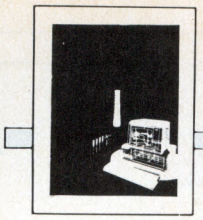


BUS STRUCTURE EASES MULTIPROCESSOR INTEGRATION

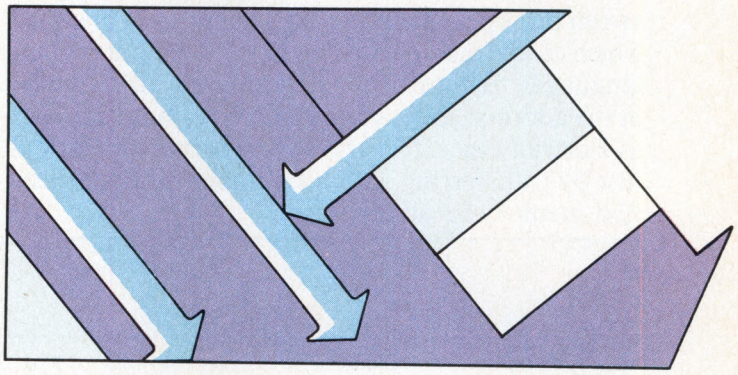
Use of a 32-bit NuBus in memory-mapped I/O and interrupt operations aids system configuration.

by George P. White

Where system integrators once needed to carefully account for processor cycles, the availability of low cost, high performance microprocessors is fostering a new approach to system design. Now, auxiliary processors handle cycle-hungry support functions such as graphics or numerical processing, thus freeing the CPU to direct its power directly to the user's application. Yet, because conventional architectures wrap system resources such as memory and I/O hardware tightly around a CPU core, traditional architectures still do not provide a sufficiently flexible framework for exploiting the cost and performance advantages of multiprocessor designs.

Originally developed at the Massachusetts Institute of Technology specifically for multiprocessor architectures, Texas Instruments' 32-bit NuBus provides system integrators with system architecture independence, high bandwidth, easy system configuration, a simple protocol, and small pin count. In fact, because the NuBus maintains a simple protocol for all bus operations, 49 signal lines are sufficient to handle all transactions for up to 16 different devices in the 4-Gbyte address space of the buses.

George P. White is manager of Nu Machine development at Texas Instruments, 17881 Cartwright Rd, Irvine, CA 92714. He holds a BS in electrical engineering from the Massachusetts Institute of Technology.



The NuBus accomplishes this feat by supporting only read/write transactions. Unlike conventional bus structures, which require separate signals for memory access, interrupts, and I/O operations, the NuBus includes all these operations under its umbrella of read/write transactions. I/O and interrupt operations are mapped into the address space and handled just as accesses to system memory.

Furthermore, by laying system resources within this 4-Gbyte memory address space, the NuBus decouples the logical function of system resources from the physical implementation of system hardware. Instead of altering hardware switches and jumpers, engineers can reconfigure systems simply by changing variables in the NuBus address space. Consequently, without disturbing the logical architecture, system integrators are free to modify a system's physical framework to achieve the required balance between system cost and performance.

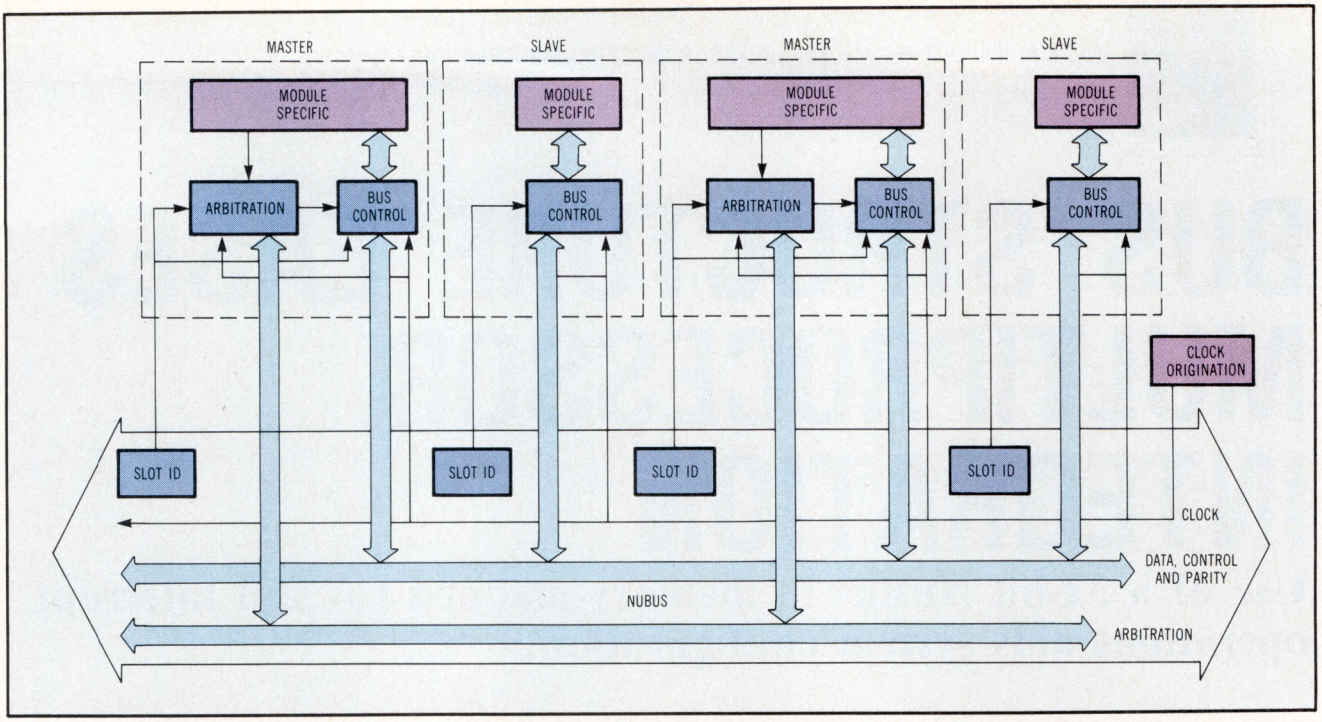


Fig 1 Identified by a unique value wired into the backplane, each module capable of participating in the arbitration mechanism is free to serve as either bus master or slave in the NuBus. A 10-MHz system clock synchronizes this 32-bit multiplexed bus to provide up to a 37.5-Mbyte/s transfer rate in block mode.

Accommodating I/O and interrupt operations within this framework provides good mapping between bus operations and the structures in memory, which can be manipulated by high level programming languages. By uniformly dealing with all resources in the address space, even a high level language such as Fortran can affect I/O and interrupt operations just by transferring a 32-bit number to some specified memory location.

TABLE 1
NuBus Bus Definitions

Classification	Signal	No. of Pins
Utility	$\overline{\text{RESET}}$	1
	$\overline{\text{CLK}}$	1
Control	$\overline{\text{START}}$	1
	$\overline{\text{ACK}}$	1
	TM0	1
	TM1	1
Addressdata	$\overline{\text{AD}}$<31..0>	32
Arbitration	$\overline{\text{ARB}}$<3..0>	4
	$\overline{\text{RQST}}$	1
Parity	$\overline{\text{SP}}$	1
	$\overline{\text{SPV}}$	1
Slot ID	$\overline{\text{ID}}$<3..0>	4
	Total signals	=49
Powerground	+5	11
	−5	8
	+12	2
	−12	2
	$\overline{\text{GND}}$	23
Reserved	$\overline{\text{RSVD}}$	1
	Total pin count	=96

Furthermore, uniform memory of the NuBus model aids device access to memory at a lower level. The small address space supported in earlier bus structures (eg, Multibus) could force programmers to deal differently with memory, depending on the source of the memory reference. For example, a CPU would access its onboard local memory with one set of addresses, while references to the same memory originating from other boards would need to incorporate an offset into that set of memory. With its single memory map, the NuBus permits any device to access any memory—even that local to a particular board—with a consistent set of addresses.

Simple structure

Driven by a central system clock, the NuBus is a synchronized bus that provides designers with a framework free from the specific control structure of a particular microprocessor. In fact, the NuBus specification imposes few constraints on system design—any module that can arbitrate for the NuBus can potentially serve as bus master (Fig 1).

No particular slot position is defined as the bus master. Instead, each slot has an identification hardwired into the backplane. Consequently, boards can be differentiated without the need for jumpers or switches. Besides the signals shown in Fig 1, the only other signals required by the NuBus are reset and power (Table 1). Although adopting a triple-height Eurocard form, the NuBus specifies use of only a single 96-pin connector and uses only 49 of those for signals—relegating the rest as extra power and ground lines.

To the system integrator, this low pin count benefits both present and future designs. Having fewer interconnection pins in current designs translates into more mechanically reliable systems. On the other hand, future systems can easily migrate to designs using VLSI bus interface chips, which demand low pin count for low cost designs.

With its 10-MHz cycle, the NuBus supports a 37.5-Mbyte/s block transfer rate across 32 multiplexed address and data lines. Although optimized for 32-bit transfers, the NuBus also supports unjustified 8-bit byte and 16-bit half-word transactions. In contrast, a justified bus like Multibus II always places 16-bit data in the least significant 16 lines, even if the address specifies data in a more significant position in a word.

A justified bus structure permits designers to attach 8- or 16-bit interfaces to a common set of lines, but at the cost of a more complex bus structure. Moreover, the NuBus's unjustified structure results in a simpler organization at the small cost of a few more transceivers.

Bus transactions

Each 100-ns NuBus cycle ensures that signals settle along the NuBus before receivers latch in the values (Fig 2). With the rising edge of the cycle, drivers assert (low) or deassert (high) signals on NuBus lines. After signals settle during the 75-ns (unasserted) portion of the bus cycle, receivers sample the signals on the falling edge of the clock signal. The 25-ns (asserted) portion of the signal helps avoid skew in bus signals.

All basic NuBus signals require a clock cycle. In addition, various NuBus signals combine to form higher level transactions, like those for read/write operations. In fact, the NuBus permits these transactions to be a variable number of clock cycles long. Consequently, although it is a synchronous bus controlled by a central system clock, the NuBus provides the adaptability of an asynchronous bus without losing the design simplicity of a synchronous bus.

All NuBus transactions involve a dialogue between a device requesting service (master) and a device providing service (slave). Unlike traditional systems where a bus slave is a device incapable of independent action, the NuBus master/slave concept simply describes the temporary relationship between two NuBus modules during a particular bus transaction. In fact, provided that it can arbitrate for the NuBus, any module can serve equally well as slave or master in the NuBus protocol.

In the NuBus, a transaction commences with a START signal from a bus master and terminates with an acknowledge (ACK) signal from a slave. In parallel with toggling the START signal, the master notifies the slave of the type of transaction by manipulating transfer mode (TM) and the lower 2 bits of the AD lines. Thus, with these four control signals,

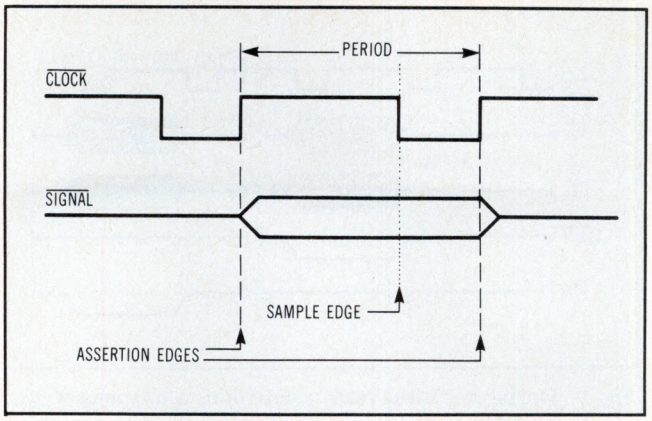


Fig 2 Each NuBus clock cycle lasts 100 ns. During the rising edge, devices place signals on the NuBus. After allowing 75 ns for these signals to settle, receivers latch in data during the falling edge of the clock signal. A 25-ns sample period helps avoid bus skew problems.

the master can initiate read/write transactions involving bytes, half-words, or words (Table 2).

Transactions across the multiplexed NuBus include separate phases for address and data. For example, in the initial phase of a read transaction, a bus master such as a CPU sets the address lines, asserts the TM control signals to indicate the type of transaction, and toggles the START line (Fig 3). When it has prepared its response, the slave replies in the latter phase of the transaction by placing the data on the AD lines, indicting the status of its response on the TM lines, and asserting ACK.

On the other hand, in a write operation, the bus master first places the address on the AD lines and toggles the TM and START control lines. In the next cycle, the bus master places the data to be written on the AD lines and waits. After it has latched the data, the addressed slave acknowledges and indicates the status of the completed transaction by setting the TM lines.

TABLE 2
Read/Write Transactions

$\overline{TM1}$	$\overline{TM0}$	$\overline{AD1}$	$\overline{AD0}$	Type of Cycle
low	low	low	low	write byte 3
low	low	low	high	write byte 2
low	low	high	low	write byte 1
low	low	high	high	write byte 0
low	high	low	low	write half-word 1
low	high	low	high	write, block
low	high	high	low	write half-word 0
low	high	high	high	write word
high	low	low	low	read byte 3
high	low	low	high	read byte 2
high	low	high	low	read byte 1
high	low	high	high	read byte 0
high	high	low	low	read half-word 1
high	high	low	high	read, block
high	high	high	low	read half-word 0
high	high	high	high	read word

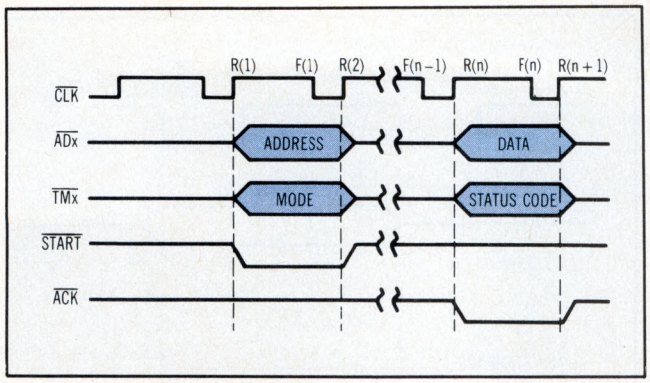


Fig 3 During a NuBus read transaction, a bus master places the address of the desired data on the bus, identifies the transfer mode (TM), and toggles the START line. When the addressed slave is ready to respond, it places the data on the address/data lines, indicates the status, and toggles the acknowledge (ACK) line.

Just as the AD lines double for address and data, the TM lines indicate transaction type during the address phase and the status of the result during the data phase. Of the four possible result codes, two results—bus transfer complete and error—correspond to positive and negative ACK codes commonly found in bus architectures.

Similarly, the third result code—bus timeout errors—is a signal used to guard against transactions targeted for nonexistent memory locations. For example, a NuBus master might initiate a transaction into the area of memory occupied by a certain module and receive a bus timeout code (from a separate logic). This indicates that the required module has been functionally removed from the NuBus.

The fourth result code—try-again-later—is a unique signal that should find a major role in multiprocessor systems. Logically indicating a result falling between error and success, the try-again-later code indicates that the master should simply defer the transaction to a later time, rather than jump into extensive exception-handling routines. This signal can find extensive applications in situations where a device serves more than one master. In dual-ported memory, for example, when a master finds an access blocked because of contention with another device using the memory, the master can sit on the bus—preventing its use by other potential bus masters—or release the bus and try again later. The NuBus try-again-later signal provides a mechanism to implement the latter, more efficient method.

Besides potentially improving bus throughput in multiprocessor systems, the try-again-later signal fills a critical need in avoiding deadlock in these systems. For example, in the Nu Machine, a separate 8088-based module acts as a converter between the NuBus and the Multibus (see Panel, "The key role of the diagnostic unit"). But, the use of such a converter can easily result in deadlock if the converter tries to access the NuBus at the same time that a NuBus module tries to access the converter. When this happens, both the converter and the NuBus master may find themselves deadlocked waiting for access to the other. The converter, however, relieves the potential deadlock by simply transmitting a try-again-later signal to the NuBus master and completing its own operation.

In addition to its role in boosting bus efficiency and mediating deadlock possibilities, the try-again-later signal can be used as a prefetch signal to slow devices. Moreover, in a bus converter to some slow bus, (rather than holding up a high speed bus like the NuBus), a converter can transmit a try-again-later signal to free the high speed bus and simultaneously access the information from the slower bus. In this way, the converter has the desired data available immediately upon the original requester's return.

Block transfers

Besides simple read/write transactions, the NuBus also supports block-mode transfers. Although some bus protocols such as Multibus II permit block-mode transfers of any length, the NuBus supports transfers only of smaller blocks—2, 4, 8, and 16 words. More compact block transfers of this sort

The key role of the diagnostic unit

Designed specifically for multiprocessor architectures, the NuBus already stars as central performer in Texas Instruments' Nu Machine. Equipped with a 68000-based CPU, the Nu Machine is designed to be independent of any particular CPU. Although such a processor-independent architecture provides system integrators with a flexible framework for crafting their own systems, it also demands an alternate approach for ensuring basic system operation in the absence of any particular CPU.

Filling this maintenance role as well as illustrating the use of the NuBus in a simple multi-CPU design, is the Nu Machine's system diagnostic unit (SDU). This unit is a separate 8088-based module with onboard memory, serial I/O for console communications, and maintenance and diagnostic test stored in onboard ROM. When the system is powered up, the SDU tests the NuBus through a series of bus transfers, identifies all boards in the system, initiates self-test routines associated with each board in the system, and signals the operator.

In addition to this maintenance work, the SDU becomes another potential master on the NuBus, acting as a converter between the NuBus and Multibus, once the system's integrity is ensured. In normal operation, the NuBus and Multibus system can operate independently. The Multibus appears as a 1-Mbyte window within the SDU's address space.

When a NuBus master addresses a memory location falling in this Multibus window, hardware-mapping logic converts the NuBus access into a Multibus reference without intervention of the 8088 CPU at bus speeds. On the other side of the window, the converter monitors each Multibus cycle for references to addresses which are mapped to NuBus. When a conversion is required, the SDU uses a Multi-to-NuBus page map to construct references to the NuBus address space.

TABLE 3

Block Length and Destination Address

$\overline{AD5}$	$\overline{AD4}$	$\overline{AD3}$	$\overline{AD2}$	Block Size Words	Block Starting Address
don't care	don't care	don't care	high	2	(AD31 to AD3) 000
don't care	don't care	high	low	4	(AD31 to AD4) 0000
don't care	high	low	low	8	(AD31 to AD5) 00000
high	low	low	low	16	(AD31 to AD6) 000000

conform more closely to the fundamental concept that a bus is a data-transfer highway freely available to any potential bus master. Furthermore, transferring arbitrarily long blocks requires a mechanism to suspend, or preempt, the transfer. This results in a more complex structure.

In addition, unlike other protocols, the master warns the slave that it is going to transfer a block and supplies the transfer length at the beginning of the operation. Supplying this information at the beginning opens the possibility for higher performance response. For example, if it is supplied with the size of the transfer beforehand, a slave has the opportunity to speed the transfer by prefetching the requested data from its storage.

By setting address bits AD2 to AD5, the bus master indicates the length of the block in the first phase of the transaction, as well as the destination address of the block (Table 3). In subsequent phases of the transaction, the bus master transmits (in block write) or receives (in block read) successive words from memory until the requested amount of words has been transmitted or an error occurs.

As each word within a block transfer is read or written, the slave uses TM0 as an intermediate ACK. The slave sends the ACK only after the final word in the block transfer has been transmitted.

One for all

Unlike other bus architectures, read and write serves for all operations on the NuBus, including I/O and interrupts, because I/O and interrupts are mapped into the NuBus's 4-Gbyte address space. In fact, all devices occupy a reserved portion of the NuBus address space specified by each slot's ID (Fig 4). Thus, the device that occupies slot 0 may occupy addresses between F0000000 to F0FFFFFF.

Originally popularized by Digital Equipment Corp's PDP-11, memory-mapped I/O operations write to memory addresses instead of using special I/O instructions or wires. Instead of a memory cell, the specified location contains a universal asynchronous receiver/transmitter command register. As a result, high level languages gain the ability to deal with I/O directly. Furthermore, the same memory management schemes that translate memory references and isolate system memory from application programs, now apply to I/O. Consequently, application programs can safely draw on a subset of system resources directly without concern that users might intrude on sensitive areas.

In the NuBus, a similar situation applies to interrupts. To initiate an interrupt in conventional systems, a device asserts a special line that runs to the CPU. When the CPU acknowledges the interrupt, the device replies with some identifying code that the CPU uses to enter an interrupt software routine. These conventional systems need only deal with the problem of detecting the source of an interrupt.

On the other hand, a multi-CPU system not only needs to specify the source of an interrupt, but must also be able to post an interrupt to a specific device. Thus, the NuBus maps interrupts into its address space so that devices can direct interrupt requests to specific devices.

Memory-mapped interrupts become particularly important in systems that can support multiple bus masters. Where a conventional approach would require a separate wire for each potential bus master, the NuBus approach provides a more flexible, less hardware-dependent approach.

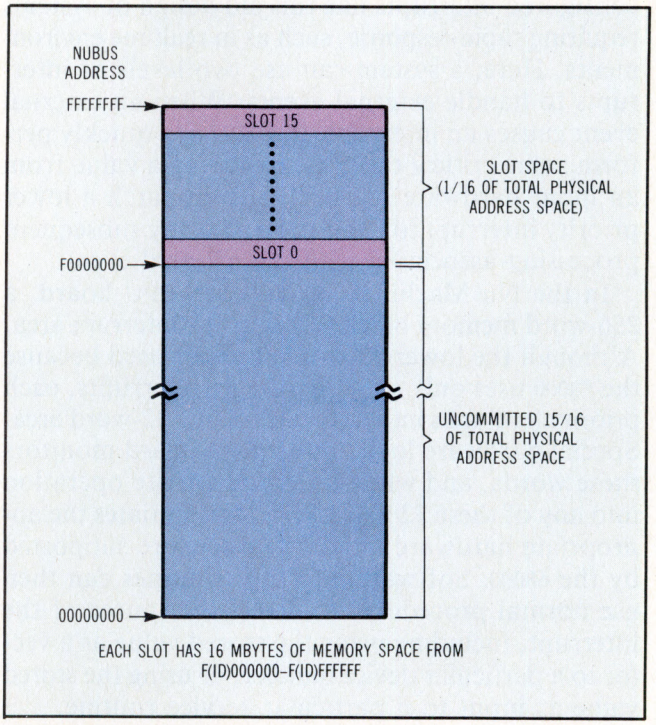


Fig 4 The NuBus associates each of its 16 slots with a reserved portion of its 4-Gbyte address space. Each slot spans an address range from F(ID) 000000 to F(ID) FFFFFF. Thus, slot 0 fills addresses from F0000000 to F0FFFFFF.

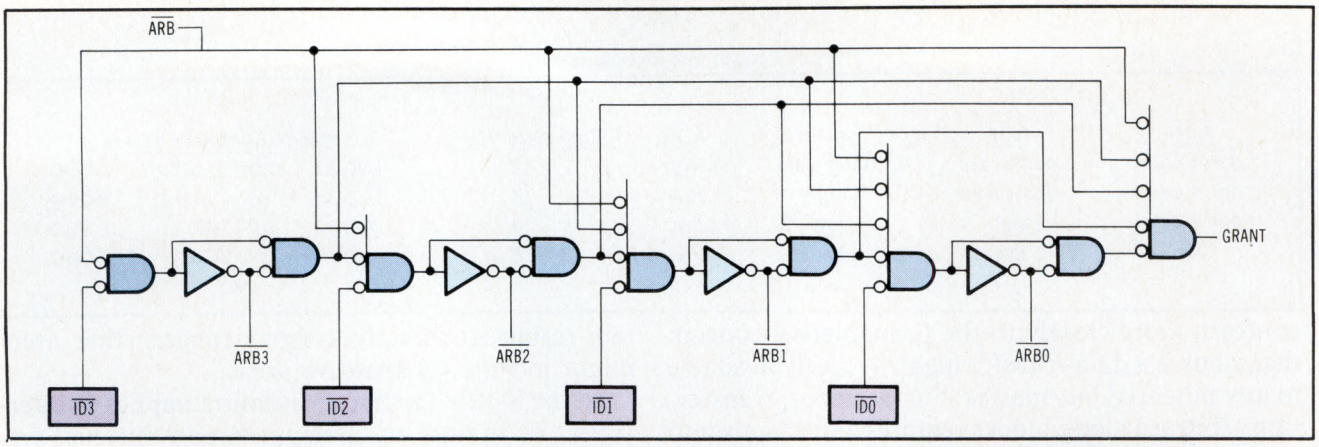


Fig 5 Simple combinatorial logic is sufficient to implement the NuBus arbitration (ARB) logic. The $\overline{ARB}x$ lines are common to all modules, while the $\overline{ID}x$ inputs are unique to each card. The $\overline{ARB}$ signal is asserted if the module is requesting the bus, while the grant signal indicates whether the $\overline{ARB}x$ lines match the slot's $\overline{ID}x$ lines.

Furthermore, as separate system resources evolve into more powerful devices, memory-mapped interrupts provide a graceful migration path. For example, devices such as disk controllers are gaining more independence from the CPU. After transferring a block of data between memory and disk, a smart controller on the NuBus can write its status word to a special location, interrupting the CPU for further action. The NuBus offers system integrators a simple mechanism to take advantage of such powerful techniques.

Using the NuBus's interrupt structures, programmers can easily implement advanced features like coroutines, in which a high level language issues an interrupt to another process. Similarly, software-configured interrupts like this aid design of systems requiring rapid response, such as in realtime environments. Here, a system can use two levels of interrupts to handle external events. When an external event causes an interrupt, the CPU can quickly perform some critical task (eg, removing a value from an input hardware register), and dispatch a lower priority interrupt to itself to handle any subsequent processing associated with the external event.

In the Nu Machine's 68000-based CPU board, a 256-word memory block serves as the interrupt area. Although the lower 32 words are not used because the 68000 uses only seven hardware interrupts, each priority level falls into a corresponding 32-word area. Special hardware logic on the CPU board monitors these words, and when it detects a write operation into any of these 32 words, the logic initiates the appropriate hardware interrupt procedure supported by the 68000. Software-interrupt routines can then use normal procedures to detect the source of the interrupt, including using the stored value as a vector to a particular device handler, or using the stored value as input to a particular service routine.

In addition to providing a consistent approach for handling I/O and interrupts, the memory-mapped approach adopted by the NuBus permits system integrators to rely on software to configure systems. In the Nu Machine, each hardware module uses a special set of memory locations within its reserved memory to specify configuration information. Thus, during a software configuration phase, an operating system can simply access these locations to obtain necessary system information. If a module is not present, the bus timeout that results during the NuBus transaction tells the operating system that the corresponding device is unavailable and should not be included in the system definition.

Critical arbitration

In any bus-oriented system with more than one potential master, the concept of arbitration is crucial. Multiprocessor systems such as the Nu Machine rely on bus arbitration mechanisms ensuring that each processor is allowed access to the bus regardless of its defined priority. The NuBus, optimized for multiprocessor architectures, provides a fair arbiration mechanism that guarantees access to any module requesting bus access.

The NuBus traces its arbitration lineage through a long history of bus protocols extending back to the S-100 bus. In the daisy chain arbitration technique, a signal propagates serially through all boards in the backplane. If it decides that it wants to assume control of the bus, a board simply does not propagate the signal.

Unfortunately, this approach does not provide a fair distribution of bus cycles. Boards closer to the daisy chain origin stand a better chance of acquiring the signal and of starving lower priority boards of bus access. Furthermore, this approach requires that all slots on the backplane be filled or bypassed with jumper cables. This is often at the cost of mechanical difficulties as jumpers fall off or users connect incorrect pins.

In the analogous software situation, scheduling algorithms can avoid starvation of lower priority tasks caught in a mix of higher priority processes. However, arbitration demands strict mechanisms integrated into the basic structure of the bus itself. The

mechanism cannot provide exceptions. During multiprocessor system design, an engineer cannot prejudice the architecture with the idea that a certain potential master should be allowed more bus cycles than others.

For example, when compared with a disk controller, an Ethernet communication module might be considered a lower priority device and assigned a correspondingly lower hardware priority. However, the data transmitted through this communication module may assume major importance in the system. Consequently, a system integrator must be certain to disassociate hardware priority on the bus from importance. Conventional daisy chained systems force designers to associate hardware priority with importance. The NuBus, however, maintains a strict approach to fairness in arbitration. No single module can be weighted to enjoy more bus activity at the expense of others.

To do this, the NuBus adopts a parallel scheme. Bus arbitration begins when one or more potential bus masters assert the bus request (RQST) line, and each attempts to place its unique ID on four open-collector lines on the bus—the arbitration (ARB) lines. Each slot is given a unique identification by a 4-bit ID code hardwired into the etch of the backplane.

In placing ID codes on the ARB bus, the boards use a strategy common to other bus architectures that ensures that only the highest ID stays on the bus. If a board sees a bit that is of a higher order than its own bits as it puts its code on the bus, the board lifts its signal. For example, if the board with ID4 attempts to place its ID on the ARB lines (assert ARB2) and finds that ARB3 is already asserted—indicating that a higher ID is already on the ARB lines—it simply removes its signal.

Implemented with simple combinational logic (Fig 5), this technique dictates that only the highest ID remains on the ARB lines at the end of the two bus cycle arbitration contests. Because this arbitration contest occurs over separate lines in parallel with read/write transactions, this mechanism does not cut into bus throughput. In fact, because typical NuBus transactions require at least two bus cycle, a new bus master will be ready to initiate its own transactions when the previous bus master has completed its turn on the NuBus.

Determining fair arbitration

Although this arbitration scheme does seem to involve a priority aspect, fairness counteracts it. If three boards request the NuBus simultaneously, the highest numbered board wins. However, fairness ensures that the lower numbered pair of boards will gain access before that particular higher numbered winner gains the bus again.

Fairness in the NuBus is a simple protocol—once the RQST line is asserted, no other boards are allowed to request the bus. For example, if three boards want the bus at the same time, they will all assert the RQST line at the same time. The highest numbered board will win out, but the other two boards will still be asserting the RQST line. After the last board of the three uses the bus, the RQST line will become deasserted. This then starts another arbitration contest.

In addition to serving as the medium for bus request and achieving arbitration fairness, this single RQST line also mediates bus locking in the NuBus. Bus locking is typically used for indivisible test and set instructions used to implement semaphores for interprocess communication. In bus locking, once a device wins the bus, it simply continues to assert RQST and maintain its ID on the ARB lines. During the next arbitration, the same board will always win because, by the definition of fairness, its ID will always be the highest numbered in the current arbitration round. The NuBus needs no extra wire, mechanism, or state to accommodate bus locking. This mechanism is enfolded within the larger concept of fairness.

Introducing TI's Climb on the 32-

Nu Machine. bit NuBus now.

The Nu Machine™ Computer. The first system in the Texas Instruments Nu Generation Computer family. The only system now available built on a modern 32-bit bus. The processor-independent NuBus™ architecture helps meet your advanced-technology design requirements today. And tomorrow.

First high-performance 32-bit bus

The NuBus technology, designed at M.I.T., is optimized for 32-bit data and address transfers. Its 37.5-Mbyte/sec bandwidth combines with an elegant arbitration scheme to ensure fast and fair data flow.

Innovative, flexible architecture

The NuBus design was developed to support sophisticated system architectures and eliminates the built-in obsolescence of processor-dependent systems. It lets you concentrate on developing applications, not architecture. Your significant investments are protected as new technologies develop.

The Nu Machine's open architecture solves your make vs. buy dilemma. Multiple-processor configuration support combines with the NuBus high bandwidth, high-resolution graphic displays, cache memory, and high-speed disks to make the Nu Machine system attractive to sophisticated end-users, systems integrators, and OEMs in the engineering and scientific marketplace.

Anticipating industry trends, the power and expandability of TI's Nu Machine allow it to accept 32-bit processors of the future.

Open system supporting industry standards

TI's Nu Machine system is currently available with a 10-MHz 68010 processor supporting a UNIX™-based operating system with enhancements for windowing and high-resolution displays.

Those who want to design their own system processors and controllers can now license the NuBus design from Texas Instruments.

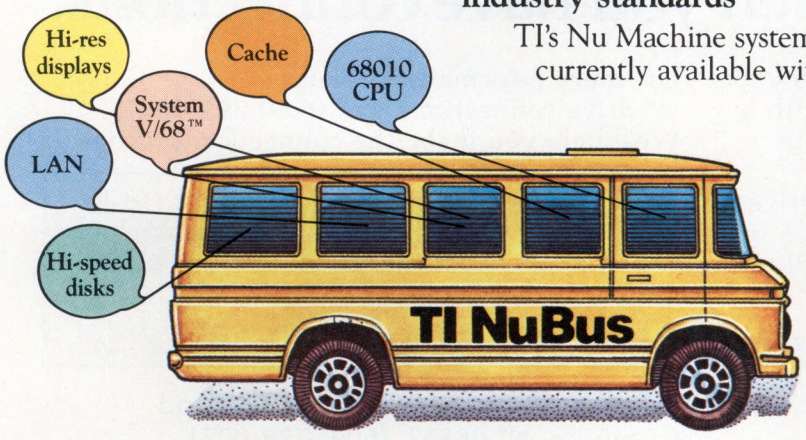
Also, a NuBus-to-Multibus™ converter allows the use of existing interface cards and peripherals from third parties.

The system you can build on from now on

Because its high performance and flexibility are designed for the long run, TI's Nu Machine can be updated when other systems are outdated.

And, Nu Machine computers are backed by TI's service and customer-support network and by TI's commitment to quality and reliability.

To climb on the NuBus bandwagon, call toll-free: 1-800-527-3500. Or write Texas Instruments Incorporated, P.O. Box 402430, Dept. DNA2Ø3ØS, Dallas, Texas 75240.



Combining innovative NuBus architecture with advanced graphics, powerful peripherals, and UNIX-based software, TI's Nu Machine provides the outstanding performance and flexibility required by scientific and engineering systems designers.

TEXAS INSTRUMENTS

Creating useful products and services for you.

27-7379

CIRCLE 72