor module 1 is rendered operable by way of a console command and is reloaded with the multiprocessor system from the disc by way of the processor module 0. The application program PC is informed that the processor module is now available, and it tells the multiprocessor system to create another copy of itself (application program PD) in the processor module 1. All elements of the multiprocessor system are now operable.

During the whole of this time both interprocessor buses and both processor modules had been rendered unserviceable and reintroduced into the system, but the application

program and the terminals continued without a break.

It is an important feature of the multiprocessor system that not only can the application program continue while something has failed, but also that the failed component can be repaired and/or replaced while the application program continues. This is true not only for the processor modules and interprocessor buses but also for all elements of the multiprocessor system, such as power supplies, fans in the rack, etc. The multiprocessor system 31 thus is a true continuously operating system.

THE INPUT/OUTPUT SYSTEM AND DUAL PORT DEVICE CONTROLLER:

The multiprocessor system 31 shown in Fig. 1 includes an input/output (I/O) system and dual port device controllers 41 as noted generally above.

The general purpose of the I/O system is to allow transfer of data between a processor module 33 and peripheral devices.

It is an important feature of the present invention that the data transfer can be accomplished over redundant paths to insure fail soft operations so that a failure of a processor module 33 or a failure of a part of a device controller 41 will not inhibit transfer of data to and from a particular peripheral device.

Each device controller 41 has dual ports 43 and related structure which, in association with two related I/O buses 39, permit the redundant access to a peripheral device as will be described in more detail below.

The I/O system of the present invention also has some particularly significant features in terms of performance. For example, one of the performance features of the I/O system of the present invention is the speed (bandwidth) at which the input/output bus structure operates. The device controllers 41 collect data from peripheral devices which transmit data at relatively slow rates and transmit the collected data to the processor modules in a burst multiplex mode at or near memory speed of the processor modules 33.

As illustrated in Fig. 1, each processor module 33 is attached to and handles a plurality of individual device controllers 41; and this fact makes it possible for each device controller 41 to be connected (through dual ports 43) to more than one processor module 33 in a single multiprocessor system.

With reference now to Fig. 12 of the drawings, each processor module 33 includes, in addition to the interprocessor control 55 noted above, a central processor unit (CPU) part 105, a memory part 107 and an input/output (I/O) channel part 109.

As illustrated in Fig. 12 and also in Fig. 1, each device controller 41 controls one or more devices through connecting lines 111 connected in a star pattern, i.e. each device independently connected to the device controller.

In Fig. 12 a disc drive 45 is connected to one device controller 41 and a tape drive 49 is connected to another device controller 41.

With continued reference to Fig. 12, each CPU part 105 includes a microprocessor 113. A microprogram 115 is associated with each microprocessor 113. A part of the microprogram 115 is executed by the microprocessor 113 in performing I/O instructions for the I/O system. The I/O instructions are indicated in Fig. 12 as EIO (execute I/O), IIO (interrogate I/O), HIIO (interrogate high priority I/O); and these instructions are illustrated and described in greater detail below with reference to Figs. 15, 16 and 17.

The microprocessor 113 has access to the I/O bus 39 by way of the I/O channel 109 by a collection of paths 117 as illustrated in Fig. 12.

With continued reference to Fig. 12, the I/O channel 109 includes a microprocessor 119, and a microprogram 121 is associated with the microprocessor 119.

The microprogram 121 has a single function in the multiprocessor system, and that function is to perform the reconnect and data transfer sequence illustrated in Fig. 16 (and described in more detail below).

The I/O channel 109 of a processor module 33 also includes (as shown in Fig. 12) data path logic 123.

As best illustrated in Fig. 13, the data path logic 123 includes a channel memory data register 125, an input/output data register 127, a channel memory address register 129, a character count register 131, an active device address register 133, a priority resolving register 135 and parity generation and check logic 137.

The path 117 shown in Fig. 12 includes two buses indicated as the M bus and the K bus in Fig. 13.

The M bus is an outbus from the microprocessor 113 and transmits data into the input/output data register 127.

The K bus is an inbus which transmits data from the data path logic 123 into the microprocessor 113.

With reference to Fig. 12, a path 139 connects the data path logic 123 and the memory subsystem 107.

This path 139 is illustrated in Fig. 12 as including both a hardware path 139A and two logical paths 139B and 139C in the memory subsystem 107 of a processor module 33.

Logical paths 139B and 139C will be described in greater detail below in connection with the description of Fig. 16.

The hardware path 139A includes three branches as illustrated in Fig. 13.

A first branch 139A-1 transmits from

memory into the channel memory data register 125.

A second path 139A-2 transmits from the channel memory address register 129 to memory.

And a third path 139A-3 transmits from the input/output data register 127 to memory.

With reference to Fig. 12, the input/output channel of a processor module 33 includes a control logic section 141.

This control logic section 141 in turn includes a T bus machine 143 (see Fig. 13) and request lines RECONNECT IN (RCI) 145, LOW PRIORITY INTERRUPT REQUEST (LIRQ) 147, HIGH PRIORITY INTERRUPT REQUEST (HIRQ) 149 and RANK 151 (see Fig. 14).

The I/O bus 39 shown in Fig. 14 and Fig. 12 also includes a group of channel function lines 153, 157 and 159. See also Fig. 13. The TAG bus (T bus) 153 consists of four lines which serve as function lines, and there are three lines SERVICE OUT (SVO) 155, SERVICE IN (SVI) 157, and STOP IN (STI) 159 which serve as handshake lines as indicated by the legends in Fig. 14.

As shown in Fig. 14 and Fig. 12, the I/O bus 39 also includes a group of data lines 161, 163, 165, 167 and 169.

The DATA BUS lines 161 and PARITY 163 are bi-directional and serve as data lines and as indicated in Fig. 14, there are sixteen DATA BUS lines 161 and one PARITY line 163 in this group.

The lines END OF TRANSFER (EOT) 165, PAD OUT (PADO) 167 and PAD IN (PADI) 169 serve as data status lines, and indicate special conditions that may occur on the data lines 161 and 163 from time-to-time.

Finally, the I/O bus 39 includes a reset line (IORST) 171 as also shown in Fig. 14 and in Fig. 12.

Each T bus command illustrated in Fig. 18 requires some specific format on the data bus 161 while a T bus command is valid. This specific data bus format is illustrated for the T bus functions Load Address and Command (LAC) and Read Device Status (RDST) shown in Fig. 18, for the preferred embodiment.

In the case of the T bus function LAC, the data or field transmitted on lines Ø to 5 of the data bus 161 specify the operation to be performed; the field transmitted on lines 8 to 12 of the data bus specify the device controller 41 (or more precisely the port 43 of that device controller which is attached to the data bus 161) to which the command is addressed; and the field transmitted on data bus lines 13 to 15 specify which device attached to the device controller is to be operated on by that device controller 41 in response to this command.

In the case of the T bus function RDST, data bus bits Ø, 1, 2 and 3 indicate ownership error, interrupt pending, device busy, and parity error respectively. Bits 4 to 15 return device dependent status.

The functions on the T bus are transmitted in three sequences, shown in Figs. 15, 16 and 17 and described in detail below.

Each T bus function is asserted by the channel and a handshake sequence is performed between the channel 109 and the device controller 41 using the handshake lines 155, 157 and 159 to acknowledge receipt of the T bus function. Control of the T bus and handshake is the function of the T bus machine 143 in Fig. 13.

Fig. 28 is a timing diagram showing the operation of the handshake between the I/O channel 109 and the ports 43.

As illustrated in Fig. 28, line 155 transmits the service out signal (SVO) and line 157 transmits the service in signal (SVI).

The channel clock cycle is shown in vertical orientation with the SVO and SVI signals.

As illustrated in Fig. 28, the service in (SVI) signal is not synchronized with the channel clock and may be asserted at any time by the device controller in response to a service out signal from the I/O channel 109.

Before asserting service out (SVO), the channel 109 asserts the T bus function and, if required, the data bus.

The channel then asserts a service out signal as indicated by the vertical rise 279 in Fig. 28; and, SVO remains true until the device controller responds with service in (SVI) (281), acknowledging the channel command; SVI remains true until the channel drops SVO.

When the device controller 41 asserts the service in (SVI) signal, the channel 109 removes the service out (SVO) signal (as shown by the vertical drop 283 in Fig. 28) in a time period typically between one and two clock cycles; and in response, the device controller drops service in (SVI) as shown by the vertical drop 285 in Fig. 28.

When the device controller drops the service in (SVI) signal, the channel 109 is free to reassert a service out signal (SVO) for the next transfer; however, the channel will not reassert SVO until SVI has been dropped.

The arrows 281A, 283A and 285A in Fig. 28 indicate the responses to the actions 279, 281, 283 respectively.

The handshake is completed at the trailing edge of the vertical drop 285 as shown in Fig. 28.

On an output transfer, the interface data register 213 of the controller accepts the data at the leading edge of service out (vertical rise 279) and transfers the data to the control part of the device controller 187 at the trailing edge of the service out (the vertical drop 283).

On an input transfer the channel 109 accepts data from the device controller at the trailing edge of service out (the vertical drop 283).

Thus, a two line handshake is used to interlock transfer of information between the channel 109 and its device 41, since they act asynchronously.

This is the general handshake condition, indicated as handshake 2L in Figs. 15, 16 and 17.

In addition, two special handshake considerations occur, when appropriate.

First, channel commands used to select a device controller are not handshaken by SVI, since no single device controller is selected during this time.

These commands include (as shown in Fig. 18):

SEL—Select;
LAC—Load Address & Command;
HPOL—Hi Priority Interrupt Poll;
LPOL—Lo Priority Interrupt Poll; and
RPOL—Reconnect Interrupt Poll.

Also, commands used to terminate a sequence are not handshaken by SVI since they cause a selected device controller to deselect itself.

These commands include (as also shown in Fig. 18):

DSEL—De-Select;
ABTI—Abort Instruction (I/O); and
ABTD—Abort Data.

For all of the commands noted above which are not handshaken, the channel asserts SVO (155) for a given period of time (e.g., two clock cycles) and then the channel removes SVO. This type of handshake is referred to as Handshake 1L in Figs. 15, 16 and 17.

Second, data transfer is handshaken normally except that when a device controller wishes to signal that it does not require further service, it returns stop-in (STI) instead of SVI. When SVO is next dropped by the channel, the port deselects itself. STI otherwise handshakes in the same manner as SVI.

As a further condition on all handshakes, when the channel prepares to assert SVO, it initiates a timer (part of T bus machine 143 in Fig. 13) which times out and posts an error if the next handshake cycle is not initiated and completed within the period of time set by the timer. If the timer times out, an error is posted at the appropriate point in the sequence, and either ABTI (EIO, IIO or HIIO sequence) or ABTD (reconnect sequence) is sent to the device controller 41 (see discussions of Figs. 15, 16 and 17).

Fig. 29 shows the logic for the handshake shown in Fig. 28. The logic shown in Fig. 29 is part of the T bus machine 143 shown in Fig. 13. The logic shown in Fig. 29 is the logic which is effective for the general handshake condition noted above.

The logic shown in Fig. 29 includes a service out flip-flop 287 and a service in synchronization flip-flop 289. As illustrated by the dividing lines and legends in Fig. 29, the flip-flops 287 and 289 are physically located within the channel 109.

The device controller 41 includes combinational logic 291 and a transmitter 293 which transmits a service in signal (SVI) back to the D input of the flip-flop 289.

The functioning of the logic shown in Fig. 29 is as follows.

The channel 109 asserts service out by turning on the J input of the flip-flop 287; and when the next clock cycle starts, the service out signal is transmitted by a transmitter 295 to the device controller.

When the combinational logic 291 in the device controller is ready it enables the transmitter 293 to return the service in signal (SVI) to the flip-flop 289. This completes the handshake.

Turning now to the dual port device controller, as illustrated in Fig. 19, each of the dual ports 43 in a device controller 41 is connected by a physical connection 179 to interface common logic 181 (shown in more detail in Fig. 21) and each of the ports 43 is also associated through a logical connection 183 to the interface common logic 181 as determined by an ownership latch 185.

As shown by the connecting line 180 in Fig. 19, the interface common logic 181 is associated with the control part 187 of the device controller 41. The control part 187 of the device controller includes a buffer 189.

The dual ports 43 shown in block diagram form in Fig. 19 (and in more detail in Fig. 23) are important parts of the multiprocessor system of the present invention because the dual ports provide the failsoft capability for the I/O system.

The ports 43 and related system components are structured in such a way that the two ports 43 of one device controller 41 are logically and physically independent of each other. As a result, no component part of one port 43 is also a component of the other port 43 of a particular device controller 41; and no single component failure (such as an integrated circuit failure) in one port can affect the operation of the other port.

Each port 43 functions to interface (as indicated by the legend in Fig. 19) a processor module 33 with a device controller, and ultimately with a particular device, through the device controller 41. The port 43 is the entity that communicates with the processor module and communicates with the control part of the device controller 187 (conditional on the state of the ownership latch 185).

That is, the port itself makes the connection to a processor module (dependent upon instructions received from the I/O channel

109 as discussed in more detail below) by setting its select bit 173.

Each of the individual ports 43 in a particular device controller 41 can be connected independently to a processor module 33 and at the same time as the other port in that device controller is connected to a different module. However, the ownership latch 185 establishes the logical connection between the control part of the device controller and one of the dual ports 43 so that only one port has control of the device controller at any one point in time.

The decode logic determines what function is being transmitted on the T bus 153 at any particular time.

The control logic combines T bus functions to perform specific port functions, for example, set select bit, clear select bit, read interrupt status.

The functioning of the control logic is illustrated in the logic equations set out in Fig. 27.

When a connection sequence (to be described later in reference to Figs. 15, 16 and 17) is transmitted over the I/O bus 39, one of the ports 43 (and only the one port 43 in a device controller 41 attached to that I/O bus 39) connects (in a logical sense) to the bus 39 by setting its select bit 173.

This logical connection is determined by part of the data transmitted in that connection sequence. When connected, that particular port 43 subsequently responds to channel protocols in passing information between the channel and the control part of the device controller. The device address comparator 193 is the component part of the port 43 that determines the port's unique address.

The device address comparator 193 determines the unique address for a particular port 43 by comparing the device address field on the data bus 161 during a LAC T bus function, with device address jumpers associated with a particular port 43. When the address transmitted by the channel 109 matches the address determined by the jumpers on a particular port 43, the term ADDCOMP (see Fig. 27) is generated and the select bit 173 for that port is set (assuming that the other conditions set out in Fig. 27 allow the select bit to be set). The port 43 then responds to all T bus operations until the sequence terminates by clearing the select bit.

The abbreviations used in Fig. 27 include the following:

Add Comp—Address Compare (Device Address);

PAROKFF—Parity OK Flip-Flop;

SEL—Select;

OWN—Ownership; and

SELBIT—Select Bit.

The parity check register 177 is related to the parity generator and check logic 137 of Fig. 13 in that on output the parity generator logic 137 generates the parity to be checked by the parity checker 177 of the port 43, and this parity must check or the operation will be aborted by the I/O channel 109 of the processor module 33. On input, the interface common logic 181 generates parity to be checked by the channel parity check logic 137 in a similar fashion.

As shown in Fig. 24, the parity check is started before data is loaded into the register, and the parity check is continued until after the data has been fully loaded into the register. That is, the parity on the D bus is checked by the port parity register whenever the channel asserts SVO with an output T bus function, and the parity is monitored for the duration of SVO to insure that the data on the D bus is stable for the duration of SVO while the port transfers the data into the data register 213.

This parity check occurs on each transaction in a T bus sequence; and if a parity error occurred during any transaction in the sequence, the error is returned as a status bit in response to a T bus function during a sequence. For example, in an EIO sequence (Fig. 18 and 15) the P bit return for RDST indicates that the port determined a parity error during the EIO sequence.

As illustrated in Fig. 18, the parity error bit is a bit number 3 on the D bus in response to a RDST function on the T bus.

If a parity error occurs at some time other than during an EIO sequence, the parity error is reported during the read interrupt status (RIST) T bus function similar to the manner described above for the RDST T bus function.

The parity error is cleared at the beginning of an EIO, HO, HIIO or reconnect sequence as shown in Fig. 24.

If a parity error is detected during any sequence it is recorded by the parity check register to be returned on the D bus in response to a RDST or RIST T bus function.

With continued reference to Fig. 20, the function of the enable latch 175 in the port 43 is to allow the I/O system to recover from a certain class of errors that would otherwise render inoperative both of the I/O buses 39 attached to a particular device controller 41. The enable latch 175 accomplishes this by not allowing the port 43 to place any signals on the I/O bus 39.

The enable latch 175 is cleared by a specific disable command. This is a load address and command (LAC) T bus function with a specific operation code transmitted on the D bus 161.

Once the enable latch 175 is cleared, this enable latch cannot be programmatically reset.

The port 43 includes a status multiplexer 195. The status multiplexer 195 returns the

ownership error memtioned above if the device controller 41 is logically connected to the other port 43 of that device controller, to indicate that the device controller is owned by the other port and commands to this port will be ignored.

The port 43 includes an interface transceiver 197 for each input line (i.e., SVI, STI, Data Bus, Parity, PADI, RCI, LIRQ, HIRQ) of the I/O bus 39 shown in Fig. 14. The transceivers 197 transmit data from the port 43 to the I/O channel 109 when the port select bit 173 is set and the T bus function on the T bus 153 requires that the device controller 41 return information to the channel. The transceivers 197 pass information from the data bus 161 into the port 43 at all times.

It is a feature of the present invention that the power on circuit 182 acts in association with the transceivers 197 to control the behavior of the transceivers as the device controller 41 is powered up or powered down, in a way which prevents erroneous signals from being placed on the I/O bus while power is going up or down. This feature is particularly significant from the standpoint of on line maintenance.

As shown in Fig. 20, each transceiver 197 comprises a receiver 198 and a transmitter 200.

The transmitter is enabled by an enable line 202.

There are several terms which are on the enable line 202. These include the select bit 173, a required input function on the T bus, and a signal from the PON circuit 182.

The signal from the PON circuit, in a particular embodiment of the present invention, is connected in a "wire or" connection to the output of the gate which combines the other terms so that the output of the PON circuit overrides the other terms by pulling down the enable line 202. This insures that the transmitter 200 (in one specific embodiment, an 8T26A or 7438) is placed in a high impedence state until the PON circuit detects that the power is at a sufficient level that the integrated circuits will operate correctly. The PON circuit output stage is designed to take advantage of a property of the specific transceiver integrated circuit used. On this particular type IC if the driver enable line 202 is held below two diode drops above ground potential, the transmitter output transistors are forced into the off state regardless of the level of power applied to the integrated circuit. This ensures that the driver cannot drive the bus.

This particular combination of features provides a mode of operation wherein the output of the integrated circuit is controlled as power comes up or goes down, whereas normally the output of an integrated circuit is undefined when power drops below a certain level.

This same circuit is used on the X and Y buses of the interprocessor bus system to control the transceivers and control signals generated by the interprocessor control 55. As indicated in Fig. 30, each central processor unit (CPU) 105 has a PON circuit 182 which is similar to the PON circuit 182 in the device controller. The PON circuits therefore control the transmitters for all of the device controllers 41 and all of the interprocessor controls 55.

Details of the power-on (PON) circuit are shown in Fig. 25 where the circuit is indicated generally by the reference numeral 182.

The purpose of the PON circuit is to sense two different voltage levels of the five volt supply.

If power is failing, the circuit senses the point at which power drops below a certain level which renders the logic in the device controller or CPU an indeterminate state or condition. At this point the circuit supplies signals to protect the system against the logic which subsequently goes into an undefinable state.

The second voltage level which the PON circuit will sense is a value that is perceived when power is coming up. This second level at which power is sensed will be greater than the first level by roughly 100 millivolts to provide hysteresis for the system to eliminate any conditions of oscillation.

The PON circuit stays in a stable condition after it senses one of the voltage conditions until it senses the other voltage condition, at which point it changes state. The state at which the PON circuit is in at any particular time determines the voltage level at which the transition to the other state will be made.

The power on circuit 182 thus presents a signal establishing an indication that the power is within predetermined, acceptable operating limits for the device controller 41. If the power is not within those predetermined, acceptable operating limits, the signal output of the power-on circuit 182 is used to directly disable the appropriate bus signals of the device controller 41.

The output of the PON circuit 182 is a binary output. If the output is a one, the power is within satisfactory limits. If the output of the PON circuit is a zero, this is an indication that the power is below the acceptable limit.

The power-on circuit 182 shown in Fig. 25 and to be described in detail below is used with the device controller 41 and has seven output driver stages which are used in the application of the power-on circuit 182 to the device controller 41. However, the same power-on circuit 182 is also used with the CPU 105 and the bus controller 37, but in those applications the power-on circuit will

have a lesser number of output driver stages.

As illustrated in Fig. 25, the PON circuit 182 comprises a current source 184 and a differential amplifier 186.

The differential amplifier 186 has, as one input, a temperature compensated reference voltage input on a line 188 and has a second input on a line 190 which is an indication of the voltage that is to be sensed by the power-on circuit.

The reference voltage on line 188 is established by a zener diode 192.

The differential amplifier 186 comprises a matched pair of transistors 194 and 196.

The voltage applied on the line 190 is determined by resistors 198, 200 and 202. The resistors 198, 200 and 202 are metal film resistors which provide a high degree of temperature stability in the PON circuit.

The outputs on lines 204 and 206 of the differential amplifier 186 are applied to a three transistor array (the transistors 208, 210 and 212), and this three transistor array in turn controls the main output control transistor 214.

The main output control transistor 214 drives all output drivers that are attached. For example, in the application of the PON circuit 182 for the device controller 41 (as illustrated in Fig. 25), the main output transistor 214 drives output stages 216 through 228. The output stage 216 is used to clear the logic, the output stages 218, 220 and 222 are used in combination with the interface devices of one port 43 of the device controller 41, and the output stages 224, 226 and 228 are used in combination with the interface device of the other port 43 of the device controller 41.

Finally, the PON circuit 182 includes a hysteresis control 230. The hysteresis control 230 includes resistors 232, 234 and a transistor 236.

In operation, assuming that operation is started from a power off state to a power on condition, the power is applied through the current source 182 to the differential amplifier 186 and to the main output control transistor 214. At this time the voltage on the line 190 is less than the voltage on the line 188 so the differential amplifier 186 holds the output of the main output control transistor 214 in the off state. This, in turn, will force the output stages 216 through 228 on.

This asserts the output of the PON circuit 182 in the zero state, the state indicating that power is not within acceptable limits.

As voltage rises, the input voltage on line 190 will increase until it equals the reference voltage on line 188. At this point the differential amplifier 186 drives the main output control transistor 214, turning it on. This removes the base drive from the output stages 216 through 228, forcing these output stages off. The output of the PON circuit 182 is then a one, indicating that the power is within acceptable limits.

At this point the hysteresis control circuit 230 comes into play. While power was coming on, the transistor 236 of the hysteresis control circuit 230 was on. When the transistor 236 is on, the resistance value of the resistor 202 appears to be less than the resistance value of this resistor 202 is when the transistor 236 is off.

The point at which the main output control transistor 214 turns on is the point at which the hysteresis transistor 236 turns off. Turning off the hysteresis transistor 236 causes a slight voltage jump in the line 190 which further latches the differential amplifier 186 into the condition where the differential amplifier 186 sustains the main output transistor 214 in the on state.

The state of the PON circuit will remain stable in this condition with the main output control transistor 214 on and the output drivers 216 through 228 off until the plus five volts drops below a lower threshold point, as determined by the voltage applied on the line 190.

As the voltage on the line 190 decreases below the reference voltage on the line 188, (because the five volts supply is going down in a power failure condition), then the differential amplifier 186 turns off the main output control transistor 214. This, in turn, turns on the output driver stages 216 through 228.

Since the hysteresis transistor 236 was off as power dropped, the voltage applied to the input of the PON circuit 182 must drop somewhat farther than the point at which the PON circuit 182 sensed that power was within the acceptable limits during the power-up phase of operation.

This differential or hysteresis is used to inhibit any noise on the five volt power supply from causing any oscillation in the circuit that would erroneously indicate that power is failing.

The PON circuit 182 shown in Fig. 25 provides very accurate sensing of the two voltages used by the PON circuit to determine its state (whether a one or a zero output of the PON circuit).

In order to sense these two voltages very accurately the PON circuit must have the capability of compensating for initial tolerances of the different components and also the capability to compensate for changes in temperature during operation. In the PON circuit 182, the zener diode 192 is the only critical part that must be compensated for because of its initial tolerance, and this compensation is provided by selecting the resistor 198.

Temperature compensation is achieved because the zener diode 192 is an active zener diode and is not a passive zener diode.

(32)

Effective temperature compensation is also achieved because the two transistors in the differential amplifier 186 are a matched pair of transistors and the resistors 198, 200 and 202 are metal film resistors.

Each port 43 includes a number of lines which are indicated by the general reference numeral 179 in Fig. 20 and Fig. 19. This group of lines 179 includes the individual lines 201 (sixteen (16) of which make up the Input Bus—I Bus), device address lines 203, Output Bus lines 205 (of which there are sixteen), a take ownership line 207 and general lines 209 which transmit such signals as parity, the T bus, and other similar lines which are required because of the particular hardware implementation.

These particular lines 201, 203, 205, 207 and 209 correspond to the lines with the same numbers in Fig. 21, which is the block diagram of the interface common logic. However, there are two sets of each of these lines in Fig. 21 because the interface common logic 181 is associated with each of the dual ports 43 in a device controller 41.

With reference to Fig. 21, the interface common logic 181 includes the ownership latch 185 (see also Fig. 19). This ownership latch determines the logical connection between the interface common logic 181 and a port 43 from which TAKE OWNERSHIP signal has been received over the line 207.

As noted above, the TAKE OWNERSHIP signal is derived by the port hardware from a load address and command (LAC) T bus command (see Fig. 18) with a particular operation code in the command field on the D bus. When the port receives the function LAC on the T bus from the channel, the port logic examines the command field (the top six bits) on the D bus. Then, if the command field contains a code specifying a take ownership command, the port hardware issues a signal to set the ownership latch to connect the port to the interface common logic and thence to the control part of the device controller. If the command field specifies a kill command, the port hardware issues a signal to clear the port's enable latch. This operation happens only if the device address field on the D bus matches the port's device address jumpers, and no parity error is detected during the command. That is, no commands (including the take ownership, kill, etc.) are executed if a parity error is detected on the LAC.

As a consequence, the I/O channel 109 issuing the Take Ownership command gains control of the device controller 41, and the other port 43 is logically disconnected. Take Ownership may also cause a hard clear of the controller's internal state.

The state of the ownership latch 185 determines which port may pass information through the multiplexer 211. One the ownership latch 185 is set in a given direction, it stays in that state until a Take Ownership command is received by the other port. Assertion of the I/O reset line (IORST) will also cause ownership to be given to the other port after the internal state of the device controller has been cleared.

Control signals are chosen by the state of the ownership register 185 and from the appropriate one of the ports 43 and are transmitted by the multiplexer 211 to the control part 187 of a device controller on a set of control lines 215. Data is selected from an appropriate one of the ports 43 on lines 205 and are loaded into the data register 213 and presented to the controller on an Output Bus (O bus) 217.

Some of the control lines 215 (the lines 215A) are used to control the multiplexer 220 in selecting information from the controller as transmitted on lines 219, to be returned by the input bus (I bus) 201 to the ports 43 (Fig. 20) and then to the channel 109 of a processor module 33. A line 221 returns the device address from the appropriate port 43 to the I bus 201 and thence to the I/O channel 109.

The data buffer 189 shown in Fig. 19 is illustrated in more detail in Fig. 22.

In accordance with the present invention many of the device controllers 41 incorporate a multiword buffer for receiving information at a relatively slow rate from a peripheral device and then transmitting that information at or near memory speed to the processor module to maximize channel bandwidth utilization.

In the buffer design itself it is important that the device controllers 41 be able to cooperate with each other in gaining access to the channel 109 to avoid error conditions. In order for the device controllers 41 to cooperate properly, the multiword buffers 189 are constructed to follow certain guidelines.

These guidelines include the following:

First of all, when a device controller makes a reconnect request for the channel 109 it must have enough buffer depth left so that all higher priority device controllers 41 and one lower priority device controller 41 may be serviced and the reconnect latency of the reconnect request can occur without exhausting the remaining depth of the buffer. This is called Buffer Threshold, abbreviated T in Fig. 23.

Secondly, after the buffer has been serviced, it must wait long enough to permit all lower priority device controllers 41 to be serviced before making another reconnect request. This is called Holdoff. The buffer depth (D in Fig. 23) is the sum of the holdoff depth plus the threshold depth.

The holdoff and threshold depths are a function of a number of variables. These

include the device rate, the channel rate, the memory speed, the reconnect time, the number of controllers of higher priority on that I/O bus, the number of controllers of lower priority on that I/O bus, and the maximum burst length permissible.

A controller at high priority on an I/O bus has more controllers of lower priority associated with it on the same I/O bus than another controller at lower priority on the same I/O bus, and therefore the higher priority controller requires more holdoff depth than the lower priority controller. Similarly, a controller at low priority on an I/O bus requires more threshold depth than a controller at higher priority. The buffer 189 in a controller is constructed to take advantage of the fact that as holdoff requirement increases the threshold requirement decreases, and as the threshold requirement increases the holdoff requirement decreases. This is accomplished by making the stress at which a reconnect request is made be variable, the actual setting depending on the characteristics of the controllers at higher and lower priority in a particular I/O channel configuration. The buffer depth is therefore the maximum of the worst-case threshold depth or worst-case holdoff depth requirement, rather than the sum of the worst-case threshold depth and worst-case holdoff depth. This allows the buffer depth to be minimized, and shortens the time required to fill or empty the buffer.

A number of these parameters are graphically illustrated in Fig. 23. In Fig. 23 time has been plotted on the horizontal axis versus words in the buffer on the vertical axis for an output operation.

Starting at point D on the left hand part of Fig. 23 (and assuming a buffer filled to the full buffer depth), data is transferred to a device at a rate indicated by the line of slope $-R_D$ and this data transfer continues without any reconnect signal being generated until the buffer depth decreases to the threshold depth as indicated by the intersection of the line of slope $-R_D$ with the threshold depth line T at point 223.

At this point the reconnect request is made to the channel 109 as indicated by the legend on the horizontal axis in Fig. 23.

The transfer of data continues from the buffer at the rate indicated by the line of slope $-R_D$ and the request is held off by higher priority device controllers 41 until point 225 at which point the request is honored by the channel 109, and the I/O channel begins its reconnect sequence for this device controller.

At point 227 the first data word has been transmitted by the channel 109 to the device controller buffer 189, and the channel 109 then transfers data words at a rate indicated by the line of slope R_C into the buffer 189.

At the same time the device controller 41 continues to transfer data words out of the buffer at the rate $-R_D$ so that the overall rate of input to the buffer 189 is indicated by the line of slope $R_C—R_D$ until the buffer is again filled at the point 229. At 229 the buffer is full, and the device controller disconnects from the channel 109, and the data transfer continues at the rate indicated by the slope line $-R_C$.

The notation t, in Fig. 23 indicates the time required for the polling and selection of this device controller and the transfer of the first word. This will be discussed again below in relation to Fig. 16.

The letter B in Fig. 23 indicates the burst time. The burst time is a dynamic parameter. The length of any particular burst is dependent upon the device transfer rate, the channel transfer rate, the number of devices with transfers in progress and the channel reconnect time. The maximum time permitted for a burst is chosen to minimize the amount of buffer depth required while accomodating high device transfer rates and also the number of devices that can transfer concurrently.

Fig. 22 is a block diagram of a particular embodiment of a buffer 189 constructed in accordance with the present invention to accomplish the holdoff and threshold requirements illustrated in Fig. 23.

At the same time the device controller 41 continues to transfer data words out of the buffer at the rate $-R_D$ so that the overall rate of input to the buffer 189 is indicated by the line of slope $R_C—R_D$ until the buffer is again filled at the point 229. At 229 the buffer is full, and the device controller disconnects from the channel 109, and the data transfer continues at the rate indicated by the slope line $-R_C$.

The notation t, in Fig. 23 indicates the time required for the polling and selection of this device controller and the transfer of the first word. This will be discussed again below in relation to Fig. 16.

The letter B in Fig. 23 indicates the burst time. The burst time is a dynamic parameter. The length of any particular burst is dependent upon the device transfer rate, the channel transfer rate, the number of devices with transfer in progress and the channel reconnect time. The maximum time permitted for a burst is chosen to minimize the amount of buffer depth required while accomodating high device transfer rates and also the number of devices that can transfer concurrently.

Fig. 22 is a block diagram of a particular embodiment of a buffer 189 constructed in accordance with the present invention to accomplish the holdoff and threshold requirements illustrated in Fig. 23.

The buffer 189 shown in Fig. 22 comprises

(34)

an input buffer 231, a buffer memory 233, an output buffer 235, an input pointer 237, an output pointer 239, a multiplexer 241, buffer control logic 243 (described in more detail in Fig. 26), a multiplexer 245 connected to the buffer control logic 243 and a stress counter 247.

As also illustrated in Fig. 22, two groups of data input lines (lines 217 and 249) are fed into the input buffer 231.

One group of data input lines include sixteen device data input lines 249.

The other group of input lines include sixteen Output Bus lines (O bus lines) 217.

One or the other of these two groups of input signals is then fed from the input buffer 231 to the buffer memory 233 by a group of lines 251. There are sixteen of the lines 251.

Data is taken from the buffer memory 233 and put into the output buffer 235 by a group of lines 253. There are sixteen of the lines 253.

The output buffer 335 transmits the data back to the interface common logic 181 (see Fig. 19 and Fig. 21) on a group of sixteen lines 219 and to the devices 45, 47 (such as 49, 51, 53 shown in Fig. 1) on a group of sixteen lines 255 as indicated by the legends in Fig. 22.

The input and output pointers 237 and 239 function with the multiplexer 241 as follows.

When data is being transferred from the input buffer 231 to the buffer memory 233, the input pointer 237 is connected to the buffer memory 233 through the multiplexer 241 to determine the location into which the word is written.

When data is being transferred out of the buffer memory 233 into the output buffer 235, the output pointer 239 is connected to the buffer memory 233 through the multiplexer 241 to determine the location from which the word is taken.

The purpose of the buffer control logic 243 illustrated in Fig. 22 and Fig. 26 is to keep track of the stress placed on the buffer 189. In this regard, the degree of the full or empty condition of the buffer in combination with the direction of the transfer with respect to the processor module (whether input or output) determines the degree of stress. Stress increases as the device accesses the buffer and decreases as the channel accesses the buffer.

In the implemention shown in Figs. 22 and 26 the stress counter measures increasing stress from 0—15 on an input, and decreasing stress from 0—15 on an output. Another implementation (not shown in the drawings) would add the direction of transfer in the buffer control logic such that two new lines would access the pointers 237 and 239 and the stress counter would always measure increasing stress.

With continued reference to Fig. 22, a channel request line 215 (see also Fig. 21) and a device request line 257 (coming from the control part 187 of the device controller) are asserted to indicate access to the buffer 189.

The multiplexer 245 chooses one of these lines as a request to increase the buffer fullness and chooses the other line as a request to decrease the buffer fullness based on the direction of the transfer (whether input or output) with respect to the processor module.

The line chosen to increase buffer fullness is also used to load data from the appropriate data lines 249 or 217 (see Fig. 22) into the input buffer 231 by means of the line 259.

The channel and the device may access the buffer 189 at the same time, and the buffer control logic 243 services one request at a time. The buffer control logic 243 chooses one of the lines for service and holds the other line off until the buffer control logic 243 has serviced the first request, then it services the other request.

The servicing of a request by the buffer control logic 243 includes the following.

First of all, it determines the direction of transfer (into or out of) the buffer memory 233, and it asserts line 261 (connected to the multiplexer 241) as appropriate to select the input pointer 237 or the output pointer 239 through the multiplexer 241.

Secondly, on an output request, the buffer control logic 243 asserts line 263 which does three things.

(A) It writes the word from the input buffer 231 into the buffer memory 233 at the location determined by the input pointer 237 and the multiplexer 241.

(B) It increments the stress counter 247.

(C) The buffer control logic 243 increments the input pointer 237.

Thirdly, on an output transfer, the buffer control logic 243 asserts line 265 which accomplishes the following three operations.

(A) The buffer control logic 243 writes the word being read from the buffer memory 233 as determined by the output pointer 239 and multiplexer 241 into the output buffer 235.

(B) The buffer control logic 243 decrements the stress counter 247.

(C) The buffer control logic 243 increments the output pointer 239.

The stress counter 247 determines when the buffer 189 is full (D), or at threshold depth (T) as shown by the output line legends in Fig. 22.

The output of the stress counter is decoded, and any one of the decoded values may be used to specify that the buffer is at threshold depth. In the preferred embodiment, wire jumpers are used to select one of sixteen possible stress values, and a reconnect request is made to the channel 109 when the stress on the buffer 189 reaches that value.

The control part 187 of the device controller uses these three signals (which correspond to the legends in Fig. 23) to make reconnect requests and disconnect requests on respective lines 145 (see Fig. 14 and Fig. 12) and 159 (see Fig. 14 and Fig. 12).

The STI (stop in) signal transmitted on line 159 shown in Fig. 14 and Fig. 12 is related to the buffer depth (D), the full or empty conditions of the buffer and the direction of transfer; and the RCI (reconnect in) signal on line 145 of Fig. 14 and Fig. 12 is related to the threshold depth (T) indication from the stress counter 247 in Fig. 22. Thus, the STI signal is asserted when the buffer 189 reaches a condition of minimum stress (full on output and empty on input). The STI signal signals the channel 109 that the controller 41 wishes to terminate the burst data transfer. When the buffer passes through its threshold, it asserts the RCI signal on line 145 to indicate to the channel 109 that the buffer wishes to transfer a burst of data.

Fig. 26 shows details of the multiplexer 245, the buffer control logic 243 and the stress counter 247 of the buffer 189 shown in Fig. 22.

In Fig. 26 the multiplexer 245 is shown as two sets of gates 245A and 245B, request flip-flops 267A and 267B, a clock flip-flop 269, request synchronization flip-flops 271A and 271 B, a priority resolving gate 273 and request execution gates 275A and 275B.

The stress counter 247 comprises a counter section 247A and a decoder section 247B as indicated by the legends in Fig. 26.

As illustrated in Fig. 26, the two sets of gates 245A and 245B have used the channel request signal (line 215) and the device request signal (line 257) and the read and write signals to determine which of the channel or the device is putting data onto the buffer 189 and which is taking data out of the buffer 189.

The request flip-flops 267A and 267B store the requests until the control logic has serviced the request.

The clock flip-flop 269 generates a two phase clock used by the request synchronization flip-flops 271A and 271B and the request execution gates 275A and 275B.

The request synchronization flip-flops 271A and 271B synchronize the request to the clock generation flip-flop 269 and stabilize the request for execution.

The priority resolving gate 273 picks one of the requests for execution and causes the other request to be held off.

The request execution gates 275A and 275B execute the requests in dependence on the synchronized request.

Each output signal on the lines 263 and 265 performs the functions described above (incrementing and decrementing the stress counter, updating the buffer memory or output buffer, and updating the input pointer or output pointer).

In addition, each signal clears the appropriate request flip-flop through the lines 277A and 277B illustrated in Fig. 26.

As noted above, Figs. 15, 16 and 17 show the three sequences of operation of the I/O system.

In the operation of the I/O system, the normal data transfer between a processor module 33 and a particular device, such as a disc 45, includes an EIO sequence to initiate the transfer.

The EIO instruction selects the particular device controller and device and specifies the operation to be performed.

The device controller 41 initiates the I/O between the device controller 41 and the particular device.

The device controller 41 periodically reconnects to the channel 109 and transfers data between the device controller 41 and the channel 109. The periodic reconnection may be for the purpose of either transferring data from the channel to the device or for the purpose of transferring data from the device to the channel.

When the transfer of data is complete the device controller 41 interrupts the CPU 105, which responds by issuing an IIO or an HIIO sequence.

The IIO sequence determines the identity of the interrupting device and conditions under which the transfer completed.

The HIIO sequence is similar to the IIO sequence but is issued in response to a high priority I/O interrupt.

The "Execute I/O" CPU instruction (EIO) is defined by the T bus state changes shown in Fig. 15.

The first state shown in Fig. 15 (the state farthest to the left) is the no-operation (NOP) or idle state. The other states are the same as those listed in Fig. 18 by the corresponding mnemonics—load address and command (LAC), load parameter (LPRM), read device status (RDST), deselect (DSEL) and abort instruction (ABTI).

As in the state changes shown in Figs. 6, 7 and 8, the solid line arrows indicate a state change, and a dashed line arrow indicates a condition which must occur before a state change can occur.

The EIO instruction and execution shown in Fig. 15 is directly under control of the microprocessor 113 (see Fig. 12) of the CPU 105.

This CPU initiation is shown as transmitted to the state machine in Fig. 15 by the line 117; the initiation signal is accepted only when the T bus is in the idle state.

Once the CPU initiation is applied, the T bus goes from the NOP (idle) state to the LAC state.

(36)

In the LAC state or function a word is taken from the top of the register stack 112 in the CPU 105 (see Fig. 12) and is put on the D bus 161 (see Fig. 14).

As described above, this word is used to select a particular device controller 41 and a particular peripheral device 45, 47, 49, 51 or 53 (see Fig. 1), and the word is also used to specify the operation to be performed.

In the next bus cycle the T bus goes to the LPRM state.

In the load parameter state (LPRM) the word just below the top of the register stack in the CPU 105 (see Fig. 12) is put on the T bus 161 (see Fig. 14) by the I/O channel 109 and is passed to the device controller 41 selected during the previous LAC state.

At the conclusion of the handshake cycle, as shown by the dashed line arrow in Fig. 15, the T bus goes to the RDST state. In this state the device controller 41 returns the device status (the status of a particular device selected and comprising the set of signals describing the state of that device) from the device controller 41 and places it on the top of the register stack 112 in the CPU 105.

During the load parameter and read device status state several errors may have occurred. These include parity error, handshake time out, and an error indication in the status word. If an error did occur, then the T bus machine 143 (Fig. 13) goes from the RDST state to the abort instruction (ABTI) state.

The ABTI state instructs the device controller 41 to ignore the previous LAC and LPRM information passed to it by the I/O channel 109 and then the T bus (channel) returns to the NOP (idle) state.

If, after the RDST state no error was detected, (as shown by the dashed line arrow 114 in the top branch of Fig. 15), the T bus goes to the deselect state (DSEL).

With the T bus in the deselect state, the device controller 41 clears its select latch 173 and responds to the instruction issued to it (passed to it during the LAC state) and the T bus returns to the NOP (idle) state.

In the operation of the I/O system there are a number of device request signals that can happen asynchronously. For example, a reconnect signal may be generated after an EIO sequence to request that the channel transfer data to the controller. Or the device controller 41 may assert an interrupt request line under a number of different conditions, e.g. to signal the completion of an EIO sequence or to report an unusual condition in a peripheral device.

The device request lines are common to all device controller ports 43 attached to a particular I/O bus 39.

The channel 109 responds to reconnect requests made on the line RCI (145 of Fig. 14), and the CPU 105 responds to requests made on the LIRQ line 147 (see also Fig. 14) with an IIO sequence, and to a request made on the HIRQ line 149 with an HIIO sequence.

The first thing that the channel 109 or CPU 105 does in response to a Device Request signal is to determine the identity of the highest priority device controller 41 asserting a request. That is, there may be several device controllers 41 asserting a request to the channel 109 at one time, and the channel will select a particular device controller in accordance with a predetermined priority scheme.

In a particular embodiment of the present invention up to thirty-two device controllers 41 can be connected to a single channel 109.

The thirty-two device controllers are connected in a star poll using the sixteen bit data bus 161. One additional line 151 is used to divide the thirty-two device controllers into two groups of sixteen each. One group of sixteen device controllers is assigned priority over the other group; and priority is also assigned among the sixteen within each group. The device responding on bit zero of the D bus during a polling sequence has the highest priority within a rank, and the one responding on bit 15 has the lowest priority.

In initial introduction, it may be noted that polling (which will now be described) involves the state descriptions shown in Fig. 16 and 17 up to and including that handshake which occurs during the select (SEL) state in each figure.

With continued general reference to Figs. 16 and 17, the channel 109 sets the rank line to zero and then presents the T bus function RPOL (Fig. 16) if the response is to a reconnect request, while the CPU 105 presents an LPOL (Fig. 17) T bus function if the CPU is responding with an IIO sequence, or an HPOL T bus function if the CPU is responding with an HIIO sequence. This is the only major point of difference between the showings in Fig. 16 (the channel response) and Fig. 17 (the CPU response) with regard to polling.

Referring specifically to Fig. 16 and the response of the channel 109 to assertion of the RCI line 145 (see Fig. 14), all devices with a reconnect request pending that would respond on rank zero place a one bit response on the D bus. That is, all these devices assert a line of the D bus 161 corresponding to their priority within the rank.

The channel 109 transfers the D bus response into the priority resolve register 135 (see Fig. 13). This priority resolve register 135 output determines which device controller has the highest priority (in accordance with the scheme described above) and asserts the appropriate bit back onto the D bus 161, if there is a bit asserted in rank zero by the attached device controllers.

If there are one or more devices asserting a response to the priority resolve register on rank zero, the output of the priority resolve register is presented to all device controllers attached, along with the select function (SEL) on the T bus, and the device controller whose priority on rank zero matches the output of the priority resolve register sets it select bit 173 (see Fig. 19), and then that port will respond to subsequent states in the sequence. This is the mode of operation indicated by the solid line arrow going from the state indicated by RPOL with a rank equals zero to select (SEL).

If the priority resolving register 135 determines that no device responded when the rank line equalled zero, then the channel 109 sets the rank line to one and reissues the RPOL T bus command. Then, if the priority resolving register determines that a response occurred on rank 1, the channel asserts the T bus select function as before.

However, if the priority resolving register 135 determines that no response was made on rank 1, the channel returns to the idle state indicated by state NOP in Fig. 16.

This latter event is an example of a failure which might occur in one port 43 and which would result in the system 31 accessing that particular device controller 41 through the other port 43.

As noted above, the action of the priority resolving register 135 in response to an IIO or an HIIO sequence initiated by the CPU 105 is the same as the response of the priority resolving register 135 to a reconnect sequence initiated by the channel in response to a reconnect in on the line 145 from a device controller 41.

With continued reference to Fig. 16, the reconnect sequence begins with the poll sequence described above for reconnecting the highest priority device controller 41 making a request.

The next step in the reconnect sequence is to determine the actual device controller number contained in the device address comparator 193. As noted above, the device address comparator 193 includes jumpers to determine a physical device controller number. These are the same jumpers that are used on a LAC T bus function during an EIO sequence to determine a particular port. In the reconnect sequence the address determined by these jumpers is returned to the I/O channel via the D bus during the T bus RAC state to access a table defining the buffer area for this device.

It is also necessary to determine the direction of the transfer (whether an input or output transfer to the processor module). To accomplish this determination of the direction of the requested transfer and the device address, the channel asserts the RAC T bus function and the device controller 41 returns the device controller address and the transfer direction.

The channel uses the device address returned by the device controller 41 to access a two word entry (142) in an I/O control table (IOC) 140 (Fig. 12) which defines a buffer area 138 in the memory 107 for this particular device controller and device.

The format of a two word entry 142 is shown enlarged in Fig. 12 to show details of the fields of the two words.

There is a two word entry 142 in the IOC table 140 for each of the eight possible devices of each of the thirty-two possible device controllers 41 attached to an I/O bus 39 associated with a particular processor module 33, and each processor module 33 has its own IOC table.

Each two word entry describes the buffer location in main memory and remaining length to be transferred at any particular time for a particular data transfer to a particular device. Thus, as indicated by the legends in Fig. 12, the upper word specifies the transfer address to or from which the transfer will be made by a burst; and the lower word specifies the byte count specifying the remaining length of the buffer area and the status of the transfer.

The fields representing the status of the transfer include a protect bit P and a channel error field CH ERR. The channel error field comprises three bits which can be set to indicate any one of up to seven numbered errors.

The transfer address and byte count are updated in the IOC table 140 at the conclusion of each reconnect and data transfer sequence (burst). The transfer address is counted up and the byte count is counted down at the conclusion of each burst. The amount reflects the number of bytes transferred during the burst.

The second word also contains (1) a field in which any error encountered during a reconnect and data transfer sequence may be posted for later analysis, and (2) a protect bit to specify that the buffer area in memory 107 may be read from but not written into.

The protect bit serves to protect the processor memory 107 from a failure in the device controller 41. That is, when the device controller 41 returned the transfer direction to the channel 109 during a read address and command (RAC) T bus function, a failure in the device controller 41 could cause the device controller to erroneously specify an input transfer. Then the channel would go to the IN state and transfer data from the device controller into memory, thus causing data in the buffer 138 to be lost. The protect bit allows the program to specify that the channel may not write into this buffer area; that is, the device may only specify an output transfer.

The transfer address specifies the logical pathe 139B (see Fig. 12).

The channel places the transfer address in the channel memory address register 129 (see Fig. 13) and places the byte count in the character count register 131 (see Fig. 13).

Depending upon the direction of the transfer, (which the channel retrieved from the device during the RAC state shown in Fig. 16), the channel puts the T bus in either the IN state or OUT state and transfers data between the device controller 41 and memory 107 using the channel memory address register 129 to specify the logical path 139C (see Fig. 12). The channel memory address register 129 and character count register 131 are updated with each word transferred during the burst to reflect the next address in the buffer and the number of characters yet to be transferred. At the conclusion of a burst the contents of the channel memory address register 129 and of the character count register 131 are written into the IOC table 140.

In operation, for each word transferred in from the device on an in transfer, the channel 109 accepts the word by the handshake mechanism described above and places the word in the I/O data register 127 (see Fig. 13) and then transfers the word to the buffer area in memory defined by the logical path 139C (see Fig. 12).

On an out transfer the channel 109 takes a word from the buffer area over logical path 139C and transfers the word to the channel memory data register 125. The channel then transfers the word into the I/O data register 127 (Fig. 13) and handshakes with the device controller which accepts the word into its interface data register 213.

The high speed of the I/O channel is accomplished by pipelining where the word in the I/O data register 127 is handshaken to the device while the channel concurrently requests and accepts the next word in the transfer from memory 107 and places it in the channel memory data register 125. Since it takes just as long to put a word out to the device as it does to accept a word from memory for the device, the two operations can be overlapped.

During the burst, the channel decremented the character count register by two for every word tansferred, since there are two bytes in every word.

The burst transfer can terminate in two ways. The burst transfer can terminate normally or the burst transfer can terminate with an error condition.

In the normal case there are two possibilities.

In a first condition of operation, the character count register 131 can reach a count of either one or two bytes remaining to be transferred. In this situation the channel puts up EOT (line 165 as shown in Fig. 14) signifying that the end of transfer has been reached. If the count reaches one, then the channel asserts EOT and PAD OUT (line 167 of Fig. 14) signifying the end of transfer with an odd byte.

If the character count reaches two, the channel puts up EOT, but PAD OUT (PADO on line 167 of Fig. 14) is not required because both bytes on the bus are valid.

In either case, the device controller 41 responds by asserting STOP IN (STI) on line 159 (see Fig. 14), and the device controller 41 also asserts PAD IN (PADI) on line 169 (Fig. 14) if the channel asserted PAD OUT (PADO).

In this first case of normal termination, the transfer as a whole, not just the burst, is terminated by the channel 109.

The other normal completion is when the device controller 41 ends the burst by asserting STOP IN (STI) in response to the channel SERVICE OUT (SVO). This signifies that the buffer 189 (see Fig. 19) has reached a condition of minimum stress (as indicated by point 229 in Fig. 23).

The STOP IN (STI) can occur on an output transfer or on an input transfer.

On an input transfer, if the device controller 41 wishes to terminate the transfer as well as the burst, the device controller 41 can assert STOP IN (STI); and, to signify an odd byte on the last word, the device controller 41 can also assert PAD IN (PADI).

As shown in Fig. 16, when the transfer is terminated by a non-error condition (STI OR EOT) on either an output transfer or an input transfer (as shown by the balloons OUT and IN in Fig. 16), the channel 109 updates the IOC table entries as noted above, and returns to the idle (NOP) state shown in Fig. 16.

As noted above, the transfer can also be terminated by an error condition.

During the burst several errors may occur as follows.

First, the device controller 41 may request an input transfer into a buffer whose protect bit P is set in the IOC table as mentioned above.

Second, the device controller 41 may not return a PAD IN (PADI) signal in response to a PAD OUT (PADO) signal from the channel 109.

Third, the channel 109 may detect a parity error on the D bus 161.

Fourth, the device controller 41 may not respond to a SERVICE OUT (SVO) signal from the channel 109 within the allotted time as mentioned above in the discussion on handshakes.

Fifth, the buffer area specified by the IOC table entries may cross into a page whose map marks it absent (see the discussion of the mapping scheme in the memory system).

Sixth, a parity error may be detected in accessing the map while accessing the memory during the reconnect in and data transfer sequence. See the description in the memory system relating to the parity error check.

Seventh, the memory system may detect an un-correctable parity error when the channel 109 accesses the memory. See the description of the memory system for this parity error check.

If any of these error conditions occur, the channel 109 goes to the abort data transfer state (ABTD) as shown in Fig. 16. This instructs the device controller 41 that an error has occurred and that the data transfer should be aborted. The channel 109 then goes back to the idle state which is (NOP) as shown in Fig. 16.

When an error occurs, the channel 109 updates the IOC table entries and puts an error number indicating one of the seven errors noted above in the error field of the second word of the IOC table entry as mentioned above.

Thus, if a single error occurs, the number of that error is entered in the error field of the IOC table entry.

If more than one error occurs, the channel 109 selects the error from which recovery is least likely to occur and enters only the number of that error in the error field of the IOC table entry.

There is one other type of error that can occur. The device controller 41 may try to reconnect to the channel when the count word in the IOC table is zero. In this event, the channel will not let the device controller reconnect and the channel goes through the sequence as described above with reference to Fig. 16, but when the channel determines that the count word in the IOC table is zero, the channel 109 goes directly to the abort (ABTD) state. This is an important feature of the present invention because it protects the processor memory from being overwritten by a failing device.

If the count is zero in the byte count count of the second word of the IOC table entry 142 for a particular device, and if the device controller 41 attempts to reconnect to the channel 109, the channel issues an abort (ABTD) to the device controller 41 as noted above and leaves the channel error field of the two word entry 142 at zero.

In response to an abort data (ABTD) T bus function, the device controller 41 makes an interrupt request on the line HIRQ or LIRQ (lines 149 or 147 as shown in Fig. 14) to the channel 109.

The device controllers 41 may at any time request an interrupt on these two lines.

An interrupt generally indicates that a data transfer has been completed or terminated by an abort from the channel (an ABTD from the channel) or by an error condition within the device controller 41 or attached device, or that a special condition has occurred within the device controller or an attached device. For example, when the power is applied and the PON circuit indicates that power is at an acceptable level, the device controller interrupts the processor module to indicate that its internal state is Reset because power was off or had failed and has been reset by the PON circuit.

In response to an interrupt, the program running within the processor module 33 issues an interrogate I/O instruction (IIO) or an interrogate high priority I/O instruction (HIIO) over the I/O bus 39.

The IIO instruction is issued in response to a low priority I/O interrupt, that is, one issued on the low priority interrupt request (LIRQ) line 147 (see Fig. 14).

The HIIO instruction is issued in response to a high priority I/O interrupt, that is, one requested on a high priority interrupt request (HIRQ) line 149 (see Fig. 14).

The microprocessor 113 (see Fig. 12) executes the EIO, IIO or HIIO instruction by taking control of the channel control logic 141 and data path logic 123.

The sequence for these instructions is illustrated in Fig. 17; and, as noted above, the sequence starts with a polling sequence.

The IIO instruction polls in a sequence using the T bus function low priority interrupt poll (LPOL) while the HIIO instruction polls in a sequence using the T bus function high priority interrupt poll (HPOL).

The polling sequence which is also described above completes by selecting the appropriate device controller 41 by using the T bus function select (SEL) as shown in Fig. 17.

The appropriate device controller 41 selected is that device controller which has the highest priority and is making an interrupt request.

The sequence continues with a read interrupt cause (RIC) T bus function as shown in Fig. 17. The device controller 41 responds by returning device dependent status on the D bus 161 (see Fig. 14).

The microprocessor 113 (Fig. 12) reads the status from the D bus 161 and places the status on the top of the register stack 112 (Fig. 12).

The sequence then continues with a read interrupt status (RIST) T bus function as shown in Fig. 17. The device controller 41 responds to this RIST T bus function by returning the device controller number, the unit number and four dedicated status bits on the D bus.

Of the four bit status field, two of the bits indicate respectively, abort (ABTD) and parity error (which parity error may have occurred during a reconnect and data transfer sequence).

The microprocessor 113 copies the content of the D bus—the controller number, the device number and the interrupt status—and places that content on the top of the register stack 112.

If no error occurred during the sequence, then the sequence continues with the deselect (DSEL) state which deselects the device controller 41; and then the sequence goes into the idle (NOP) state as indicated by the line at the top of Fig. 17.

If an error did occur (and the error can be a parity error detected by the channel or a handshake time out), the channel goes from the RIST state to the abort instruction (ABTI) state as shown in Fig. 17. This deselects the device controller 41, and then the channel 109 goes back into the idle (NOP) state as shown by the bottom line in Fig. 17.

As noted above, an I/O operation between a processor module and an I/O device typically consists of a group of sequences, e.g. an EIO followed by some number of reconnect and data transfer sequences, terminating with an IIO sequence. Sequences from several different I/O operations may be interleaved, resulting in apparent simultaneous I/O operation by several devices. Thus, a large number of devices may be accessed concurrently; the exact number depends on the channel bandwidth and the actual bandwidth used by each device.

The I/O system and dual port device controller architecture and operation described above provide a number of important benefits.

These benefits include (a) flexibility to interface a wide variety of devices, (b) a maximum usage of resources, (c) a fail soft environment in which to access peripheral devices in a multiprocessor system, (d) on line maintenance and upgrade of the multiprocessor system capability, and (e) maximum system through put (as opposed to emphasizing processor through put or I/O through put exclusively) in an on line transaction system in which a large number of concurrent transactions must be processed by the I/O system and CPU.

Flexibility to interface a wide variety of devices is achieved because the system of the present invention does not presuppose any inherent characteristics of a device type. Instead, the present invention provides a structure and operation which can accommodate a wide variety of device operations.

The present invention provides for a maximum usage of resources, primarily by making a maximum usage of memory bandwidth. Each device uses a minimum of the memory bandwidth. This allows a relatively large number of devices to be associated with the particular I/O bus. Because of the inherent speed of the I/O bus, and the buffering technique of the present invention, each particular transfer is made at a relatively high speed limited only by memory speed. Because the transfers are in a burst mode, the overhead associated with each transfer is minimized. This maximizes the use of the channel bandwidth and also permits the use of high speed devices.

The present invention provides for failsoft access to peripheral devices. There are redundant paths to each peripheral device, and containment of failure on any particular path. Failure of a particular module in one path does not affect the operation of a module in another path to that device.

There are comprehensive error checks for checking data integrity over a path, sequence failures and timing failures.

Protection features prevent a peripheral device from contaminating its own buffer or the memory of the system. These protection features include a separate count word in each IOC table and a protect bit in the IOC table. The IOC table is accessible by the channel, but not by the device. This is a second level of protection to prevent the device from accessing any memory not assigned to that device.

The present invention requires only a small number of lines in the I/O bus to provide a flexible and powerful I/O system.

The operation of the device controller is well defined as power is turned on or off to protect the I/O bus from erroneous signals during this time and also to permit on line maintenance and system upgrade.

The present invention uses stress to allow the buffers to cooperate without communicating with each other.

An on line transaction system is obtained through overlapped transfers and processing.

Multichannel direct memory access provides interleaved bursts to give overlapped transfers and minimum waits for accesses to a device. Each burst requires a minimum memory overhead and allows the processor to make maximum use of the memory. This combination allows maximum use of the I/O bandwidth and minimal tie up of the processor.

Attention is directed to our co-pending Patent Application No 25490/77 (Serial No 1588803). from which the present Application has been divided and to Applications Nos 3058/80, 3072/80 (Serial No. 1588806) and 3073/80 (Serial No 1588807).

WHAT WE CLAIM IS:—

1. A processor module for a multiprocessor system of the kind in which separate processor modules are interconnected for parallel processing, said processor module comprising; a central processing unit having a processor clock; a memory; and interprocessor control means for connecting the

The microprocessor 113 copies the content of the D bus—the controller number, the device number and the interrupt status—and places that content on the top of the register stack 112.

If no error occurred during the sequence, then the sequence continues with the deselect (DSEL) state which deselects the device controller 41; and then the sequence goes into the idle (NOP) state as indicated by the line at the top of Fig. 17.

If an error did occur (and the error can be a parity error detected by the channel or a handshake time out), the channel goes from the RIST state to the abort instruction (ABTI) state as shown in Fig. 17. This deselects the device controller 41, and then the channel 109 goes back into the idle (NOP) state as shown by the bottom line in Fig. 17.

As noted above, an I/O operation between a processor module and an I/O device typically consists of a group of sequences, e.g. an EIO followed by some number of reconnect and data transfer sequences, terminating with an IIO sequence. Sequences from several different I/O operations may be interleaved, resulting in apparent simultaneous I/O operation by several devices. Thus, a large number of devices may be accessed concurrently; the exact number depends on the channel bandwidth and the actual bandwidth used by each device.

The I/O system and dual port device controller architecture and operation described above provide a number of important benefits.

These benefits include (a) flexibility to interface a wide variety of devices, (b) a maximum usage of resources, (c) a fail soft environment in which to access peripheral devices in a multiprocessor system, (d) on line maintenance and upgrade of the multiprocessor system capability, and (e) maximum system through put (as opposed to emphasizing processor through put or I/O through put exclusively) in an on line transaction system in which a large number of concurrent transactions must be processed by the I/O system and CPU.

Flexibility to interface a wide variety of devices is achieved because the system of the present invention does not presuppose any inherent characteristics of a device type. Instead, the present invention provides a structure and operation which can accommodate a wide variety of device operations.

The present invention provides for a maximum usage of resources, primarily by making a maximum usage of memory bandwidth. Each device uses a minimum of the memory bandwidth. This allows a relatively large number of devices to be associated with the particular I/O bus. Because of the inherent speed of the I/O bus, and the buffering technique of the present invention, each particular transfer is made at a relatively high speed limited only by memory speed. Because the transfers are in a burst mode, the overhead associated with each transfer is minimized. This maximizes the use of the channel bandwidth and also permits the use of high speed devices.

The present invention provides for failsoft access to peripheral devices. There are redundant paths to each peripheral device, and containment of failure on any particular path. Failure of a particular module in one path does not affect the operation of a module in another path to that device.

There are comprehensive error checks for checking data integrity over a path, sequence failures and timing failures.

Protection features prevent a peripheral device from contaminating its own buffer or the memory of the system. These protection features include a separate count word in each IOC table and a protect bit in the IOC table. The IOC table is accessible by the channel, but not by the device. This is a second level of protection to prevent the device from accessing any memory not assigned to that device.

The present invention requires only a small number of lines in the I/O bus to provide a flexible and powerful I/O system.

The operation of the device controller is well defined as power is turned on or off to protect the I/O bus from erroneous signals during this time and also to permit on line maintenance and system upgrade.

The present invention uses stress to allow the buffers to cooperate without communicating with each other.

An on line transaction system is obtained through overlapped transfers and processing.

Multichannel direct memory access provides interleaved bursts to give overlapped transfers and minimum waits for accesses to a device. Each burst requires a minimum memory overhead and allows the processor to make maximum use of the memory. This combination allows maximum use of the I/O bandwidth and minimal tie up of the processor.

Attention is directed to our co-pending Patent Application No 25490/77 (Serial No 1588803), from which the present Application has been divided and to Applications Nos 3058/80, 3072/80 (Serial No. 1588806) and 3073/80 (Serial No 1588807).

WHAT WE CLAIM IS:—

1. A processor module for a multiprocessor system of the kind in which separate processor modules are interconnected for parallel processing, said processor module comprising; a central processing unit having a processor clock; a memory; and interprocessor control means for connecting the

processor module to any other processor module in the multiprocessor system across an interprocessor bus operated at a bus clock that is not synchronous with any of the processor clocks, said interprocessor control means including queue means for buffering data between the interprocessor bus and the processor memory, said queue means having a first logic state machine operated in synchronism with the bus clock and a second logic state machine operated in synchronism with the processor clock, and interlock means operatively associated with the first and second logic state machines for qualifying certain transitions of one of the state machines in dependence upon the state of the other state machine so that the interprocessor bus can be operated at the bus clock rate and the central processing unit and memory of the processor module can be operated at the processor clock rate without loss of the data being transferred or loss of information about the state of the transfer sequence.

2. A processor module according to claim 1 substantially as hereinbefore described with reference to, and as shown in, the accompanying drawings.

FORRESTER, KETLEY & CO.,
Chartered Patent Agents,
Forrester House,
52 Bounds Green Road,
London N11 2EY,
also at
Rutland House,
148 Edmund St.,
Birmingham B3 2LD.
Scottish Provident Building,
29 St. Vincent Place,
Glasgow G1 2DT.
Agents for the Applicants.

Printed for Her Majesty's Stationery Office by Burgess & Son (Abingdon) Ltd.—1981. Published at The Patent Office, 25 Southampton Buildings, London, WC2A 1AY, from which copies may be obtained.

* C L M S *

1

(9)

SPECIFICATION NO 1588804

ı direction given under Section 17 (1) of the Patents Act 1949 this application proceeded in the name of
NDEM COMPUTERS INCORPORATED, a Corporation organised and existing under the laws of the State
elaware, of 19333 Vallco Parkway, Cupertino, California 95015, United States of America.

E PATENT OFFICE

Bas 83005/4