BUS ARBITER STREAMLINES MULTIPROCESSOR DESIGN

Arbiter coordinates 8- and 16-bit microprocessors' access to a shared, multimaster bus and offers flexible operating modes to accommodate different system configurations

James Nadir and Bruce McCormick

Intel Corporation
3065 Bowers Ave, Santa Clara, CA 95051

Performance improvements and cost reductions afforded by large scale integration technology have spurred the design of multiple microprocessor systems that offer improved realtime response, reliability, and modularity. In multiprocessing, more than one microprocessor shares common system resources—such as memory and input/output devices—over a common multiple processor bus (Fig 1). This concept allows system designers to partition overall system functions into separate tasks that each of several processors can handle individually and in parallel, increasing system performance and throughput.

The 8086 family of 16-bit microprocessors and support components permits the designer to select only those components that are necessary to meet cost and performance requirements. One method for achieving this building block approach to system design uses a compatibility mechanism of the 8086 and the MULTIBUS™ multiprocessing bus standard. However, multiprocessing systems require devices that can coordinate the use of global or shared resources. The 8289 bus arbiter (Fig 2) provides this multiprocessing coordination in conjunction with MULTIBUS architecture.

MULTIBUS Approach

MULTIBUS architecture in a multiprocessor system allows each processor to work asynchronously. Therefore, a fast microprocessor operates at its own speed regardless of the speed of the slowest microprocessor. This technique tolerates duty cycle and phase shift variations, and offers hardware modularity. When new system functions are desired, additional microprocessors can be integrated without impacting existing task partitioning.

The MULTIBUS approach implements this asynchronous processing structure by synchronizing all microprocessor bus requests to a high frequency reference system bus clock that can operate at up to 10 MHz. Synchronized requests are then resolved by a priority encoder. As a result, the number of resolving circuits common to all microprocessors is minimized. The synchronizing or arbitrating function is integrated into the bus arbiter, allowing it to resolve arbitration problems of a shared system bus in a multimaster, multiprocessing environment.

Critical code sections in memory can be identified by a flag or word, called a semaphore, which is set by one of the microprocessors. The bus arbiter prevents use of the shared memory bus while a microprocessor is setting the semaphore, creating a "locked test and set" condition. In addition, the bus arbiter provides a flexible, definable set of bus modes so that a designer can configure a system to meet a variety of applications.

The bus arbiter operates in conjunction with a bus controller that generates memory and input/output (I/O) signals to interface the multiprocessors to a multimaster system bus (Fig 3). Unaware of arbiter presence, a microprocessor sends out control signals as though it has exclusive use of the system bus. If a microprocessor does not have use of the multimaster system bus, the bus arbiter prevents the bus controller, data transceivers, and address latches from accessing the system bus, ie, all bus driver outputs are forced into the high impedance state. Since system commands are not issued, no acknowledgement (transfer acknowledgement) is returned, causing the microprocessor to extend its transfer cycle by entering into wait states. The microprocessor extends its transfer cycle until the bus arbiter acquires access to the multimaster system bus, then,

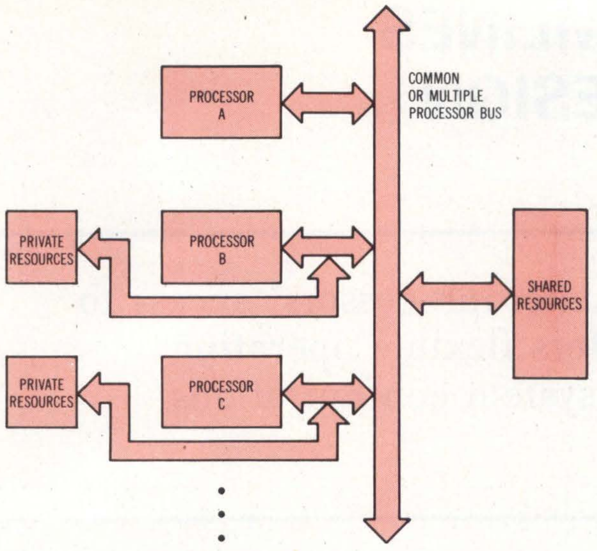

Fig 1 Multiprocessing system using common bus. Several processors sharing expensive resources, such as disc drives or line printers, is efficient and cost-effective. Bus contention problems inevitably arise on common system bus unless steps are taken to arbitrate usage

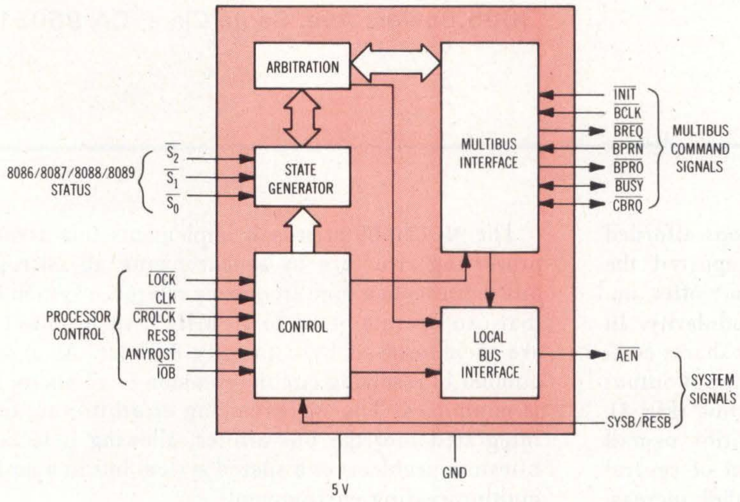

Fig 2 Bus arbiter block diagram. 20-pin, 5-V-only, bipolar arbiter for use with medium to large multimaster, multiprocessing systems provides system bus arbitration for systems with multiple bus masters, as well as bipolar buffering and drive capability

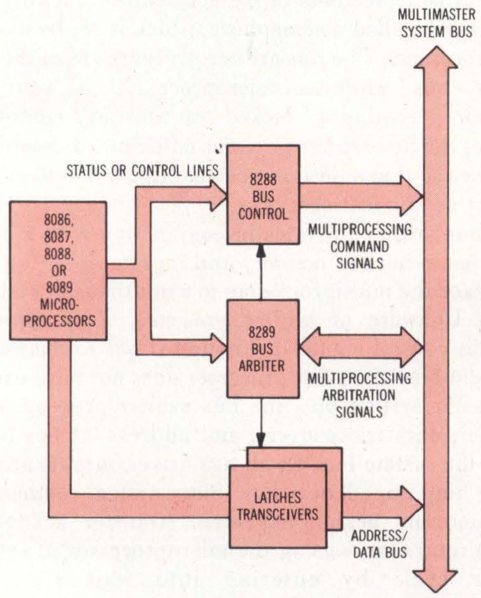

Fig 3 Bus arbitration chip aids multiprocessor design. 8-bit 8088 and 16-bit 8086, 8089 microprocessors are designed for use in multiple microprocessor systems. To prevent contention for common system bus, arbiter provides several methods for arbitrating bus use

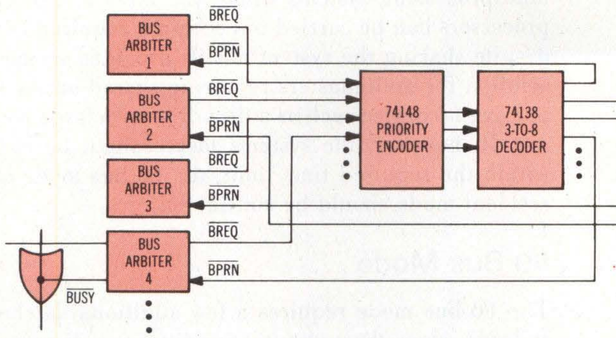

Fig 4 Parallel resolving technique synchronizes asynchronous requests. Requests for common bus use ($\overline{\text{BREQ}}$) are sent by each microprocessor's bus arbiter to encoder-decoder circuit, which responds to arbiter with highest priority by returning bus priority in signal ($\overline{\text{BPRN}}$). Bus arbiter then provides its microprocessor access to system bus

the arbiter allows the bus controller, data transceivers, and address latches to access the system bus.

After the bus controller issues its control line signal and a data transfer has taken place, a transfer acknowledge signal is returned to the microprocessor. Then the microprocessor completes its transfer cycle. Thus, the arbiter serves to coordinate microprocessor, or bus master, access to the multimaster system bus.

Priority Resolving Techniques

Since there can be many bus masters on a multimaster system bus, a technique for resolving simultaneous requests among bus masters must be provided. The bus arbiter offers several resolving techniques; all are based on the concept that at a given time, one bus master has higher priority. These techniques include parallel priority resolving, serial priority resolving, and rotating priority resolving.

Parallel Priority Resolving Technique

In the parallel priority resolving technique, a separate bus request ($\overline{\text{BREQ}}$) line is connected to each of several arbiters on the multimaster system bus (Fig 4). Each $\overline{\text{BREQ}}$ line enters a priority encoder which generates, as output, the binary address of the highest priority $\overline{\text{BREQ}}$ line asserted at its inputs. The output binary address, after being decoded, selects the corresponding bus priority in ($\overline{\text{BPRN}}$) line to be returned to the highest priority requesting arbiter. The arbiter receiving priority ($\overline{\text{BPRN}}$ true) then allows its associated bus master access to the multimaster system bus as soon as the bus becomes available.

Even when one bus arbiter gains priority over another arbiter, it cannot immediately seize the bus. It must wait until the present bus occupant completes its transfer cycle, ensuring transfer integrity. Upon completing its transfer cycle, the bus occupant determines that it no longer has priority and surrenders the bus, releasing $\overline{\text{BUSY}}$.

Serial Priority Resolving Technique

The serial priority resolving technique eliminates the need for a priority encoder/decoder arrangement by daisy chaining the bus arbiters. This setup is accomplished by connecting the higher priority bus arbiter bus priority out ($\overline{\text{BPRO}}$) line to the $\overline{\text{BPRN}}$ line of the next lower priority arbiter (Fig 5). The highest priority bus arbiter has its $\overline{\text{BPRN}}$ line grounded, signifying to the other arbiters that is always has highest priority when requesting the bus. Priority is passed in series

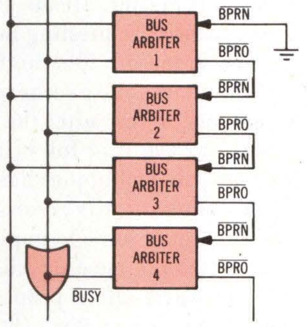

Fig 5 Serial priority resolving technique. Simpler daisy chain scheme provides effective bus arbitration for system designs that do not have more than three arbiters and microprocessors vying for bus access. Connected in order of priority from top to bottom, first arbiter has priority at all times because of grounded state of its $\overline{\text{BPRN}}$ line. When not using bus, arbiter 1 sends low state signal to next higher priority arbiter via its $\overline{\text{BPRO}}$ line

from a higher priority arbiter on the chain, when it does not need the system bus, to any requesting lower priority arbiter.

Rotating Priority Resolving Technique

Arrangement of the rotating priority resolving technique is similar to that of the parallel priority resolving technique except that priority is dynamically reassigned. The priority encoder is replaced by a more complex circuit that rotates priority in standard time increments among requesting arbiters, guaranteeing each arbiter equal access to the multimaster system bus.

In all three techniques, lower priority masters obtain the bus when a higher priority processor is not accessing the system bus. A strapping connection is available, however, that allows the multimaster system bus, completing its transfer cycle, to be surrendered immediately to any bus master requesting the bus, other than itself, regardless of priority. If there are no other bus masters requesting the bus, the arbiter maintains the bus as long as its associated processor has not entered the halt state. The bus arbiter does not voluntarily surrender the system bus and has to be forced off by another bus master. Means and conditions do exist whereby a lower priority requesting bus master can acquire the system bus from an idle higher priority bus master. This action minimizes the overhead required to obtain use of the system bus; so that after the bus has been acquired, the processor can use it at full efficiency.

Each of the three priority techniques has advantages and disadvantages. The rotating priority resolving technique requires an extensive amount of logic to implement, while the serial technique can accommodate only a limited number of bus arbiters before the daisy chain propagation delay exceeds the multimaster sytem bus clock ($\overline{\text{BCLK}}$) period. The parallel priority resolving technique is generally the best compromise. It allows many arbiters to access the bus without requiring excessive logic for implementation.

System Bus Modes

The bus arbiter provides several definable system bus mode configurations for microprocessors. In the I/O peripheral bus mode, the arbiter permits a microprocessor access to both a private I/O peripheral bus and a multimaster system bus. In the resident bus mode, the arbiter allows a microprocessor to communicate over both a resident bus and a multimaster system bus. A resident bus is defined as a private bus that has both memory and I/O devices, as opposed to an I/O peripheral bus that has only I/O devices. Other configurations provide communication across several multimaster buses.

A particular configuration determines the technique by which the arbiter requests and surrenders the system bus. If the arbiter is configured to operate with a non-I/O microprocessor (normal processor), which has access to both a multimaster system bus and a resident bus, then the arbiter requests the use of the multimaster system bus only for system bus accesses. While the processor is accessing the resident bus, the arbiter permits a lower priority bus master to seize the system bus. An I/O processor configuration with both I/O peripheral and system buses behaves similarly.

Single-Bus Mode

The single-bus mode is the simplest mode. It is sufficient for multiprocessing systems where the tasks of several microprocessors can be carried out within a required time frame despite sharing the system bus. It provides an inexpensive solution for multimasters requiring shared access to an expensive I/O device such as a disc drive or a large memory array. If, however, the systems tasks cannot be carried out within the required time limit, the I/O bus mode or system resident mode should be considered.

I/O Bus Mode

The I/O bus mode requires a few additional latches and is suitable when throughput considerations dictate that the overall bus structure be separated into an I/O bus and a memory or system bus. This mode is commonly used with the 8089 I/O microprocessor, in its remote configuration, to separate I/O space from memory space. With this processor, all instructions operate on either system or I/O address space, treating all peripherals as memory mapped devices. Memory for program code or buffering can be placed on either the system bus or the local I/O bus. The 8086 and 8088 microprocessors are constrained to exclusive use of I/O instructions when referencing I/O space. If this constraint is a limitation and it becomes desirable to allocate some of the processor functions to private resources, then the resident bus mode should be considered.

Resident Bus Mode

The resident bus mode allows maximum flexibility for a microprocessor, yielding access both to local resources and to system resources. The central processing unit (CPU) can interact with local resources at full speed without contention on the system bus. System bus accesses can be minimized to those of swapping in and out from mass storage or the use of expensive resources that should not be duplicated on the processor local bus. By using a programmable read only memory (P/ROM) for memory mapping, memory space is altered easily.

Bus Interfaces

To fully describe the bus arbiter functions, each of the three operating modes is examined. Typical examples describe a single-bus configuration, an I/O bus configuration, and a resident bus configuration.

Single-Bus Interface

Fig 6 shows a typical multiprocessing system configuration with the bus arbiter in the single-bus mode. In this system design, each of the three bus masters is assigned a priority ranging from priority 1, the highest, to priority 3, the lowest. Priority is established using the parallel priority scheme; disregard, for now, the dashed signal interconnects.

Each bus arbiter monitors its associated processor and issues a $\overline{\text{BREQ}}$ whenever this processor requests bus access. A common clocking signal, $\overline{\text{BCLK}}$, runs to each arbiter in the system.

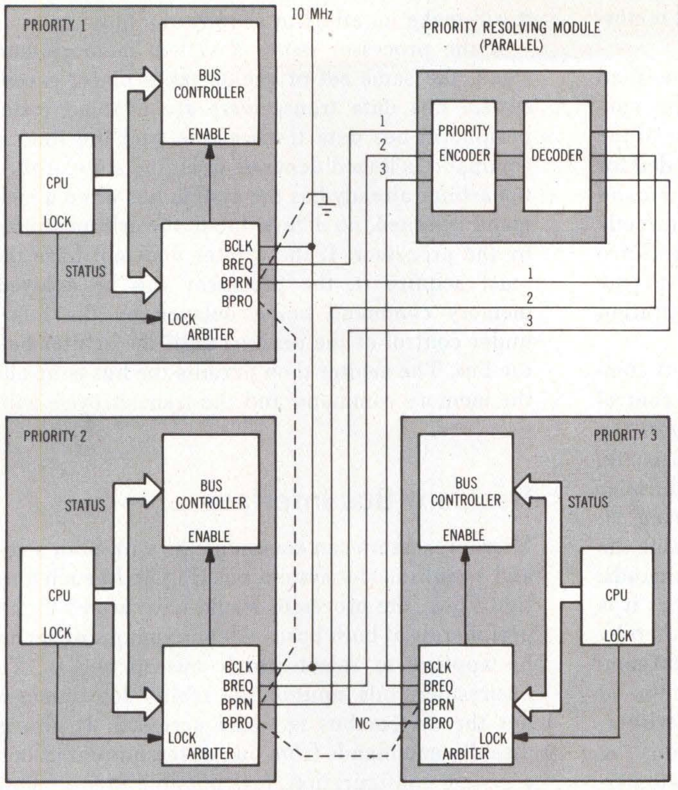

Fig 6 Parallel and serial resolving techniques. Each microprocessor circuit module and its bus arbiter (color) are shown connected in a parallel mode (solid lines) or serial mode (dashed lines) configuration

If the serial priority resolving mode is used, the system is connected by the dashed signal lines from the $\overline{\text{BPRO}}$ of one arbiter to $\overline{\text{BPRN}}$ of the next lower priority arbiter (Fig 6). The $\overline{\text{BREQ}}$ lines are disconnected, and the priority encoder/decoder arrangement is removed. This serial priority mode is more straightforward than the parallel priority mode except in that the daisy chain propagation delay from the highest priority bus arbiter $\overline{\text{BPRO}}$ line to the lowest priority bus arbiter $\overline{\text{BPRN}}$ line, including the setup time requirement ($\overline{\text{BPRN}}$ to $\overline{\text{BCLK}}$), cannot exceed the $\overline{\text{BCLK}}$ period. This period is nominally 100 ns, but it can be stretched by slowing down the maximum clock frequency of 10 MHz. The penalty is longer waits for bus arbitration.

This configuration dictates that the number of arbiters that can be daisy chained together for a given $\overline{\text{BCLK}}$ frequency be limited. Of course, the lower the $\overline{\text{BCLK}}$ frequency, the more arbiters can be daisy chained together. Three arbiters can be daisy chained when using the maximum $\overline{\text{BCLK}}$ frequency of 10 MHz.

How quickly bus arbiter 1 can acquire the bus depends upon the configuration and the strapping connections of the arbiter from which it is trying to acquire the bus. For example, if the $\overline{\text{LOCK}}$ input to arbiter 2 is active (low) at the time, then arbiter 1—even though it is of higher priority—cannot acquire the bus until after $\overline{\text{LOCK}}$ is released (goes high). Another factor to be considered is the microprocessor state, in its transfer cycle, at the time the arbiter is instructed to yield the bus. If the transfer cycle has just started, it will take longer for the bus to be released than if the cycle is just ending.

Higher priority bus masters force a lower priority bus master arbiter to surrender the bus by the reassignment of priority. If generating a $\overline{\text{BREQ}}$, a higher priority bus master would cause the present bus occupant to lose its $\overline{\text{BPRN}}$. Lower priority bus masters acquire the bus by pulling down the open collector signal, common bus request ($\overline{\text{CBRQ}}$). The present bus occupant recognizes $\overline{\text{CBRQ}}$ whenever it is not accessing the system bus; when it is activated, the bus is released. Priority is established to the next highest requesting aribter, and the requesting arbiter then acquires the bus.

I/O Bus Interface

In the I/O bus mode, the processor communicates with and controls a host of peripherals over the peripheral bus. An 8089 I/O microprocessor can use either memory or peripherals on its local bus. When the I/O processor needs to communicate with system memory, it is done over the system memory bus. Fig 7 shows a typical I/O processor in its REMOTE mode. Resident memory exists on the peripheral bus to provide programmed I/O routines and buffer storage. Resident memory is treated as an I/O peripheral. When a peripheral device needs servicing, the I/O processor accesses resident memory for the proper I/O driver routine and services the device, transmitting or storing peripheral data in a buffer storage area of resident memory (or sending it directly to system memory if necessary). The resident memory buffer storage area can then be emptied or replenished from system memory via the system bus. Using the I/O bus interface allows an I/O processor the capability of executing from local memory (on the peripheral bus) concurrently with a host processor, enhancing system performance and removing the burden of I/O from the host processor.

Like the arbiter, the bus controller also must be notified of the operating mode. In the I/O bus mode, the bus controller issues I/O commands independently of the state of the arbiter. It is assumed that all I/O commands are intended for the I/O bus and, hence, that there is a separate I/O command bus from the controller. All I/O commands are sent directly to the I/O bus and are not influenced by the arbiter. Since memory commands are assumed to be directed to the system bus, they must still be influenced by the arbitration mechanism provided by the arbiter.

For example, suppose the processor issues an I/O command. The bus controller generates the necessary control signal to latch the I/O address and to configure the transceivers in the correct direction. In the I/O bus mode, the peripheral data enable (PDEN) pin of the controller serves to enable the I/O bus data transceivers during I/O commands. Similarly, data enable (DEN) serves to enable the system bus data transceivers during memory commands. Signals PDEN and DEN are mutually exclusive, so that it is not possible for both sets of transceivers to be on, thereby avoiding contention between the two sets. Since the I/O commands are generated independently of the arbiter in the I/O bus mode, the I/O bus has no delay effects due to the arbiter. During the time in which the processor is accessing the local I/O bus, the arbiter—if it already has the bus—will permit it to be surrendered to either a higher or lower priority request (via $\overline{\text{CBRQ}}$) independently of where the processor is in its transfer cycle (for example, independent of the machine state). If the arbiter does not already have the bus, it will make no effort to acquire the bus.

If the processor issues a system memory command instead, the same set of events takes place, except that the system bus data transceivers are enabled instead of the peripheral bus data transceivers, and the time at which a command is issued depends upon the state of the arbiter. If the arbiter already has the system bus when a memory command is issued, no delays due to the arbiter will be detected by the processor. If the arbiter does not have the bus and must acquire it, the processor will be delayed (via the memory command being delayed by the bus controller under control of the arbiter) until the arbiter has acquired the bus. The arbiter then permits the bus controller to issue the memory command and the transfer cycle will continue.

Resident Bus Interface

Microprocessors can communicate with both a resident bus and a multimaster system bus (Fig 8). In such a system configuration, the processor would have access to memory and peripherals of both buses. Memory mapping techniques can be applied to select which bus to access. The system bus/resident bus input on the arbiter determines whether or not the system bus is to be accessed. It also enables or disables commands from one of the bus controllers. In such a system configuration, it is possible to issue both memory and I/O commands to either bus and, as a result, two bus controllers are needed, one for each bus.

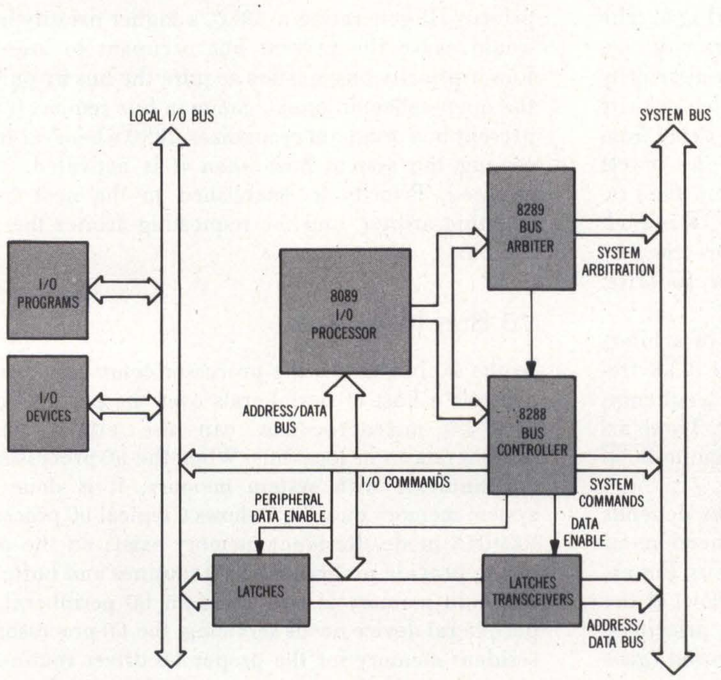

Fig 7 I/O bus interface. By halving bus structure into local I/O bus and common system bus, 8089 I/O processor has exclusive access to its peripherals but can also communicate with system bus. Commands to local bus are not under control of bus arbiter and will never be delayed by it. Requests for system bus must proceed through arbitration and can be delayed in event that arbiter does not have access to bus at time of request. Only one bus controller is needed to implement this technique

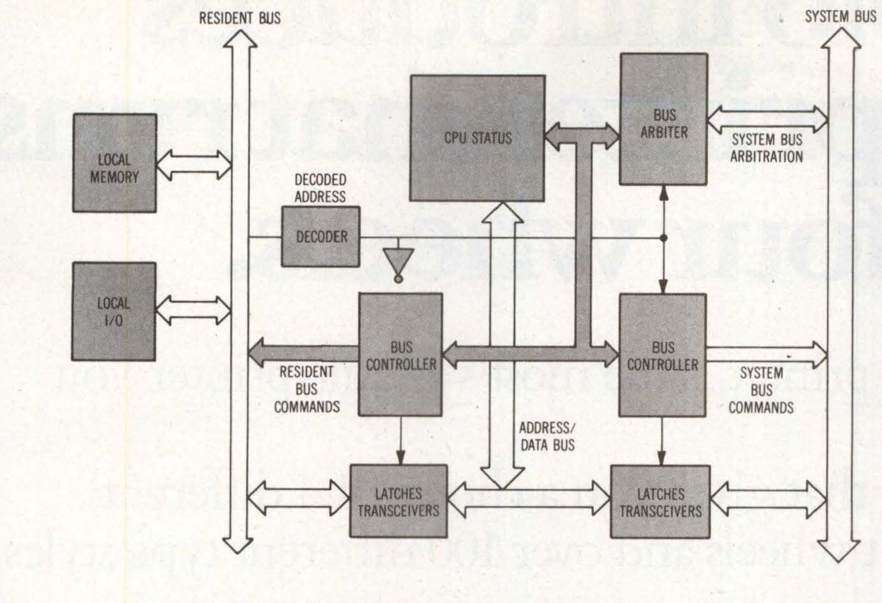

Fig 8 Resident bus scheme for additional performance. In resident bus mode, single 8086 or 8088 has access to both common system bus and local resident bus. Processor can be handling resident bus-oriented operation while other system processors use system bus. Many tasks can be carried out simultaneously, increasing overall performance. As in I/O bus mode, only one arbiter is required but two bus controllers must be used

In Fig 8, memory mapping techniques are applied on the resident bus side of the system rather than on the multiprocessor or system bus. Both sets of address latches (resident bus and system bus) are latched with the same address in this case, by their respective bus controllers. The system bus address latches, however, may or may not be enabled, depending upon when the arbiter has bus access. The resident bus address latches are always enabled; hence, the memory mapping technique is applied to the resident bus.

A simpler system with an 8086 or 8088 microprocessor can exist if it is desirable to have only P/ROM, read only memory, or read only peripheral interfaces on the resident bus. These microprocessors additionally generate a read signal in conjunction with the bus controller signals. By using this read signal and memory mapping, the microprocessors can operate from local program store without having the contention of using the system bus. Using this technique eliminates the need for a second bus controller.

In actual operation, both bus controllers respond to the processor status line and both simultaneously issue an address latch enable (ALE) strobe to their respective address latches. Both bus controllers issue command and control signals unless inhibited. The purpose of the memory mapping circuits is to inhibit one of the bus controllers before contention or erroneous commands can occur.

Summary

A bus arbiter brings a powerful dimension to system design architectures by allowing 8-bit and/or 16-bit microprocessors to execute easily in a multimaster, multiprocessing environment. With the flexible modes of the arbiter, a designer can define one of several bus architectures to meet cost and performance needs. Modularity, improved system reliability, and increased performance are some of the benefits that a multiprocessing system provides.

James Nadir is a project engineer at Intel and has designed some of the 8086 microprocessor family support circuitry and their interface to the MULTIBUS™ system bus. He graduated from Rutgers University with a BSEE.

Bruce McCormick is one of the product managers for the 8086 family of microcomputers. His experience includes applications and product engineering work as well as product marketing at Intel Corporation. He holds BSEE and MSEE degrees from Purdue University, and an MBA from Santa Clara University.

How valuable is this article to *you*?

High 704 **Average 705** **Low 706**

Please circle the appropriate number in the "Comments" box on the Inquiry Card.