6

Input and Output Operations

We have discussed several of the characteristics attributed to a machine. including the methods of information representation and the instruction set. We have also discused methods of designing the functional elements, such as the arithmetic unit or the control unit. But it is not sufficient to compute; the results of the computation must be made available to other systems. These systems may be other computers, computer peripherals. or similar devices. Eventually, the information may need to be presented in a form easily understood by humans; many interface systems convert information not only into readable text, but also graphic images. synthesized sound, or some *other* suitable form.

The term "input" is attached to the process of transferring information into the computer, and "output" to the transfer of information out of the machine. When both are possible, it is simply "I/O." In this chapter we will discuss the methods used to perform these transfers, some of which we have already alluded to in the consideration of instruction sets. This will include mechanisms used for asynchronous and synchronous bus transfers, time multiplexing of information on buses, and so on. We will also consider arbitration techniques, which decide who is the "owner" of a bus when a transfer is made. And we will include both programmed control and direct memory transfers to move information. Included with the discussions are a number of examples that illustrate the concepts and techniques. Once the ideas are understood at both the conceptual and implementation level. I/O systems and interface modules can be more easily designed and understood.

The instruction set architecture of a machine will determine the apparent organization of the I/O system. That is, the mechanisms envisioned for system I/O will be one of the factors considered in the process of the creation of the instruction set of the system. In many respects, the computer system will be judged by its ability to coordinate information transfer in a reasonable fashion. A more comprehensive view of the total system impact is obtained by considering computer system performance from a systems aspect, taking into account the characteristics of the CPU, the peripheral devices, and the transfer mechanisms. (See, for example, [LaZa84].) Our intention is to understand the principles utilized in the transfer mechanisms.

6.1. Asynchronous Bus Transfers

The block diagram of Figure 6.1 indicates that a number of functional units can exchange information over a common communication medium: the bus. The transfer of information will begin when one of the modules recognizes a need to communicate with another module. This need will result from any of a number of mechanisms, such as a processor module that must obtain the status of an interface module, or an I/O module that must transfer information into system memory. If a module has the ability to control the bus. we call it a "bus master." In general, there will be several bus masters in a bus-oriented system. When a master needs to transfer information, it will request ownership of the bus. The process of allocating control of the bus to a bus master is called arbitration, and we will discuss arbitration mechanisms in a later section. When a master has obtained control over the bus, it then initiates a bus transfer by activating the appropriate lines. The module activated by this transaction (the one that responds to the master) is called the "bus slave." The set of rules or algorithm utilized in this process is called the "bus communication protocol." This protocol will identify the sequence of events to occur in the process of transferring information. and specify the timing requirements of the transfer.

In this section we will discuss the exchange of information over the lines assuming an asynchronous protocol. That is, the modules of the bus system do not share a common clock, and the transfer proceeds in an asynchronous manner. In the communication process, the master and the slave assert signals on common communication lines in a predetermined manner so that the transfer can proceed. We will assume that the arbitration process has been completed and that the master is in control of the bus. The master is now capable of initiating the transfer. and will do so by activating the appropriate bus lines according to the defined protocol. The bus lines (except power and ground, which are also distributed along the bus) belong to one of three groups: address, data, or control, as shown in Figure 6.2.

The address lines are used to identify the target of the transaction. That is, the master places an address onto the address lines that will uniquely identify the location to be used for the transfer. The number of address lines that can be used for this function determine the number of addressable locations, since N lines are capable of selecting one of 2^N locations. This address is the only mechanism the master has to identify the target module. All of the modules that can respond to addresses to perform transfers are connected to the address bus, and they receive

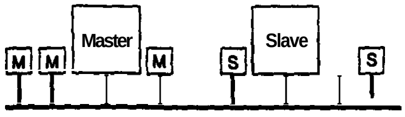


Figure 6.1. Module Organization for Bused Systems.

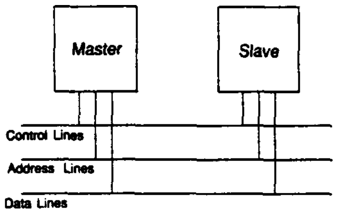
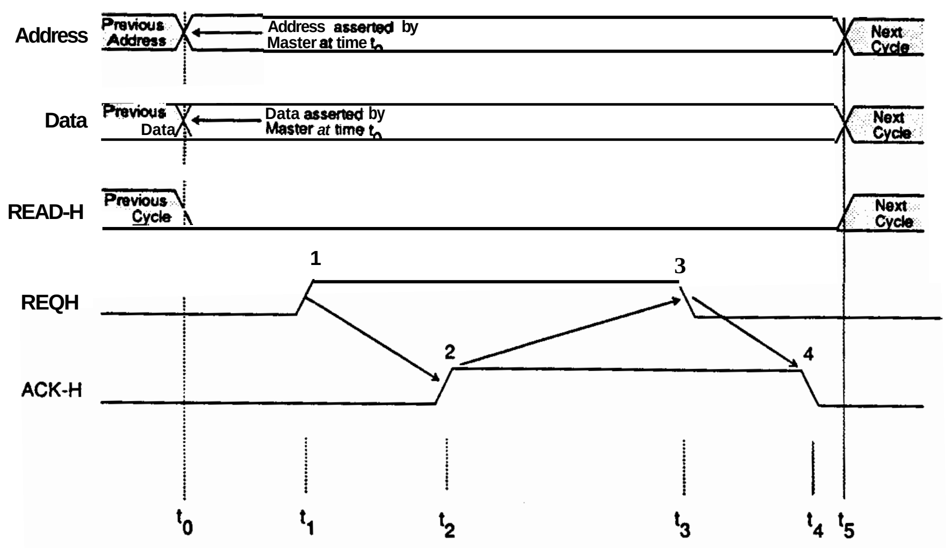


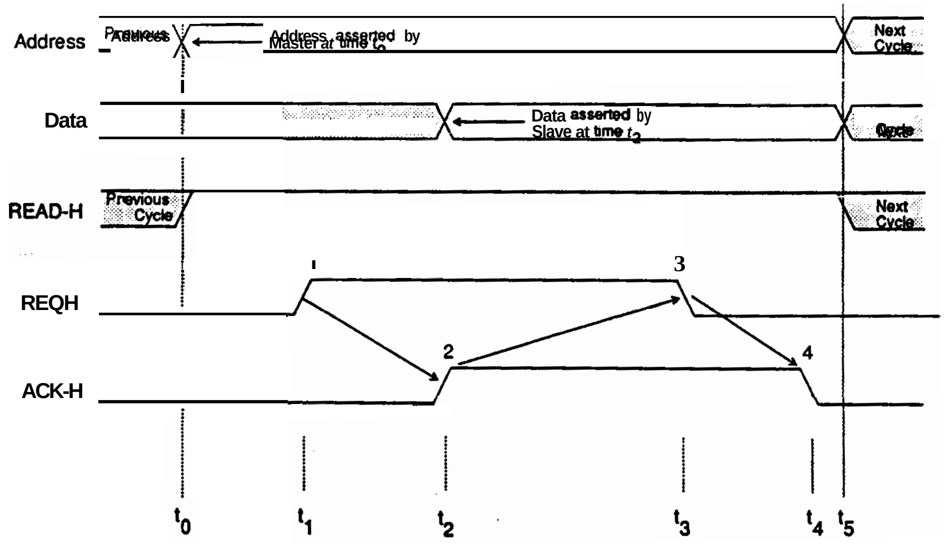
Figure 6.2 Bus Lines Connecting the Master and the Slave.

this address and compare it to their assigned address space. The address should identify only one module: if more than one module recognizes the address. the transaction will not function properly. (As usual. there are exceptions to this rule, which we will note later.) Once a master has initiated a transfer. it will allow a predetermined amount of time for the address comparisons, then proceed with the transaction. The slave module with an assigned address matching the target address will respond to the master, and the transaction will proceed governed by the assertion of the control lines. The control lines are used to synchronize the action between the master and the slave modules. The mechanism for this is shown in Figure 6.3. This figure shows the address lines as a group. the data lines as a group, and three of the control lines. A number of other control lines will he involved with the arbitration mechanism. but for this discussion we will limit ourselves to the three control lines identitied in the figure: READ-H. REQ-H. and ACK-H. The READ-H line identifies a read transaction when it is asserted. That is. when it is high, the master module is reading a location fmm the slave module. When the READ-H line is not asserted (when it is low), the master module is writing to the slave module. The READ-H line has the same timing requirements as the address lines, which are explained in conjunction with the other control lines.

The two lines that control the timing and sequence of the events involved in the transaction are the request line (REQ-H) and the acknowledge line (ACK-H). The write cycle proceeds as shown in Figure 6.3(a). The master, which has already obtained control of the bus, asserts the address of the desired location. This time is identified as t_0 in the figure. A finite time is required for this address to propagate to all of the slave modules and be decoded by them. so the master must wait for a specific period before asserting the request line. The amount of time required is a function of the technology in which the hardware is implemented, and the physical and electrical characteristics of the bus. When the required time period has passed, the master asserts the request line (time t_1). This is then accepted by all of the slave modules, but only the module with the matching address will respond. When the slave has performed the requested action, which in this case is to accept the data on the data lines, the slave module asserts the acknowledge line (time t_2). When the master detects the assertion of the acknowledge line. it recognizes that the work of the transaction has been completed. So it releases the request line (time t_3), and when the slave detects the release of the request line, it releases the acknowledge line (time t_4). The master must keep the address lines asserted after the release of the request line to prevent any spurious action that may occur if the address changes before the release of the request



(a) Write Cycle



(b) Read Cycle

Figure 6.3. Read and Write Transactions for Asynchronous Handshake Protocol.

line has propagated through the decode logic of the slave modules. This may be accomplished by holding the address lines for a specific time after the release of the request signal, or until the master detects the release of the acknowledge by the slave module. This mechanism is sometimes called the four event bus transfer. since four events (1 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ 4 in Figure 6.2) are involved in performing the transfer.

The read transaction is almost identical to the write, and the appropriate lines are shown in Figure 6.3(b). The major differences are that the read control line is asserted and that the data is now asserted by the slave module. The master begins the transaction as before, by asserting the address and waiting the necessary time for propagation delay and skew. Even though the master may assert all of the address lines simultaneously, they will not all arrive at the decoder of the slave modules simultaneously, since the electrical characteristics of the bus and the propagation delays of the address lines may be different from one another. The time difference from the arrival of the first signal to the arrival of the last signal is called the skew time. and the bus protocol must include a sufficient time delay to account for the maximum expected skew time of the bus. When the master has allowed time for propagation delay and skew, it then asserts the request line (time t_1), asking the addressed slave to provide the information. The addressed slave module performs whatever action is required to obtain the data: if it is a memory device this will require a memory cycle. but if it is an interface module the information may be readily available. When the data has been obtained. the slave module asserts the data onto the data bus. as well as asserting the acknowledge line (time t_2). At this point the master must wait for a period of time to allow for skew, then it accepts the data and releases the request line (time t_3). When the slave detects the release of the request line. it releases the acknowledge line. Some time after the release of the request line, the master is free to release the address.

This basic asynchronous communication protocol is used by a number of different microprocessorsand minicomputers. It has the advantage of not needing a specific clock, since the transaction proceeds according to the signals asserted by each module. Since the modules can proceed as fast as their functions allow, the transactions can proceed as fast as data is available. The drawback is that the built-in delays, needed to allow for signal skew and propagation delay, force a relatively long minimum cycle time. For the *UNIBUS,* which is the bus on which the Digital Equipment Corporation PDP11 series is based, a typical minimum cycle time is 400 ns. Nevertheless, because of its simplicity and ease of function, the asynchmnous bus protocol is used extensively. One example is the Multibus, which originated with some products from Intel.

Example 6.1: Asynchronous protocol: The Multibus is an asynchronous protocol that fits the discussion above. What are the signal and control lines utilized by the Multibus, and the associated delays?

The asynchronous protocol, as described in the above paragraphs and in Figure 6.3, is directly applicable to the Multibus, with a few modifications in nomenclature. The signals on the Multibus are all asserted low, so the address, data. and control lines have a low voltage for a "1" and a high voltage for a "0." Then are 20 address lines and 16 data lines, which gives an addressable space of one megabyte. The address lines are used for both I/O and memory addresses. After a master has asserted the address, it waits for 50 nsec before,asserting the request line; this is the time allowed for skew and delay. The appropriate request line is asserted low (as opposed to the high assertion shown in Figure 6.3). Instead of having a read line to identify the direction of the transfer, the Multibus has separate request lines for memory read (MRDC-L), memory write (MWTC-L), I/O read (IORC-L), and I/O write (IOWC-L). This allows the address lines to be used by memory and I/O devices, and the appropriate interface module will respond only when the necessary control line is asserted. When a slave module responds, regardless of the request line that activated the module, it will assert a transfer acknowledge signal (XACK-L), in the manner shown in Figure 6.3.

Example 6.2: Interface to asynchronous system: Assume that a floating point multiplier is to be interfaced to the Multibus in the I/O space. This multiplier requires two 32-bit words to be available, one in Register X and one in Register Y. Design an interface module for the Multibus that will read and write to Register X and Register Y, and also cause the multiply to occur when accessed. Assume that the multiply process will take a variable amount of time depending on the data. and that the multiplier will assert a DONE signal when the answer is available.

The Multibus protocol allows 16-bit bus masters to address 4.0% different I/O locations, so we will assume that the floating point multiplier in question is to occupy the following addresses:

Address	*Request Line*	*Action*
$DF0_{16}$	IOWC-L	Write to Register X (low 16 bits).
	IORC-L	Read from Register X (low 16 bits).
$DF1_{16}$	IOWC-L	Write to Register X (high 16 bits).
$DF1_{16}$	IORC-L	Read from Register X (high 16 bits).
$DF2_{16}$	IOWC-L	Write to Register Y (low 16 bits).
$DF2_{16}$	IORC-L	Read from Register Y (low 16 bits).
$DF3_{16}$	IOWC-L	Write to Register Y (high 16 bits).
$DF3_{16}$	IORC-L	Read from Register Y (high 16 bits).
$DF4_{16}$	IORC-L	Read from Result (low 16 bits).
$DF5_{16}$	IORC-L	Read from Result (high 16 bits).

The design of this system is relatively straightforward, since the logic is basically combinational in nature. The only timing requirements are those imposed by the bus protocol, and the sequentiality of action defined by the protocol is also enforced by the master. The data path for this interface module is shown in Figure 6.4(a). The registers are made up of positive edge triggered devices ('273s), which hold the information for the floating point multiplier. Note that for this system an inverting bus transceiver has been inserted into the data path. This has the benefit of presenting only one electrical load to the data bus, but incurs the penalty of an additional delay, which needs to be included in the design process. Many multipliers have registers built in, so in one sense the external registers are redundant. However, the specification indicates that these values should also be made available to the bus upon request, so the registers are needed to provide that capability. Tri-state drivers ('541s) are used to send the information to the internal data bus, which is enabled onto the Multibus data lines by the transceiver. This path is also used by the product from the multiplier.

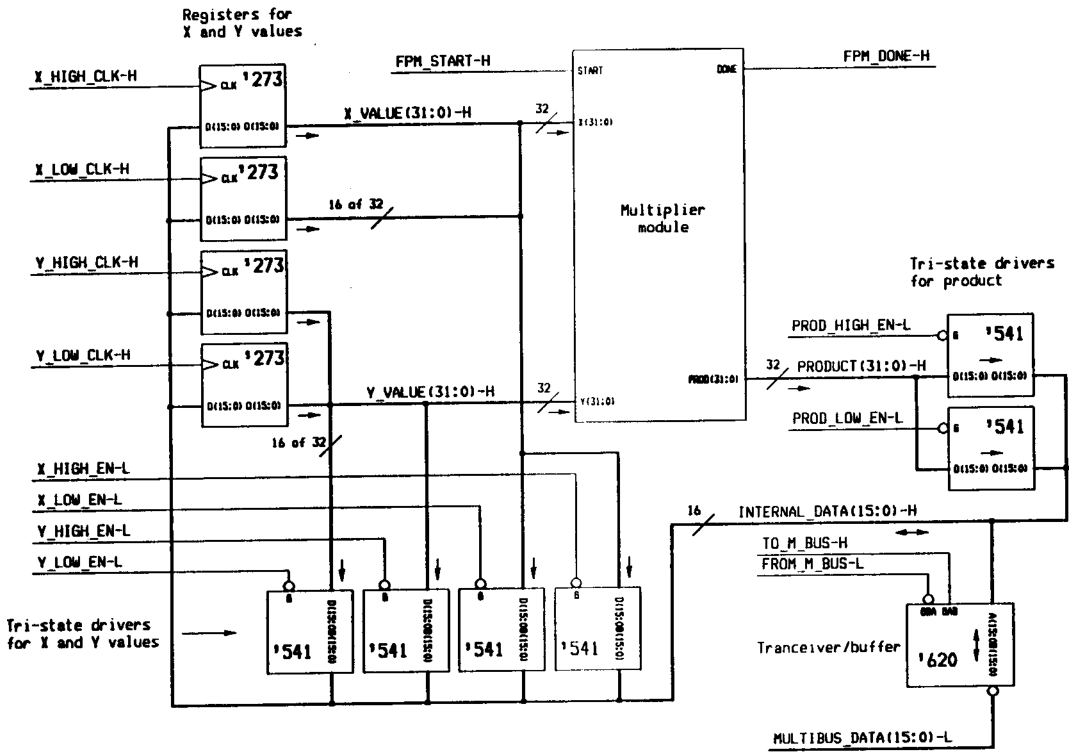


Figure 6.4(a). Data Path for Multibus Interface Module of Example 6.2.

The control signals used for this system arc derived by the logic shown in Figure 6.4(b). The address lines arc checked for a proper address pattern. However, since the address pattern could be asserted for memory addresses as well as the I/O addresses needed for this system, no action is

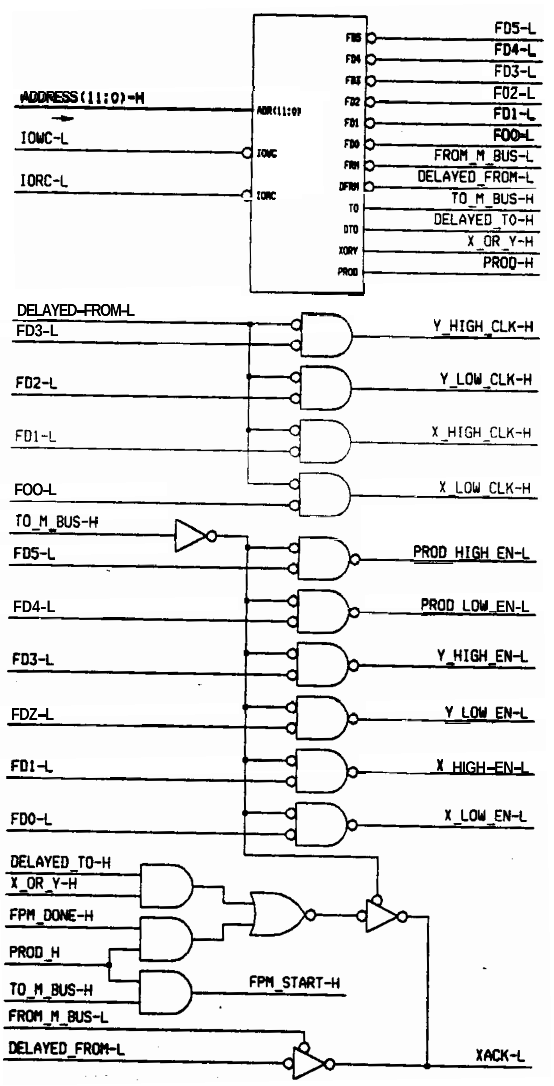


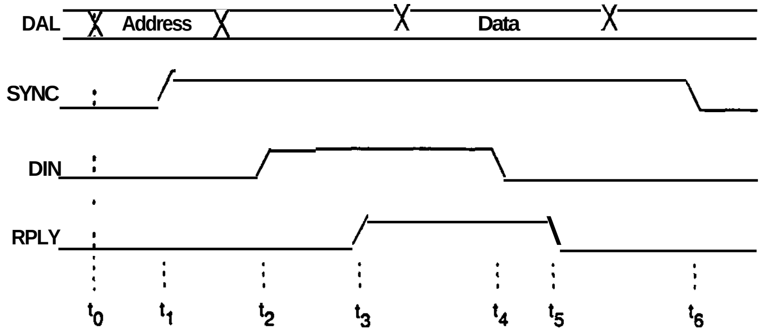
Figure 6.4(b). Control Signals for Multibus Interface Module of Example 6.2.

taken until the I/O request lines are asserted. If the transaction is a write to the X or Y register, then the Multibus data lines are enabled onto the internal data bus (with FROM_M_BUS-L), and after a delay to allow the data to propagate to the registers, the appropriate clock line is asserted. Figure 6.4(b) does not indicate how this delay is obtained, but a number of different methods could be utilized, from a tuned delay line to a synchronous method using the clock provided on the Multibus. The slave response to the I/O request lines is through the acknowledge (XACK-L), which is seized when the address is recognized, but not asserted until the transaction is complete. For filling the X and Y registers the acknowledge will be asserted when the delay has been completed. Similarly, reading the X or Y registers, or the lower bits of the product, involves a delay to allow the data to propagate onto the internal data bus and then to the Multibus data lines. When a propagation delay time has been accounted for, then the acknowledge can be asserted. Requesting the higher bits of the product causes a multiply to occur, so the acknowledge is asserted when the done signal is asserted by the multiplier. This necessitates that the most significant word of the product be requested first to achieve proper results.

The Multibus, and many other buses that use the asynchronous handshaking technique to transfer information, can be effectively utilized to pass data in a single bus environment. However, the lines required to perform this type of transfer are rather numerous. The Multibus utilizes 41 lines to perform these transfers, and the UNIBUS uses 38 lines. One of the ways to reduce the number of wires required is to time multiplex the address and data lines. That is, one set of lines contains the address for part of the time and data for another part; the information content of the lines is determined by the control signals. Thus, the total number of wires required to perform transfers is reduced. The tradeoff is between the number of wires on the bus (or pins on the integrated circuit, or on the edge of the board, or ...) and the increased time required to perform the transfer. Since the lines are utilized for two functions (address and data), then the number of control lines will increase. Nevertheless, the total number of wires is decreased, and the speed of the bus is sufficient for many applications.

Example 6.3: Time multiplexed asynchronous protocol: Digital Equipment Corporation has built a number of devices based on a protocol and physical configuration called the Q-Bus. This is a time multiplexed data/address bus with an asynchronous protocol. What is the sequence of events involved in performing a read and a write with the Q-Bus?

The waveforms for the transfers of the Q-Bus are shown in Figure 6.5. This is an abbreviated version, since there are control lines to indicate when a transfer is in the I/O page, when it is a byte transfer, and so on. But the basic principles are demonstrated by the figure. The levels indicated in the figure are logical levels only, since the assertion level of the signals on the bus itself is negative, so that on the bus a "1" is indicated by a low voltage level. The read cycle [Figure 6.5(a)] begins with the master asserting the address on the time multiplexed data/address lines (DAL), then allowing a time for propagation delay and skew. The SYNC line is then asserted (time t_1), which is used by slave devices to latch the address information as needed. The master releases the DAL lines, and at time t_2 asserts DIN, which indicates to the addressed slave that the transaction is a read. From this point the transaction follows the four event sequence, with DIN



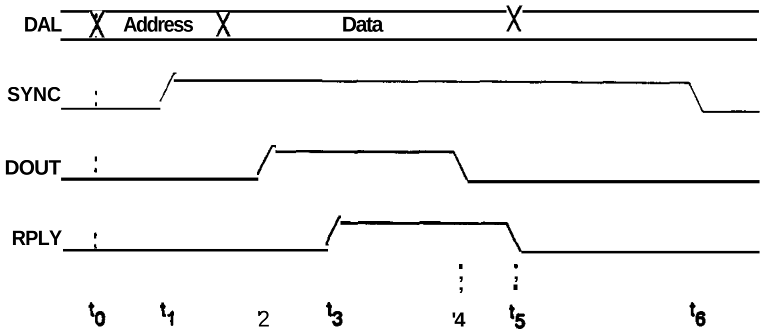


Figure 6.5. Read and Write Cycles on a Time Multiplexed **Bus:** (a) Read Cycle; (b) **Write** Cycle.

representing the request line, **t_3),** and then within 125 nsec **asserts** the **data.** The master **responds** by releasing DIN (time t_4). When the slave detects ***the* release** of **DIN,** it **releases** RPLY (lime **t_5),** and then releases the DAL lines. The last event in the cycle is the **release** of the SYNC signal by the master in preparation for the next cycle.

The write cycle shown in Figure 6.5(b) is **very** similar **to the** above **sequence** of **events. The** major difference is the **assertion** of the **data** *on the* **DAL lines** by **the master after the address has been issued and synchronized by** the SYNC signal. *Once* again. *the* four event cycle mechanism is used. **The** master identifies the cycle **as** a write cycle **by asserting DOUT. The slave accepts** *the* **data and asserts** RPLY. The **master then releases DOUT.** which **allows** the slave **to release** RPLY. The bus **protocol calls fa** *the* master **to hold** the **data on** the DAL hes for **at least** 175 nsec **after releasing DOUT. And as before, the termination** of the cycle is indicated by releasing SYNC

The asynchronous method for information transfer can be very useful for exchanging data in time multiplexed systems and in systems with dedicated address and data lines. It is simple to comprehend, and interface modules between the bus and external devices can be designed and constructed in a relatively easy manner. The absence of a clock allows the transaction to proceed at the rate at which data (and address) information is available. Nevertheless, the data rates for this type of transfer are in general not as high as those for a synchronous protocol. Before we discuss the reasons for this, let's examine some of the arbitration mechanisms used to identify the module that will control the bus transaction.

6.2. Arbitration Mechanisms

In any system with multiple master modules. that is, modules that can assert the control lines on the bus, a mechanism must be provided for arbitration. Using some predefined priority algorithm, this mechanism must uniquely identify the module that will take charge of the bus for the next transfer. It is possible to have this decision follow each bus cycle, so that there is an arbitration between each bus transfer. But in general the arbitration process is performed in parallel with data transfers. so that during the current transfer arbitration is being performed for the next transfer. In this section we will consider arbitration mechanisms and how they can be utilized to assure that control is passed to the proper module.

Three basic mechanisms can be utilized for making the decision as to the proper module to control the bus for the next cycle. These are shown in Figure 6.6. In each case. the masters (M_1, M_2, ...) request access to the bus by asserting a bus request (BR). When the arbitration mechanism is ready to select a new master module to control the bus, it will assert the bus grant signal (BG) associated with that module. The behavior of the devices receiving the bus grant depends on the type of arbitration mechanism involved, as we shall see. When a device needs access to the bus and it detects that the bus grant line has been asserted, then it will be the next to receive control of the bus. If more than one master requests ownership of the bus at the same time, then the arbitration process selects one. and the remaining modules must wait until a later time for their respective transfers.

The fastest arbitration mechanism is the parallel system. In this system each master module has a dedicated connection to the arbitration unit, and when a master module needs control of the bus it will assert its assigned bus request line. The arbitration unit then has the responsibility of dealing with the system in some predetermined fashion. That is, the algorithm utilized in the design of the arbitration unit is not limited by the interconnection system. The arbitration can be done on the basis of first-asserted/first-served, round robin, assigned priority levels, or whatever mechanism is determined in the design process. Thus, this mechanism allows a variety of possibilities, from extremely simple to extremely complex.

In the parallel scheme, when the arbitration unit has determined that a master module has priority and should have control of the bus, it asserts the bus grant line associated with that master module. This module can then control the transfers on the bus. The data, address, and handshake lines are controlled by the selected master, and when the master no longer requires access to the bus, it will release the bus request signal. The parallel arbitration system is then free to allow other master modules to gain access to the bus.

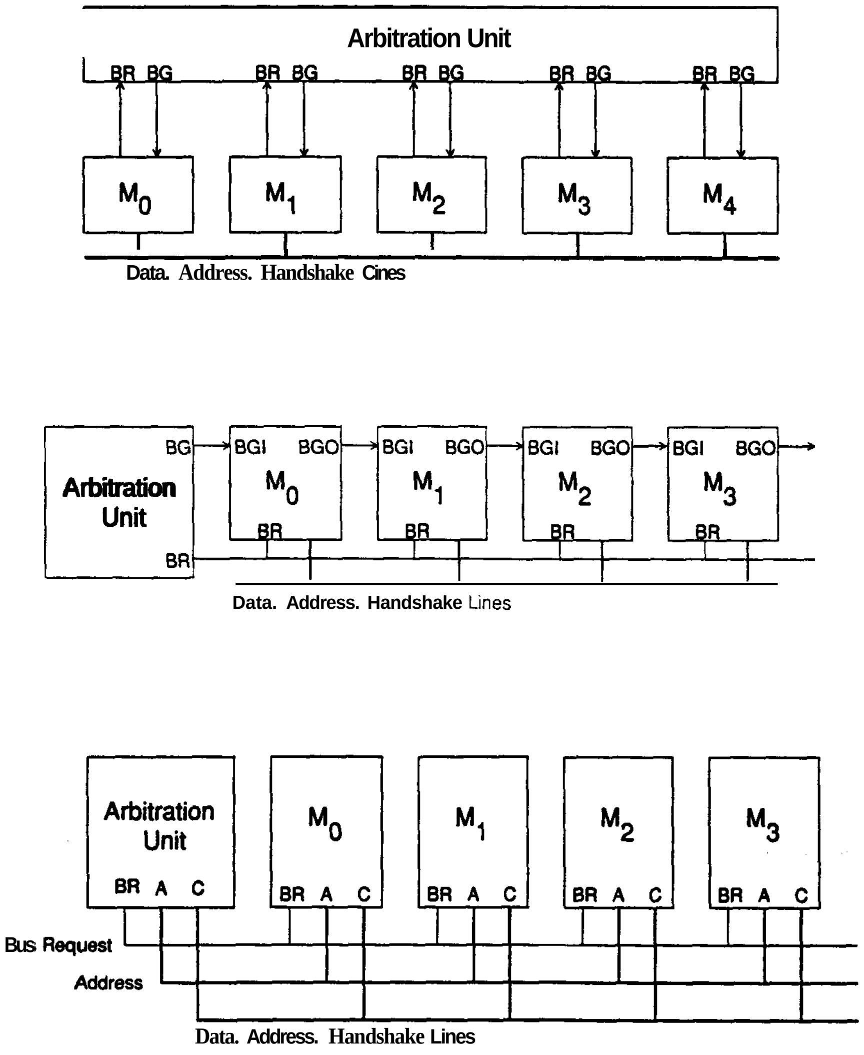

Arbitration Unit
BR BG
M0
M1
M2
M3
M4
Data. Address. Handshake Lines
BG
BGI BGO
BR
Data. Address. Handshake Lines
Arbitration Unit
BR A C
Bus Request
Address
Data. Address. Handshake Lines

Figure 6.6. Bus Arbitration Mechanism.

The details of the transfer mechanism will vary with each implementation. but the parallel mechanism provides the highest speed of arbitration. The cost for this speed is the additional lines required to allow each possible master direct access to the arbitration unit, and the hardware costs associated with whatever arbitration algorithm is implemented. The number of lines required could be extensive, needing two lines for each module as shown in the following example.

Another mechanism would need only one line per module, as we will see later.

Example 6.4: Parallel arbitration system: Design a parallel arbitration system that will allow up to eight bus masters to access a common set of control lines. The assumed mechanism for master-slave data exchange is the four event handshake that has been discussed. If no bus master has control of the bus, then the requests are to be synchronized by an internal 10 MHz clock. If a master module has control of the common handshake lines, then the requests are synchronized on the trailing edge of REQ-H.

This type of a system can be easily constructed with a priority encoder and a decoder, such as shown in Figure 6.7. Notice that the assertion levels are low in this example. When no request is pending for the bus (no bus master requires use of the bus) the decoder is disabled, and no master has control. The requests for access are synchronized by a 10 MHz clock, and when one of the masters has made a request for the bus, the appropriate bus grant line will be asserted. If more than one module has requested the bus. then only the highest priority bus grant line will be asserted. Note that the nature of the '148 priority encoder, with its asserted low outputs, inverts the normal order on the decoder outputs.

The desired behavior, as defined above, is that the synchronization of requests take place on the trailing edge of REQ when the bus is being used by a bus master. The gates on the input of the clock of the synchronizing register multiplex between the 10 MHz clock and the bus request to allow this to happen. This simple mechanism is. in general. not sufficient. since it does not preclude the possibility of glitches occurring on the clock line. (What additional gating is required to assure that no glitches occur'?)

The example demonstrates the simplicity with which parallel systems can be constructed. However, more exotic priority algorithms. such as first-asserted first-serviced. will lead to more complex implementations. But because of the speed with which arbitration can proceed in this case. systems that need the performance will provide the lines necessary to allow parallel arbitration. Because of

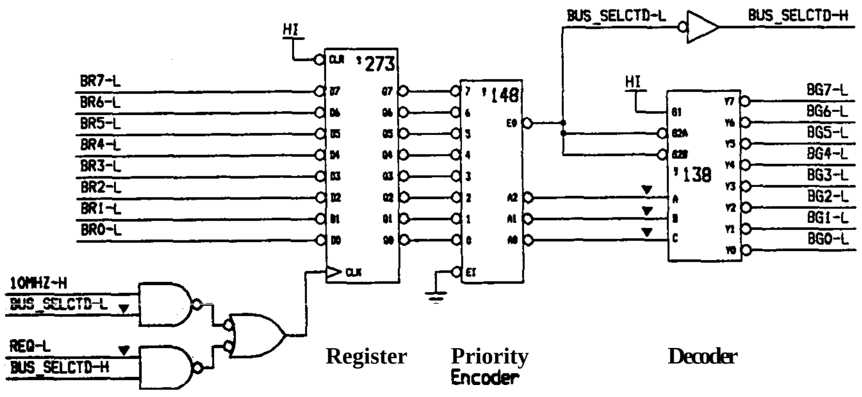


Figure 6.7 Simple Parallel Arbitration System.

the need to have dedicated lines to the master modules for parallel arbitration, the number of allowable masters on any system is fixed at the time of implementation. This places a fixed limit on the number of allowable masters. and the system cannot be expanded beyond that limit in a parallel fashion. Expansion is one of the benefits of the next type of system to be considered, the serial arbitration system.

Serial arbitration is a technique in which the bus grant lines of the bus masters are connected together in a serial fashion, as shown in Figure 6.6(b). There is a single bus request line, which is connected to all bus masters. The arbitration unit is not aware of which bus master needs access to the bus, and so the arbitration mechanism is simplified to asserting the bus grant signal at the proper time in the bus cycle. The arbitration unit is then responsible for examining the lines controlling the transfers on the bus and deciding when control of the bus can be given to a new master module. When the bus can be controlled by a new module, the arbitration unit asserts a single bus grant line connected to the first module. Since this module is the first to receive the bus grant signal, it has the highest priority: a device can receive the bus grant signal only if the modules between it and the arbitration unit do not need the bus. Because of this connection method, where one module passes the signal on to another in a serial fashion, this is referred to a "daisy chain" mechanism. And because of its serial nature, there is no limit to the number of devices that can be connected in this manner. However. each additional device results in a longer maximum arbitration time.

The serial mechanism for bus arbitration needs at least three lines to function. although more can he used. as indicated by the example below. The three lines are hus request. bus grant in. and hus grant out. A master module indicates that it needs to access the bus by asserting a common request line, as shown in the figure. This line is implemented in open collector technology, or some other method that will allow multiple units to assert the signal simultaneously. The arbitration unit uses this signal to identify when a new bus master needs access to the bus, as described above. When the arbitration unit determines that a different module can control the bus, it asserts the bus grant line. Each master receives the grant signal on its bus grant in line, and if the module does not need to access the bus, it asserts the bus grant out line. In this way the assertion of the bus grant signal is passed from one module to another, until it arrives at a module which needs access to the bus. This module does not assert the bus grant out line, but rather assumes ownership of the bus and performs the needed transfer. A master module of lower priority that needs access to the bus will continue to assert the request line. and at a later time a new bus grant signal will be asserted by the arbitration unit and passed to it.

The priority scheme of this system is strictly physical: devices of higher priority are physically (and hence electrically) closer to the arbitration unit. Devices of lower priority are farther away from the arbitration unit. The number of devices included has a direct effect on the speed of the function. Since each device must check the bus grant signal in a serial fashion, the total time for the arbitration function is proportional to the number of devices on the bus. Of course, the closer the device is to the arbitration unit (fewer modules in between), the faster the operation. But since each module requires time to complete the bus grant in to bus grant out sequence, there is a practical limit to the number of devices that can be utilized.

Because of the serial nature of the arbitration process, care must be taken to avoid the situation where two masters access the bus simultaneously. This possibility will arise in systems in which the modules operate asynchronously with respect to each other and to the transactions taking place on the bus. In this case, a module could quire access to the bus directly after the bus grant out signal had been asserted to inform the next module in the chain that it can access the bus. If the first module is allowed to immediately command the bus and release the bus grant out line, then both units could be in a situation where they are. accessing the bus. A practical solution to this problem is to design the units to be edge sensitive rather than level sensitive. That is, the master modules would be capable of taking ownership of the bus only when the bus grant signal is changed from its unasserted to its asserted level. Thereafter, the unit must wait until the next assertion of the signal, even though it is currently asserted. This mechanism will prevent more than one module from assuming control simultaneously.

Example 6.5: Serial arbitration system: The UNIBUS uses serial arbitration to identify bus master modules that need access to the bus. What are the lines involved in this arbitration process. and how does the protocol function? Also, what circuitry is need to connect to the arbitration lines to properly utilize the serial arbitration lines?

A number of lines in the UNIBUS are used by the master modules to control access to the bus. For the purposes of understanding the mechanism, we need consider only four signals: BR-L (bus request, asserted low), BG-H (bus grant, asserted high), SACK-L (selection acknowledge. asserted low), and BBSY-L (bus busy. asserted low). These lines and the relationship between them are shown in Figure 6.8. The sequence of events begins at t_A, when the bus arbitration unit recognizes that a new arbitration cycle can begin. since SACK is nor asserted. When a master module needs to transfer information over the bus, it will signal the arbitration unit by asserting the BR line (t_B). The arbitration unit will then respond by asserting BG (t_C). Some time later (t_D), the bus grant signal will be received at the master module; there may be other master modules through which this signal has passed to reach the module that requested the transaction. When the bus grant signal is received by the module needing the bus, it will *not* pass the signal *on,* and it will assert SACK. This signals the arbitration unit that the arbitration process was successful, and it can now release the bus grant signal. At the same time. BR will be released by the module, but this will not necessarily mean that the line will return to its unasserted level, since another master module may also be asserting the request line. When the arbitration unit receives the assertion of SACK, it releases BG (t_E). The actual arbitration process is now complete, but the bus is still being used by a different module. When the current bus master completes its cycle, it will release BBSY (t_F), signaling the next bus master that it has completed its operation. The new bus master will wait for SSYN (not shown) to be released, indicating that the slave involved in the last transfer is idle, and BG to be released. At that time it will be able to control the transactions on the bus. The new bus master will then assert BBSY to signal the fact that it is controlling the bus, and relase SACK, to allow the arbitration process to select a new bus master.

A logic diagram of an system that does this is shown in Figure 6.9. The gates receiving bus signals (RCV) and drive bus lines (DRV) have special electrical characteristics that minimize the electrical loads placed on the bus. Otherwise, the gates have the normal NAND or NOR function shown by the shape of the gate.

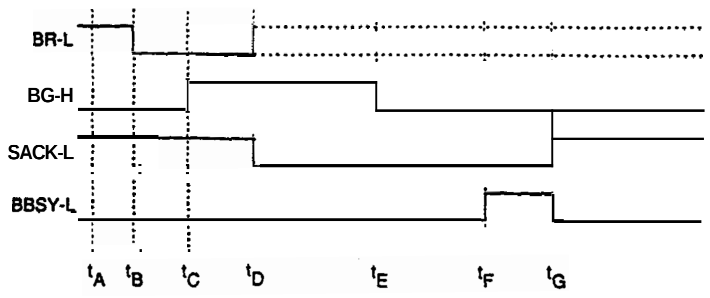


Figure 6.8. UNIBUS Bus Arbitration Lines.

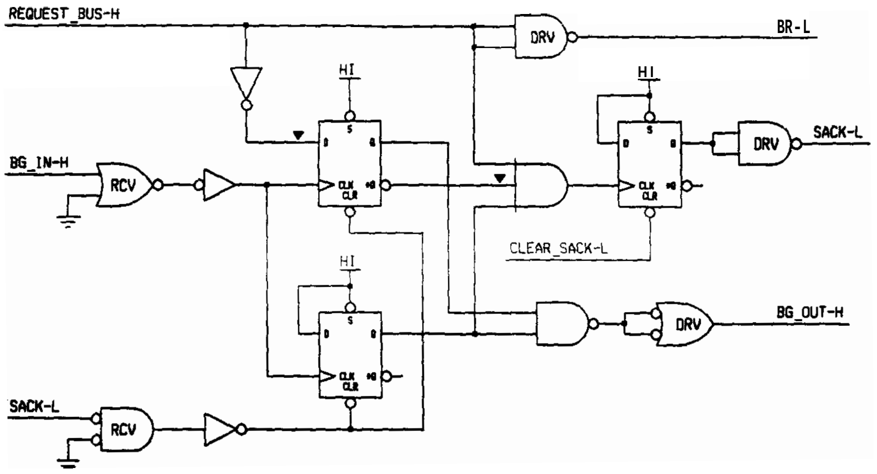


Figure 6.9. Logic for UNIBUS Bus Request-Bus Grant

The UNIBUS protocol was chosen for this example for three reasons. First a great number of devices have been built to interface with the UNIBUS. and so for sheer numbers this is a very prolific mechanism. Second, this example demonstrates that the arbitration process can proceed in parallel with the transfer currently in progress. Many asynchronous buses require that the current transaction terminate before arbitrating for ownership of the bus. And third, the mechanism described here is utilized in one form or another by almost all asynchronous bus arbitration systems.

The protocol described in Example 6.5 is similar to many schemes that use the daisy chain method of arbitration. One of the problems that can arise with this mechanism is the transfer of control from one master to another. Although the arbitration system can select a bus master to assume control of the bus, the actual transfer of control will not occur until the current bus master releases the

BBSY line. Therefore. a bus master may control the bus for an extended period of time, not allowing other modules access for transfers. In that sense, the **protocol** is not "fair," and may not be applicable in some circumstances. To prevent this type of device lockout, schemes can force the system to arbitrate for every transfer, **instead** of arbitrating for ownership. Or a a mechanism may be included that will force a module to relinquish ownership of the bus and allow the arbitration process to find a new bus master.

The use of one kind of bus arbitration does not exclude the use of another. The **UNIBUS** uses parallel arbitration in combination with serial arbitration, as does the VME bus. Parallel arbitration occurs in the **UNIBUS** because there are five sets of **BR-BG** lines, each of which has a different priority. The access to the bus between these five sets is done in a parallel fashion. Each of the five sets of **BR-BG** lines is a serial line, and operates as described in the example above.

The final bus arbitration technique we are going to mention is polling, which is shown in Figure 6.6(c). Here each master module has access to a common request line, which it will assert when it requires access to the common resources. The arbitration unit must then decide which of all of the possible modules made the request. It does this by placing the address of a master module on the address lines and querying each in turn, until it finds the highest priority module needing the bus. This method has the benefit that any priority scheme can be implemented — FIFO, round robin, and so on. But the cost of the mechanism is large in time requirements. For that reason it is almost never used for arbitration of bus lines. but it does find application in the arbitration of I/O requests. That is. a processor. under program control. will poll I/O devices to ascertain the module requesting an Interrupt.

6.3. Synchronous Bus Protocols

The term "synchronous bus" can refer to a number of different techniques for **transferring** information between modules. The common characteristic of all of these mechanisms is that a clocking signal is used to synchronize all of the transfers. This **restricts** the length of the bus, since the signal must propagate to all bus masters and bus slaves, and be received with a reasonable degree of simultaneity at all locations. In this section we will consider some of the mechanisms that can be used for synchronous data transfers on bus systems.

One type of a synchronous bus is not a multiple master, general purpose bus. This is a bus bus system under the direct control of a central unit. This type of system fits into the model shown in Figure 6.1, but each of the units is directly connected to a master control unit. This central control unit then decides which module is to assert information onto the bus, and which element is to accept the information. That is, no general address is decoded by slave modules, but rather the central control unit selects both the source and the destination. The **microprogrammed** modules studied in Chapter 5 are included in this classification. since the contents of the bus are determined by the microcode word during each **micro** cycle.

Another bus protocol that is synchronous in some aspects is typified by the bus **connections of** some high performance **microprocessors. The M68020** has a protocol almost identical to that described above, except that the **mechanisms** allow for dynamic bus sizing and other flexibility. The mode of operation is **synchronous** with the system clock, giving the appearance of a synchronous mechanism. If the slave (memory, for example) is not able to respond to the processor fast enough to allow continuous operation, the processor automatically inserts idle bus periods, called "wait states," until the slave responds with the desired data Thus, the only difference in method is that the M68020 works in increments of the basic system clock, rather than using completely asynchronous signals.

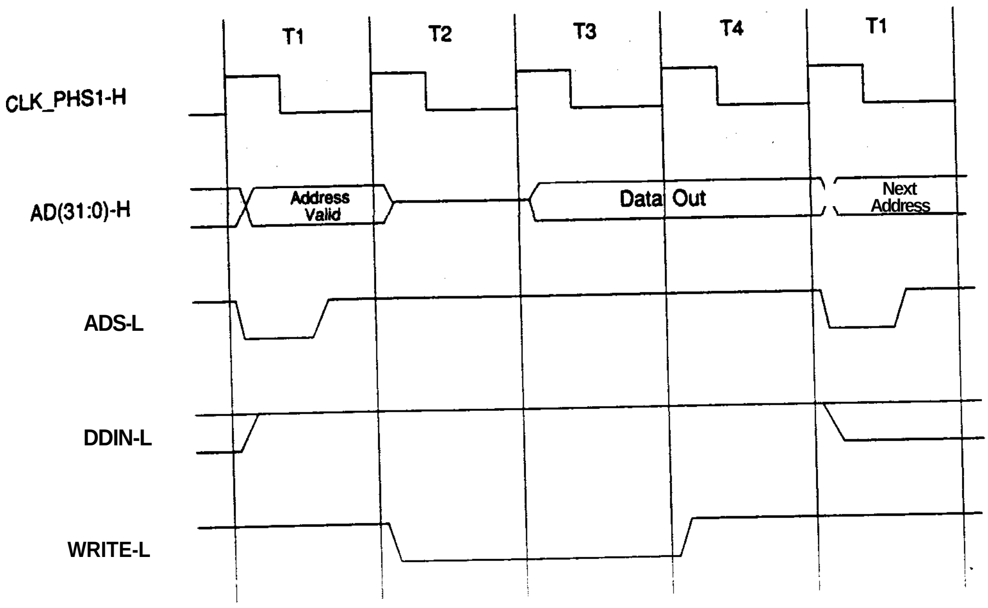
Another bus protocol is used by a number of micmprocessors, and works in conjunction with the system clock. One of the problems that has become prevalent as integrated circuits have increased in complexity is providing enough pins to transfer the information into and out of a device. To minimize the total number of pins required for information transfer. some devices time multiplex the bus lines to allow one set of pins to present both address and data information. Thus, a processor with a 32-bit data path and a 32-bit address requirement can use one set of 32 pins, and synchronize all requests in such a way that all bus modules know when the address is available, and when the data is required.

A sample of the NS32332 protocol is presented in Figure 6.10(a), which presents a write cycle. The 32 bits of address and data share the time multiplexed AD(31:0)-H lines: the presence of a valid address is identified by ADS-L, and the data is synchronized by WRITE-L. The DDIN-L line identifies the direction of data transfer. The minimal transaction requires four cycles; the address is presented in the first cycle, and the data is available during later cycles. If the slave cannot respond within the required time, the master can wait until the transaction is able to proceed. This may occur. for example. if a dynamic memory is performing a refresh cycle when the processor requests a transaction. Most systems that use this technique will latch the address and create the appearance of separate address and data buses. A block diagram of one such arrangement is shown in Figure 6.10(b). To a slave device attached to the separate address and data lines, this communication mechanism appears the same as those previously described: the four event transaction proceeds in exactly the same way.

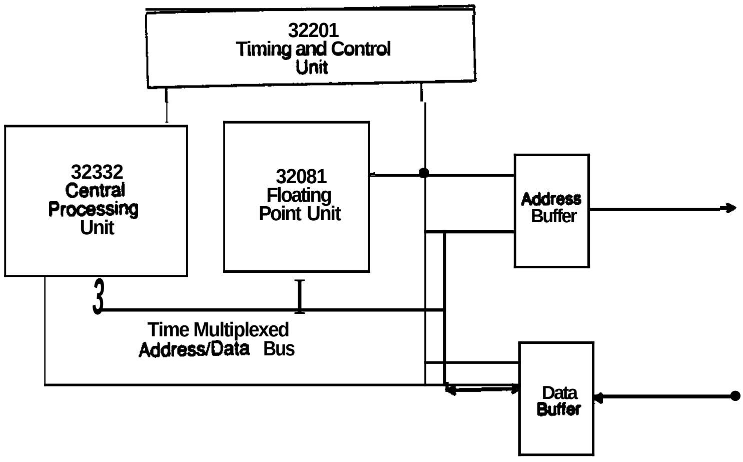
The time multiplexed data/address lines provides a mechanism to efficiently utilize one scarce system resource, the number of pins on the device. But another system resource that is not effectively utilized in the protocols described above is time. The master must alert the slave that some information is needed, and then wait for the slave to respond. A more time efficient mechanism would be to identify the basic components of a transfer and so design the protocol and the bus to allow these components to occur simultaneously. This requires a greater complexity on the part of both the master and slave modules, but it does more efficiently utilize the wires used to connect the modules together.

One of the beneficial features of asynchronous protocols identified in Section 6.1 is that the transaction proceeds as fast, or slow, as both sender and receiver agree that the information can be transferred. If some event requires more time, then the protocol essentially waits for the event to complete, and then proceeds with the transfer. This provides for increased flexibility, and it also provides for fairly simple interface modules. However, the overall data rates will be higher if more capability is provided in both the sender and receiver to minimize the amount of time that the bus lines are utilized to exchange the information. This is the basic premise of synchronous protocols, and the mechanism provides for time efficient use of the bus lines.

In the protocol described in Section 6.1. the bus master was responsible for asserting the address, and then allowing time for propagation delays and signal skew before asserting the request line to initiate action. One of the reasons that a



(a)


32201 Timing and Control Unit
32332 Central Processing Unit
32081 Floating Point Unit
Address Buffer
Data Buffer
Time Multiplexed Address/Data Bus

(b)

Figure 6.10. **NS32332 Bus Transactions:** (a) **Timing Diagram** for a Write **Operation;** **(b)** Block **Diagram** of **Interface** Logic.

synchronous protocol is more time efficient than the asynchronous protocol is that the action of all of the modules is coordinated by the presence of a common clock. This establishes an exact time when the information must be present on the bus, and when each module attached to the bus will know that information is available. This establishes bounds on the time required to transfer the information, and interface modules must all be designed to operate within those bounds. Thus, this mechanism calls for the interface modules to meet a time standard, rather than having the protocol adjust the time requirements to satisfy the needs of the various interface modules. The modules connecting to the bus must then be capable of transferring information at the rate determined by the bus protocol.

The mechanism of data exchange for synchronous protocols operates on a different set of principles than the asynchronous methods previously described, and this leads to a slightly different nomenclature when dealing with the units. We will call the module that initiates a transaction a commander, because it sends a command to another module. The command may or may not contain data, depending on the type of transfer. The module that fields the command we will call a responder. since it responds to the request in an appropriate manner. As with the asynchronous protocols. a number of different mechanisms will function properly. We will describe first a sample mechanism for write and read. and then examine a specific instantiation of a protocol.

There are four components of the transfer of information, and all four must be completed for a successful transfer. We discussed each of these functions in the process of describing the asynchronous protocols and arbitration mechanisms, but did not identify them as necessary constituent parts of the transfers. These components are:

1. Obrain control of the bus. This is the responsibility of the commander and the arbitration network. When a module requires a transfer, it communicates that need to the bus interface module, which initiates a request for the bus. When the arbitration process allows the commander interface module access to the bus, the transfer can proceed.
2. Initiate transfer. If this is a write, this will include data. The commander places appropriate address (and data, if needed) and control information on the bus. The responder with that address will react by accepting the request This does not imply that the responder module will be able to handle the request, only that the request has been received.
3. Decide how to handle the request. This is the task of the responder bus interface module. This does not mean that the subsystem attached to the bus will necessarily accept (or provide) information immediately, but the bus interface module of the responder must be capable of deciding how to respond to the request. For example, if a memory is ready to accept information, it will be capable of accepting the information, and the bus interface module will decide that the information can be accepted. On the other hand, if a memory is busy with a previous request and unable to accept data, the bus interface module will decide to reject the request.
4. Inform commander of the decision of the responder. This is the feedback mechanism to allow the handshake to occur, and indicates to the commander that the request has been handled. If the request was a write, for example, the system attached to the commander bus interface module can proceed with its tasks. However, if the system attached to the responder interface module was

unable to accept the data. this decision is relayed to the commander, and the commander interface module can then initiate the request anew.

These four components are present in the asynchronous protocol, with its associated arbitration mechanism, but are not as evident as in synchronous protocols. The arbitration component can be handled in parallel, as in the UNIBUS protocol, or after a bus is available, as with most microprocessor bus systems, such as the NS32032 systems. Component 2, initiating the transfer, is handled by the bus master in an asynchronous protocol; the master module asserts the address, waits the prescribed time. and alerts the slave modules by asserting the request line. The third component, deciding how to handle the request, is an integral pan of the slave module mechanism, since all requests in an asynchronous protocol are handled immediately. If a memory read is required, then the protocol awaits the response from the memory before proceeding. Thus, it is difficult to separate the act of responding from the decision to respond. However, in a synchronous protocol, these two elements are distinct. and are handled in a different manner. The decision process is handled by the bus interface module. while the response to the request is handled by the appropriate subsystem, such as a memory. The forth component, the handshaking mechanism, is handled by the request and acknowledge lines of the system.

These four events are shown in write and read sequences in Figure 6.1 1. The write sequence begins (period n) by the processor interface module arbitrating for use of the bus lines. When the arbitration process is settled in favor of the processor, the sequence proceeds, and the processor interface module asserts the data and address information onto the bus lines (period n+1). When the clock occurs, the memory interface module accepts data and address. and determines that the request was intended for the memory subsystem. During the next period (n+2), the memory interface module ascertains the status of the memory and determines that the data can be accepted. And finally, during the acknowledge period (n+3), the memory interface module sends an acknowledgement to the processor interface module to indicate that the transaction was successfully completed. Since the commander of the processor bus interface module started the series of events in period n. it will know that the response of the memory (accept or reject) will be found in period n+3, so it will listen to the lines at that time to find out if the write action was successful.

The read sequence is also shown in Figure 6.11. The transaction is initiated by the arbitration of the processor for the bus (period n). When the processor interface module has obtained control of the bus lines, it will then assert the address and request information on the bus (period n+1). Synchronous with the clock, the memory interface module accepts the request, and in the following period (n+2) ascertains the status of the memory and decides to accept the request. This decision is communicated to the processor interface module in the last period of this sequence (n+3). The memory subsystem is then activated in order to supply the required information. The time from period n to period m reflects the response time of the memory. When the memory provides the information, the memory interface module initiates a bus transaction, first by arbitrating for the bus (period m), and then by asserting the data onto the bus (period m+1). The processor interface module accepts the data synchronous with the clock, ascertains in the next period (m+2) that the data is in response to an earlier request, and in the last period (m+3) sends an acknowledgement to the memory interface module.

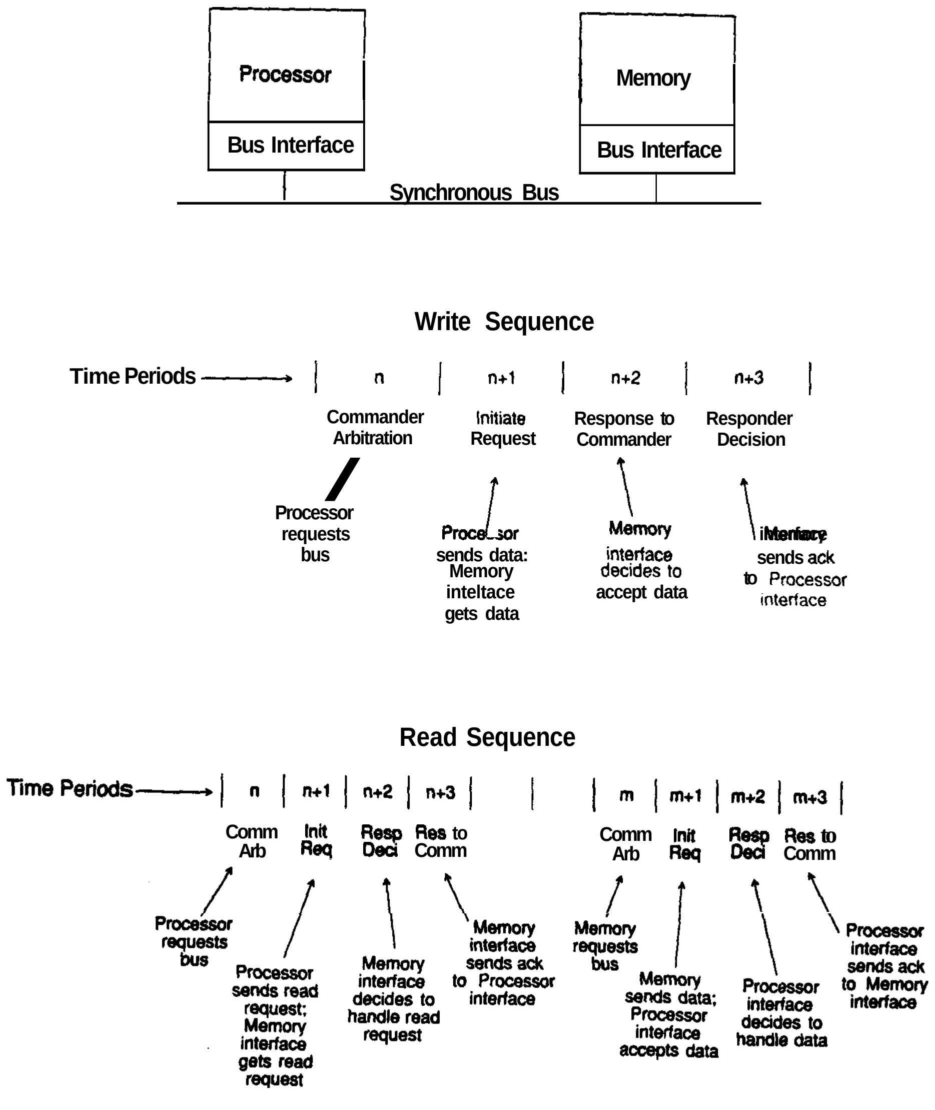

Processor
Bus Interface
Memory
Bus Interface
Synchronous Bus
Write Sequence
Time Periods
n
n+1
n+2
n+3
Commander Arbitration
Initiate Request
Response to Commander
Responder Decision
Processor requests bus
Processor sends data: Memory inteltace gets data
Memory interface decides to accept data
Memory interface sends ack to Processor interface
Read Sequence
Time Periods
n
n+1
n+2
n+3
m
m+1
m+2
m+3
Comm Arb
Init Req
Resp Deci
Res to Comm
Comm Arb
Init Req
Resp Deci
Res to Comm
Processor requests bus
Processor sends read request; Memory interface gets read request
Memory interface decides to handle read request
Memory interface sends ack to Processor interface
Memory requests bus
Memory sends data; Processor interface accepts data
Processor interface decides to handle data
Processor interface sends ack to Memory interface

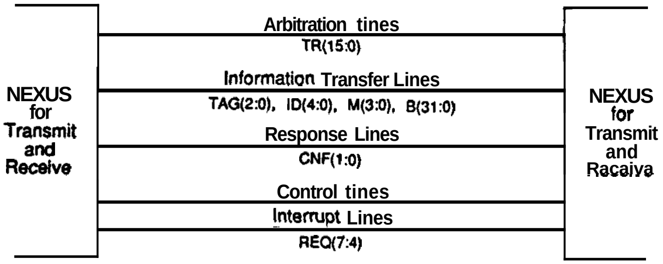
Figure 6.11 Synchronous Bus Mechanisms.

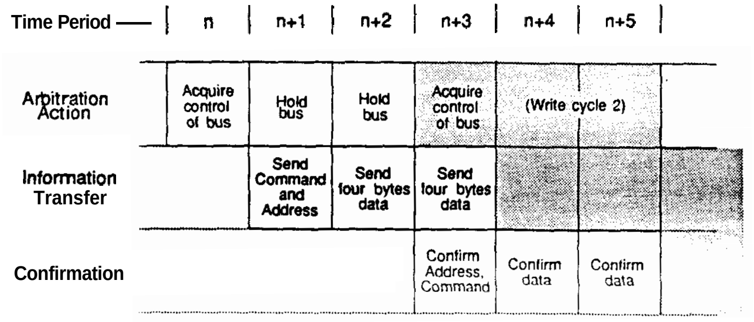
As indicated in Figure 6.11, each of the four components of the exchange happens in separate cycles of the common clock, and can be pipelined. We will discuss pipelining in more detail in Chapter 8, but the basic idea is that independent events can occur in different pieces of hardware in the same period. With multiple events occurring simultaneously, a speed advantage is obtained over the same events occurring serially. With a synchronous bus protocol, the interface devices can be designed in such a manner that each of the four functions involves a different set of hardware and a different set of bus lines, so that up to four separate transactions can be in different stages of execution at any one time. Thus, the speed advantage of synchronous bus transactions stems not only from the specific windows in which information must be valid, but from the pipelining and overlapping of transactions. Note that, if not enough transactions are available to keep the different portions of the bus busy during the various clock cycles. then the speed advantage of the pipelining is lost.

Example 6.6: Synchronous* bus *protocol: The synchronous backplane interconnect (SBI), which is the communication mechanism for the VAX 11/780 computer, is a synchronous bus protocol. What are the methods used by the protocol. and how fast can information be transferred on the bus?

The principle lines (but not all) involved in information transfer on the SBI are shown in Figure 6.12(a). The sixteen arbitration lines [TR(15:0)] allow parallel arbitration of up to 16 different modules during a clock period. The information transfer lines include the 32 data/address lines [B(31:0)] and lines for identifying the type of transaction that is occurring. The response lines [CNF(1:0)] provide a data path for confirmation of previous transactions. The principle difference between the SBI and the protocol discussed above is that the SBI time multiplexes the data/address lines so that a write will require more than one cycle. The SBI mechanism allows for one or two words of data in a write transfer, so that up to X hytes of information can be written. Such a write cycle is shown in Figure 6.12(b). To demonstrate the pipelined nature of the action, the transfer is shown in a space-time manner. The lines involved in the transfers are divided into three groups: arbitration lines, information lines, and acknowledge lines. And the action of these three sets is described for each of the cycles. The DEC name for the commander and responder interface modules is the NEXUS. The first period (n) is used by the arbination lines for the NEXUS associated with the processor to acquire control of the bus. Once this has occurred, the transfer can continue. The arbitration unit has the capability of locking out other requests for the two additional cycles needed to complete the transfer. The assertion of address and write identification information occurs in the second cycle (n+1). This information includes not only the target address of the write, but also an identifying field to specify the source of the information. The reason for this will become apparent with the read transaction. At the end of this period the NEXUS associated with the memory will receive the address and the identification information. The data/address lines are used in the next period (n+2) to send the first 4 bytes of data; at this same time, the memory NEXUS is deciding how to handle the request. At the end of the period, the acknowledgement decision has been reached, and the first bytes of data are accepted into the NEXUS. Then, during the final data cycle (n+3), the acknowledgment is retumed to the originating NEXUS for the address and write identification information. In the next two cycles additional acknowledgement information is returned for the data cycles of the transfer.

The pipelined nature of possible transactions is indicated to in Figure 6.12(b) by the shaded area that indicates a possible second write cycle to be


NEXUS for Transmit and Receive
Arbitration tines
TR(15:0)
Information Transfer Lines
TAG(2:0), ID(4:0), M(3:0), B(31:0)
Response Lines
CNF(1:0)
Control tines
Interrupt Lines
REQ(7:4)
NEXUS for Transmit and Racaiva



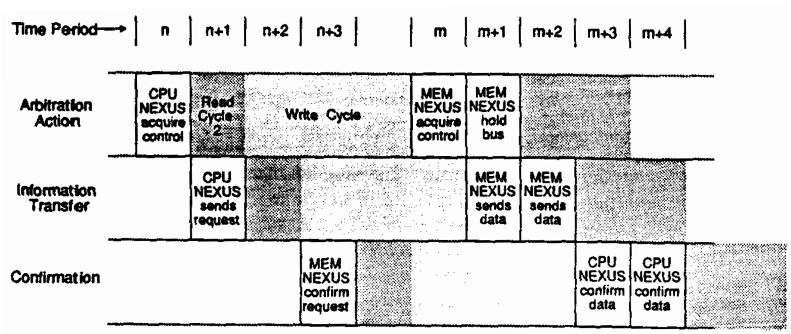


Figure 6.12. Synchronous Backplane Interconnect Protocol: (a) Control Lines Involved in SBI Information Transfer: (b) Write Transaction. 8 Bytes: (c) Read Transaction, 8 Bytes.

initiated by a *second* NEXUS. Note that the second cycle begins before the first cycle ends. This protocol allows meaningful data or address information to be placed on the data bus during each cycle.

The read transfer operates with a similar mechanism, except that the request and the response are separated by the response time of the memory.

This is shown in Figure 6.12(c). Here the NEXUS associated with the processor acquires control of the bus (period n), sends out a read request consisting of an address, a logical identifier, and a transfer type identification that informs the memory to supply 8 bytes. The processor receives the acknowledgement of the request in period n+3. Some time later, when the memory has the information for the processor, the NEXUS associated with the memory gains control of the bus (period m), and sends the data in two 4-byte transfers (period m+1, m+2). The destination of this information is carried by the identification lines, which will have the same logical identifier that was passed with the read request. The NEXUS associated with the processor sends its acknowledgement to the memory in periods m+3 and m+4.

Additional read and write transactions are shown in the shaded areas of Figure 6.12(c) to demonstrate the pipelining and parallel events possible with the protocol.

The clock cycle time for the SBI is 200 nsec. Thus, with the above protocol it is possible to send 8 bytes every 600 nsec. This gives an effective data rate of 13.3 MBytes/sec.

In this section we have considered some of the principles involved in transferring information with synchronous bus communication protocols. These mechanisms will, in general, lead to a higher data rate than their asynchronous counterparts for two basic reasons. First, the presence of a common clock limits the physical size of the system and synchronizes ail requests for action. This synchronization establishes a time at which all action must take place. Second. the separation of the components into independent pieces of hardware. and into independent bus lines, permits pipelining of the various functions. This allows concurrent use of the available resources. The net result is that data can be transferred at higher rates than achievable with other methods.

6A. Data Movement: Programmed I/O and Direct Memory Access

We have discussed some of the basic mechanisms involved in doing transfers of data over bus systems. Regardless of the exact protocol used, an arbitration mechanism is utilized to identify the module which controls the bus. This module then initiates a transfer, and the data is moved from one module to another. This mechanism is most often utilized to exchange information between a memory and a processor module. However, the same mechanism is used to transfer information and commands to and from I/O devices. In this section we want to explore some of the methods that can be used to control I/O devices and to transfer information to and from a computer system. For computer systems that include separate I/O instructions, generally an I/O bus is used for the communication. In some systems with I/O instructions, the system bus is used for memory and I/O transfers, but I/O transactions use a slightly different set of control lines to perform the transfers. However, one prevalent practice is to use the same address space for both memory and I/O devices. This method calls for the I/O devices to be assigned locations in the memory space, and then, when the device is to be activated and controlled, the processor does so by writing and reading the appropriate locations. This is called "memory mapped I/O," and is used extensively in minicomputer and micmprocessor systems. In fact. the inclusion of I/O instructions in the processor instruction set does not preclude the use of memory mapped I/O, and the manner used for connecting I/O devices is left up to the system designer.

There are three basic mechanisms for the interaction between the processor and the I/O device. The processor responsibilities of each mechanism, the system resources required, and the complexity of the I/O interface module required by each method are all different. A block diagram showing the relationship between the processor and the I/O device is shown in Figure 6.13. The I/O interface module interacts with the system bus to provide both control signals and data to an I/O device controller. Most I/O device controllers are designed in such a way that they will control a single type of device, such as a disk or tape unit. However. the device controllers are also designed in such a way that multiple copies of I/O devices can be controlled by a single I/O device controller. If another type of I/O device is to be included in the system, then a different I/O device controller is needed, with its associated I/O interface module.

Regardless of transfer mechanism utilized, the processor must have the ability to direct action in the I/O device with instructions; this mechanism we will refer to as "programmed I/O." It is possible to control both the action and the data movement of a device with programmed I/O, as we will see in an example. It is also possible to initiate the action with programmed I/O, and then allow the interface module to interrupt the processor when data is available. This interrupt capability allows the processor to proceed with other work while the data is being obtained, and then to interact with the I/O device only when data is available. Finally, the highest speed is obtained when the interface module has the capability of exchanging data directly with the memory. This is referred to as direct memory access (DMA), and is limited in speed by the transfer rate of the bus. For

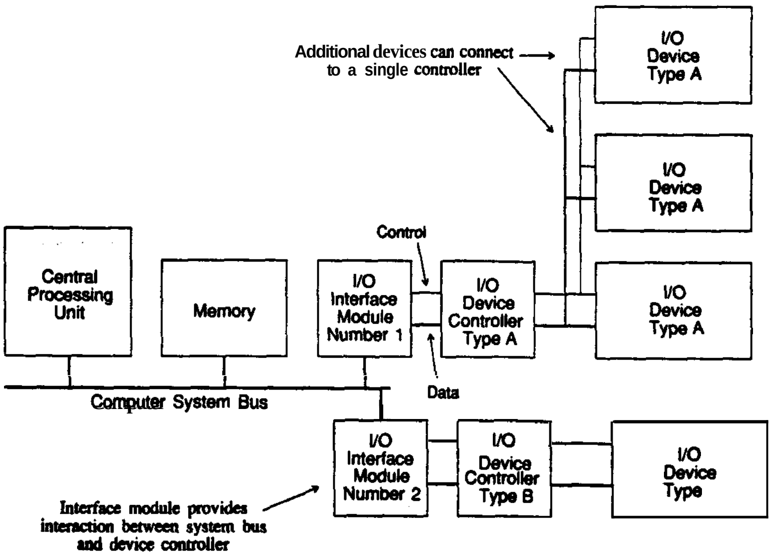

Additional devices can connect to a single controller
I/O Device Type A
I/O Device Type A
Control
Central Processing Unit
Memory
I/O Interface Module Number 1
I/O Device Controller Type A
I/O Device Type A
Data
Computer System Bus
I/O Interface Module Number 2
I/O Device Controller Type B
I/O Device Type
Interface module provides interaction between system bus and device controller

Figure 6.13. Interface System Block Diagram.

DMA transactions, programmed I/O instructions are used to set up a starting address in the system memory and the length of the transfer; and then another programmed I/O instruction initiates the action. The system is then free. to perform other tasks, and the DMA interface module interacts directly with system memory to perform the transfer.

An I/O device is controlled by writing (and reading) information to (and from) specific locations. This method is independent of the type of bus protocol used. but the examples in this chapter will all be done with the asynchronous protocol, as that is the most widely used mechanism at this time. Interface module and I/O devices can be controlled by assigning a specific action to each of the addresses used by an I/O device, or by assigning an action to specific bits or bit patterns at a single address. In either case. the processor sends the command to the interface module by writing to the proper address with the necessary bit pattern. When the interface module receives a write request. it accepts the bit pattern and performs the requested work. When the interface module receives a read request, it supplies the appropriate information to the bus. In this fashion. information can be moved to and from the I/O device.

One of the most frequent inquiries made by a processor concerns the status of the interface module and I/O device, whether it is busy or not, and whether it has data available. Thus, reading a status register in the interface module must be done quickly and easily. The status register usually contains information about the device it is controlling. For example, a tape recorder interface module might have bits in its status word that indicate if the device is on line. if it is busy. if the interrupt is enabled. and so on. The processor is then capable of determining the status of the device by reading the status register.

The simplest interface mechanism results by allowing the processor to control all aspects of the transfer. This method consumes all of the time of the processor, but can be used if the need arises. Since the machine is entirely utilized with the I/O transfer, it is not capable of being used for other tasks during this time, and this is generally not an acceptable cost. Nevertheless, the interface module between the computer system and the I/O controller can be very simple. as shown by the following example.

Example 6.7: Interface module design: Design an interface module that will connect a tape recorder to a 16-bit asynchronous bus for a read only operation using memory mapped I/O techniques. This mechanism is to be controlled by writing command patterns to address $FFFD80_{16}$, reading status at address $FFFD82_{16}$, and by reading the data at address $FFFD84_{16}$. What is the maximum data rate achievable by this mechanism?

We will delay several of the details of the tape recorder side of the interface module, and concentrate on the interaction with the bus. Assuming that the interface method to be used is the four event protocol described in Section 6.1, the lines of interest are the address and data lines, a read line. a request line, and an acknowledge line. One design for this interface module is shown in Figure 6.14, which we will examine by function.

The first function is the address decode and command line interface. The most significant lines of the address are tested with a *gating* network to look for the proper address ($FFFD80_{16}$ – $FFFD84_{16}$). This same function can also be accomplished by using an address decoder chip. such as the 74677. which looks for a specified bit pattern. However, if the address of the device is not known at &sign time, then one mechanism is to use comparators

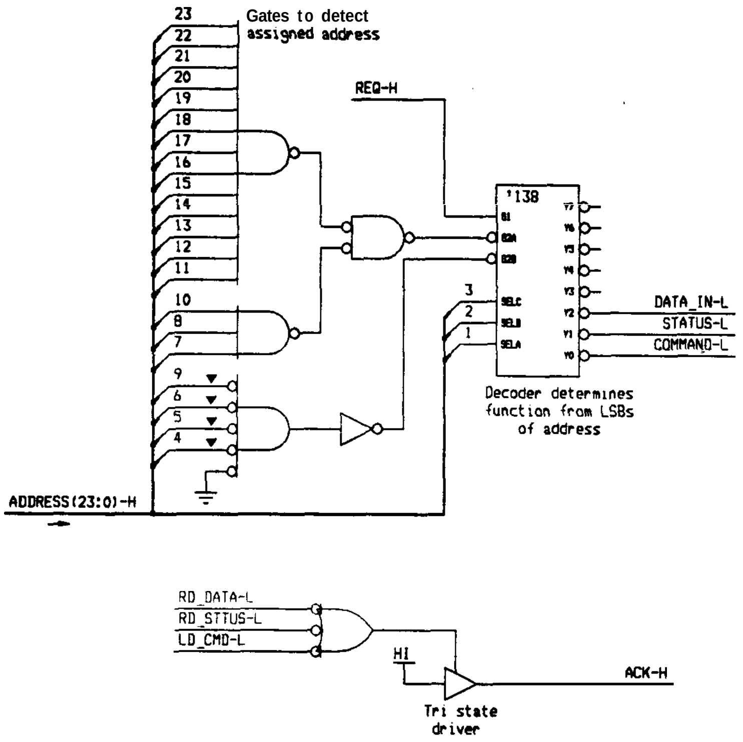


Figure 6.14(a). Tape Recorder Interface Module (Control).

configured with address specifying switches. The least significant lines are directed to a 3-line-to-8-line. decoder, which asserts a line for each of the appropriate addresses. Note that the least significant line (ADDR(0)-H) is not used; we assume that the system is always going to access this information in 16-bit words, properly aligned.

If the address matches, then when the quest line is asserted (REQ-H), the required action is immediately performed, and the acknowledge line asserted (ACK-H). No delay other than the gate delays of the circuitry is inserted into the system since the timing does not require it: information from the master is accepted with edge triggered devices, and the reaction time of the master will account for any hold time needed. Also, information sent to the master is asserted at the same time as the acknowledge line, and 'the master is responsible f a any delays necessary to account for skew on the data lines. Thus, command information (and data. if it is required by the specified interaction) is accepted without delay. Likewise. as soon as a read command is received, the requested information is provided. This register interaction will result in a faster read/write time than normal memory, although it is in the same address space.

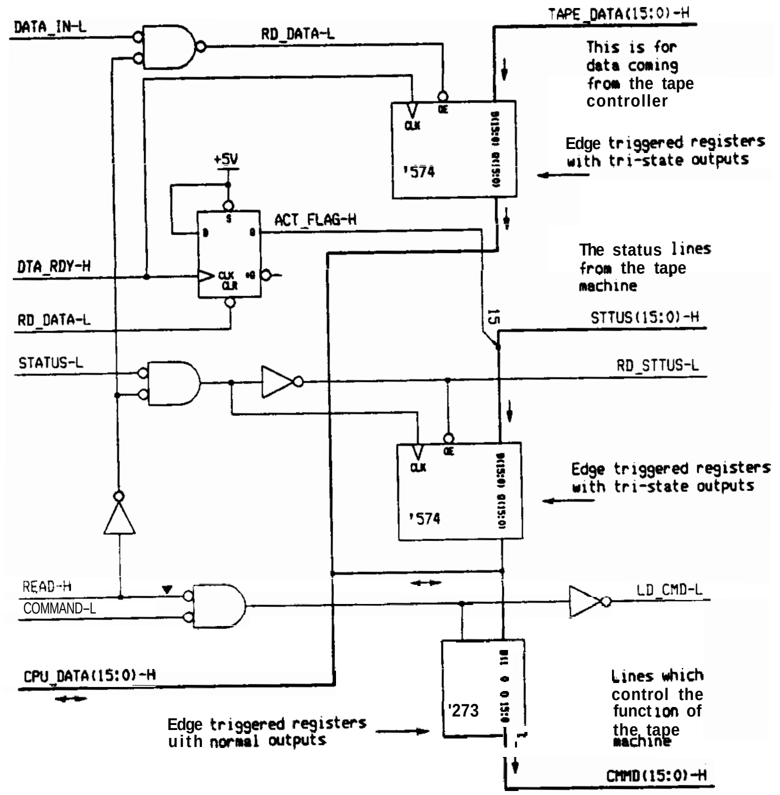

DATA_IN-L
RD_DATA-L
TAPE_DATA(15:0)-H
This is for data coming from the tape controller
OE
CLK
'574
Edge triggered registers with tri-state outputs
+5V
ACT_FLAG-H
S
D
Q
DTA_RDY-H
CLK
CLR
RD_DATA-L
The status lines from the tape machine
STTUS(15:0)-H
RD_STTUS-L
STATUS-L
OE
CLK
'574
Edge triggered registers with tri-state outputs
READ-H
COMMAND-L
LD_CMD-L
CPU_DATA(15:0)-H
'273
Edge triggered registers uith normal outputs
Lines which control the function of the tape machine
CMMD(15:0)-H

Figure 6.14(b). Tape Recorder Interface Module (Data Path).

To demonstrate the programmed I/O mechanism, consider the transactions required to cause the recorder to space forward a block, then read the, next block of 512 words. Assuming that there is a simple assembly language to work with, the following code section will perform the desired work:

```
1          MOV #<FSF>, @FFFD80           FSF is file space forward pattern.
2   one:   TESTI #<BUSY>, @FFFD82        BUSY is pattern to test busy condition
3          JNZ one                         of interface module. Loop to "one" till done.
4          MOV #<512>, R1                Set up the count.
5          MOV #<start addr>, R2         Set up the address.
6          MOV #<DAV>, R3                Set up test pattern for dam available.
7          MOV #<FFFD82>, R4             Set up address of status register.
8          MOV #<FFFD84>, R5             Set up address of data register.
9          MOV #<read cmnd>, @FFFD80     Start read action.
10  two:   TEST R3, *R4                  (800) Is there data?.
```

```
11        JZ two            (450) If not, go back to "two."
12        MOV *R5,*R2+      (1150) If so, move where R2 points.
13        DEC R1            (550) and bump R2; done 512 words?
14        JNZ two           (600) If not, go back to "two."
                            (3550)
```

The first instruction writes out the pattern to indicate to the interface module that the tape recorder should move forward to the next file mark. The next two instructions merely wait until that is accomplished. Instructions 4 through 8 set up the general purpose registers to allow faster processing in the transfer section. Instruction number 9 actually starts the read action of the recorder. Instruction 10 checks to see if the data is available. It is similar in function to instruction 2, which checks to see if the recorder is busy. However, by using values in registers, rather than values in the instruction stream, the time required for the instruction is greatly reduced. In Chapter 4 we identified different instruction times for instruction types, based on the amount of work required by the instruction. Using the times identified there. instruction two requires 1,750 nsec for completion, while instruction 10 can be done in 800 nsec. Instruction 11 is to loop until data is available, when the action moves to instruction 12, which moves the data from the interface module to the designated spot in memory. And with the autoincrement feature of the destination address, the system is ready for the next iteration. Instruction 13 decrements the counter. and instruction 14 loops if the count has not reached zero. The highest data rate will occur when the instructions 10 and 11 are executed but once each iteration. When this occurs. The loop takes 3,550 nsec. Two bytes each 3.550 nsec results in a data rate of 563 Kbyteslsec. This rate cannot be sustained over time. since it does not take into account the time required to set up the transaction.

The above example indicates what can be accomplished by a machine dedicated to performing a single transfer. However, if the device being controlled is a modem or line printer, then the data rate is much lower than that attainable by programmed I/O. Most of the time the machine would be executing the wait loop, waiting for the data movement to occur. Therefore, system designers have often designed the machines in such a way that the interface module can interrupt the action of the computer when data movement is necessary. The positive effect of this is that the machine time that would be used by looping can be effectively utilized for other functions. The negative effect of this mechanism is that the transfer rate will be lower, since more work is needed for each transfer.

Example 6.8: Interface design with interrupt: Consider the system of Example 6.7, but assume that the interface module is also capable of issuing an interrupt when data is available. What is the maximum data rate for the system?

We will make the assumption that an interrupt action causes the current PC and status register to be pushed onto the system stack, and also causes the interrupt service routine to be entered with the vector mechanism discussed in Chapter 4. This mechanism will require about 1.100 nsec in our machine. We include here two sections of code. one of which is used to set up the action, and one of which is actually executed once for each word of data transferred.

```
1     setup:  MOV #<start addr>, @ADDR     Set up the initial address.
2             MOV #<512>, @COUNT           Set up the count value.
3             MOV #<read cmnd>, @FFFD80    Start the read action.
              ...

10    srvce:  MOV @FFFD84, *@ADDR+         (3650) Move the data
11            DEC @COUNT                   (1800) Check the count.
12            JZ more                      (450) If done, do other action.
13            RTI                          (850) If not, return from interrupt.
14    mom:    ...

20    ADDR:   DATA 0
21    COUNT:  DATA 0
```

The first three instructions are used to initialize the starting address and the word count, and to start the actual read action. We are neglecting here the commands necessary to position the tape at the right spot. since additional code to discern between a movement command and a data command would further slow the action of the system. For the data movement action of interest here, the instructions of note are 10 through 13. These perform essentially the same action as the code of Example 6.7; instruction 10 moves the data. instruction 11 decrements the count, and instruction 12 gets out of the loop if the count has reached zero. The count will reach zero when the appropriate number of words have been transferred. and at that point the transfer is complete. If the transfer is not complete (COUNT has not reached zero). then instruction 13 returns the program to the execution in progress when the interrupt occurred.

The difference in instruction execution times results from the fact that now the address information is contained in the instruction stream, and many more references to memory are needed to obtain and manipulate the data. One benefit of this mechanism is that no registers need to be saved upon entering the interrupt service routine. However, the overall time will be greatly increased. with a time for interrupt and interrupt service routine of 7,850 nsec. This results in a maximum data rate of 254 Kbytes/sec.

As can be seen from the example, the data rate for interrupt driven transfers is much less than that achievable strictly with programmed I/O. However, for systems where the data rate is much lower, the interrupt scheme will allow the system to be utilized in other action while the transfer is in progress. In both cases. the action of the interface module and the movement of the data were controlled with programmed I/O instructions.

To increase the data rate of the system requires a more complex interface mechanism, one in which some of the responsibilities of the transfer are moved from the processor to the interface module. The most frequent and time consuming activity is the transfer of data from the device to the memory, and this is precisely the activity committed to hardware. This requires a more complicated interface system, and a simplified diagram of such an interface module is shown in Figure 6.15. The result is a direct memory access interface module, which will interact directly with memory in the transfer of the data.

As can be seen from Figure 6.15, an interface module with DMA capability also contains the basic elements of the programmed I/O interface system: the status register reports the status of the interface module and its associated I/O

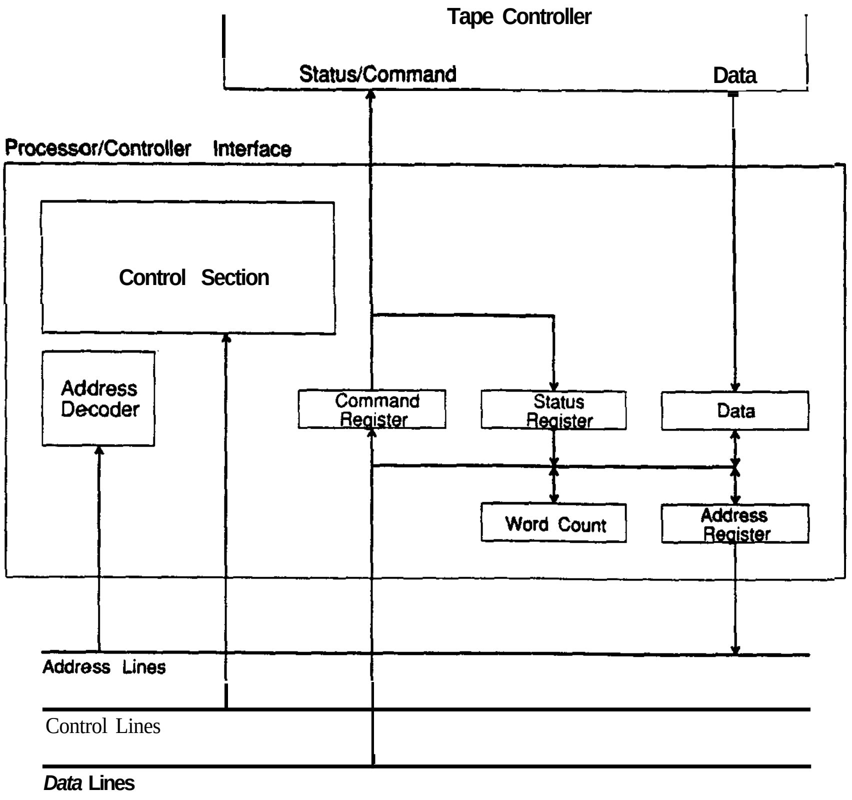


Figure 6.15. Tape Recorder Interface Module with DMA.

device, and the command register controls the action of the unit. However, two other registers have been added: the word count register (WC) and the address register (AR). These registers will be filled (and read, if required) by programmed I/O instructions. The control portion of the DMA interface module must be more complex than the previous interface modules to not only transfer control and status information, but also to control the process of automatic data movement. In general this control portion will be a sequential system designed using the concepts and ideas presented in Chapter 5.

The control of the action of the I/O device (tape movement. head positioning for a disk, etc.) proceeds as before, with programmed I/O instructions directing the appropriate movement, and the device interrupting when the specified action has been completed. However, when data movement is called for, then the code controlling the unit (commonly called the "I/O driver routine") will, with programmed I/O instructions, fill the WC register with the number of words to transfer, and the AR with the starting address in memory where this transfer is to take place. The transfer of information is then initiated with programmed I/O instructions. When the data becomes available, the interface module requests control of the bus, performs the necessary transfer, and relinquishes control of the bus. The address for the transfer is provided by the DMA interface module. After the transaction is complete, the address is changed to point to location to be used by the next transfer. In addition, the word count is decremented to keep track of the number of transfers that have occurred. Thus, the hardware handles the information transfer after data starts to Row. Using this technique, the data can be transferred at a rate limited only by the bus speed. This allows high speed devices, such as disk units, to exchange information at the data rates of the disk. A disk using an SMD protocol can transfer information in excess of 3 Mbytes/sec.

Example 6.9: *Interface* design *with* DMA: Modify the tape recorder interface module developed in Example 6.8 to include DMA capability. The word count register is accessed as location $FFFD86_{16}$, and the address register is accessed as location $FFFD88_{16}$ for the least significant 16 bits, and location $FFFD8A_{16}$ for the most significant 8 bits.

We will delay the design of the control system until the following section, but the other elements are shown in Figure 6.15. The use of the decoder is expanded to include the additional addresses required by the word count and address registers. Note that these registers are readable as well as writable. This does not improve the functionality of the unit. but will provide valuable help for both checkout and test.

The need for action on the pan of the sequential controller is indicated by a hardware flag in the control section of the interlace module. This is set when the command register is tilled by programmed I/O. One of the responsibilities of the controller is then to reset this Rag when the action has been initiated. When the action requires tape movement, such as file space forward, then the interface module requests the movement from the tape controller and waits for the completion of the action. When data movement is required, the specified tape action is requested, and when a data transfer is necessary, the appropriate bus cycle is initiated.

The transfer rate of this mechanism is limited by the bus speed of the system. For a bus system with a transaction time of 250 nsec, the maximum date rate would be 8 Mbytes/sec. This rate is somewhat inflated, since no allowance is made for other users on the bus or for the cost of bus arbitration.

This section has dealt with the transfer of information between a processor and an I/O device. There is a tradeoff in complexity of hardware and processor time to transfer information. If the complexity of the interface module is kept simple, then the responsibility of the processor to control the I/O device and the data movement increases. For transfers conducted purely with programmed I/O instructions, the processor must either continuously monitor the appropriate status lines, or it must interrogate them periodically (polling) to ascertain if any action is necessary. In either case, a large portion of the processor time is devoted to conducting the transfer.

If the concept of interrupts is utilized, then the processor is able to ignore the I/O device until action in needed, at which point the interface module will cause an interrupt, requesting interaction with the processor. The benefit of the use of interrupts is that the processor is free to do other work while the I/O device does not need supervision. The *cost* of this policy is the decrease in the speed of possible transfers. This policy is especially beneficial for action that does not involve data transfers, such as tape movement or positioning of disk heads.

The highest speed is achieved by direct interaction between the interface module and the memory, with the use of DMA. This method requires more complex hardware, but is capable of very high speed transfers. DMA interface modules combine the various techniques to achieve the high data rates. Programmed I/O instructions are used to communicate with the various registers that control the action of the I/O device. Interrupt techniques inform the processor that a requested action has been completed. The controller of the interface module interacts directly with the bus to transfer the data with minimal overhead, needing only the time required for successful bus cycles.

6.5. An Example of a Device Interface Module

Many of the concepts discussed in the preceding sections are more easily visualized when a specific example is utilized. For that reason. we will use the tape recorder mechanism that was the object of the previous examples, and we will design a simple DMA interface module capable of a limited amount of interaction. The interface module will control the behavior of tape drives as directed by the programmed I/O instructions issued by the CPU. Thus, the interface module should combine all of the techniques discussed: respond to instructions. assert signals going to the tape controller, cause interrupts. and control DMA transfers.

The task facing a designer is to ascertain the requirements of the system and build a device that will satisfy those requirements. In this case, we need information concerning three different facets of the design. Two of these are indicated in Figure 6.15, which shows the relationship of the interface and the tape controller. One piece of information is the bus specification, which identifies the elecmcal and timing requirements of interaction with a bus module. The other device specific information is the set of control and data signals used by the tape recorder. To perform the needed tape movement, read, and write operations, the device must assert these lines in the manner defined by a controller specification. The final piece of information needed is a definition of the commands to be issued and the status to be interrogated by the CPU. Thus, before the design process can begin, information about the electrical and behavioral characteristics of the interface module must be established.

The bus used for this design is the UNIBUS, but the same techniques would be applicable on a Multibus, Q-bus, VME bus, and so on. Each bus has its own characteristics, and these characteristics must be considered in the process of doing a design. The UNIBUS is relatively simple, yet it includes the salient points addressed by the previous sections. Also, because it is a 16-bit bus. the transfer techniques are not overshadowed by an enormous number of wires. To match the electrical characteristics we will use special gates, and to satisfy the timing characteristics we will use a sequential system designed with a state machine approach. Other bus systems, such as the VME bus, will use more standard gates for their interaction, but the techniques will be the same.

The tape controller that is the object of this design is capable of controlling up to four 9-track tape transports. The data path to the controller is separate from the data path coming from the controller, but both paths are 8 bits wide. Parity is used to create the ninth track for the tape, but the controller itself takes care of parity operations. In addition to the data lines, there are command signals and status indicators associated with the tape controller. The command lines are indicate in Table 6.1. Asserting these lines in the proper fashion will result in the desired control over the tape drive and the date movement The designer must create the interface module in such a way that the signals are asserted properly. The third control is labeled SETX-L, and it used to synchronize the other commands listed. For example, when a write file mark command is required, the WRITEFM-L line is asserted, and then the SETX pulse causes the tape controller to accept the command and begin the specified work.

The control signals of Table 6.1 are used to activate the controller and perform work, but in addition to that the CPU often needs to know the status of the tape system. For this reason, a number of status lines are provided, as shown in Table 6.2. These signals are received and delivered to the CPU when the appropriate programmed I/O instruction is given.

The UNIBUS specification is used to identify the required signals on the bus side of the interface module. The controller specification provides the signals given in Table 6.1 and Table 6.2. which identify the signals of the tape drive side of the interface module. With this information a preliminary data path block diagram can be formed, and this is given in Figure 6.16.

The initial registers are identical in function to those identified in previous sections in this chapter. The command register is used to receive commands from

Tabk 6.1. Control Lines to Tape Controller.

Signal	*Function*
INIT-L	Pulse to initialize transport
SET_TRAN(3:0)-L	Level to select active transport
SETX-L	Pulse to synchronize action requests
WRITE-L	Level to identify function type
READL	Level to identify function type
INPUTX-L	Command for input data
OUTPUTX-L	Command for output data
FILESRCHF-L	Command for file search forward
FILESRCHR-L	Command for file search reverse
SYNCFWD-L	Command for synchronous forward action
SYNCRVS-L	Command for synchronous reverse action
WRITEFM-L	Command for write file mark
REWIND-L	Rewind command pulse

Table 63. Status Signals Available from Controller.

Signal	Function
TAPE–READY-L	Tape transport and controller ready
P_ERROR-L	Parity error
EOF-L	End of file mark detected
BOT-L	Tape located at beginning of tape mark
EOT-L	Tape located a end of tape mark
PROTECT-L	Tape transport senses no write ring
RWDING-L	Tape is minding
SEL0-L	Transport 0 selected
SEL1-L	Transport 1 selected
SEL2-L	Transport 2 selected
SEW-L	Transport 3 selected

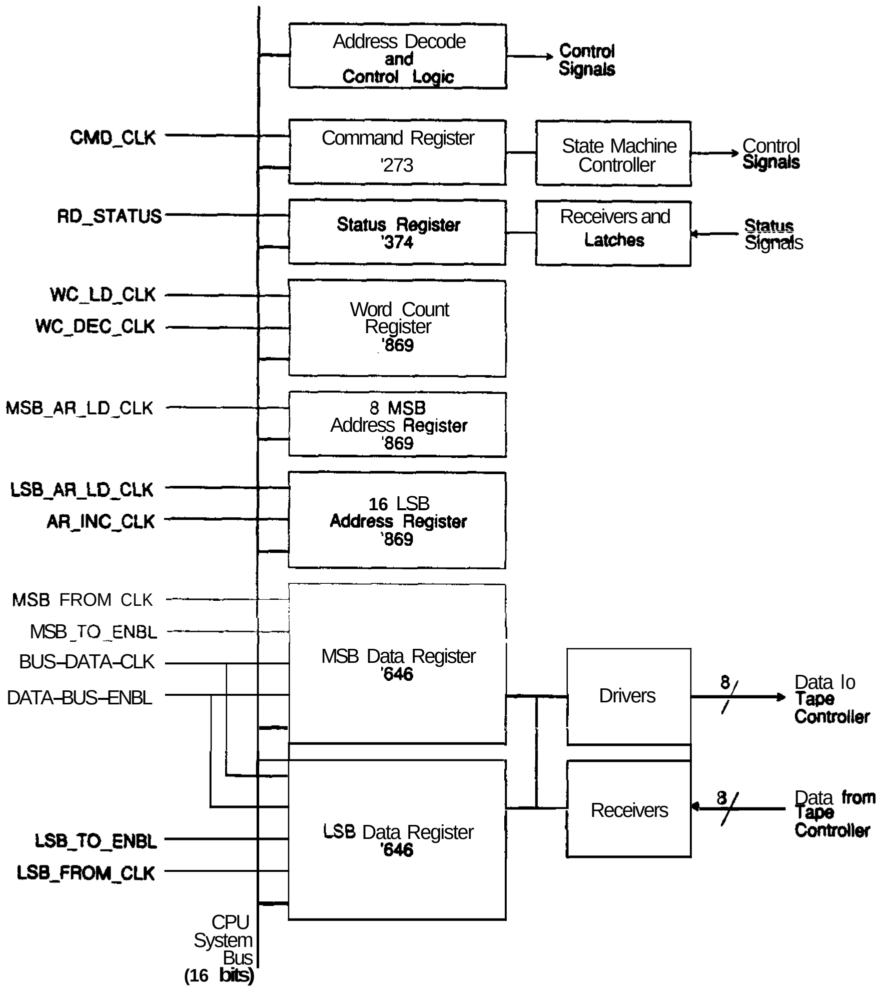


Figure 6.16. Data Path Block Diagram for **Tape Recorder Interface Module.**

the system: these will be acted upon by the control portion of the interface module. The status register allows the CPU to investigate the current status of the system and the selected transport. The word count and address registers operate as described in Section 6.4. The registers that have not been mentioned yet are the registers used to hold the data transferred to or from the tape transport. The output registers accept a word from the bus, and then, under control of the interface module, the bytes are alternately sent to the transport. The transport itself adds parity for the ninth bit. The path from the transport to the bus is the reverse of that described above; the bytes are accepted one at a time, then the 16-bit word is sent to the bus. If it is desirable to be able to transfer an odd number of bytes, then the interface module becomes correspondingly more complex.

Along with the initial data path, we also need a behavioral description of the interface module. We know that we want commands to cause tape movement, as well as commands for reading and writing. We will design a system capable of selecting one of the transports, and on the selected transport performing one of the following commands:

- ***Read forward (R):*** Tape motion is initiated by the tape controller to read data. Before this command is issued it is imperative that the interface module be initialized with a word count indicating the number of words to transfer. and an address where the data is to be located. Data delivered from the tape controller will be placed into memory by the interface module. After the block of data has been read from the tape. the number of words read is compared to the number of words expected: if they differ, an error bit is set in the status word.
- ***Write forward (W):*** Tape motion is initiated by the controller, with controls configured for a write. Again, it is imperative that the word count and address registers have been properly initialized for the required transfer. DMA transfers are performed by the interface module, and the data delivered to the controller. When the word count reaches zero. the action is stopped.
- ***Write file mark (WFM):*** A tile mark is written onto the tape by the controller.
- ***File search forward (FSF):*** The word count register must be tilled with a number indicating how many tile marks should be skipped. The interface module issues the appropnate number of tile search commands, halting when the word count has been decremented to zero.
- ***File search reverse (FSR):*** This command positions the tape by searching in the reverse direction. It is assumed that this request will be given (with an appropriate word count) to position the tape *after* the file mark in question. That is, if the tape is positioned in the middle of a file, an FSR command with a word count of one will back up one file mark, then read over the file mark. The net result is to position the unit at the beginning of the fik. An FSR command with a word count of two will position the tape at the beginning of the file before the current position of the tape.
- ***Rewind (REW):*** A rewind pulse is sent to the tape controller. The net result is a rewind action on the selected drive.
- ***Enable interrupt (INTE):*** The intermpt capability of the interface module is enabled. This will be indicated as a bit in the status register.
- ***Disable interrupt (INTD):*** The interrupt capability of the interface module is disabled.

Each command involving tape movement will cause an interrupt (if the intermpt facility is enabled) when the command has been completed. In addition. the system should provide various status information about the condition and configuration of the selected tape transport.

The command required of the interface module will be supplied over the bus and loaded into the command register. The commands are not ASCII words, but rather consist of bit patterns defined in advance to identify the desired action. For this project we define the following bit patterns as instructions:

Action	*Bit Pattern*
Read	000100
Write	001000
Write file mark	001100
File search forward	010000
File search reverse	010100
Rewind	011000
Enable interrupt facility	100101
Disable interrupt facility	100100
Select transport (xx = 0,1,2,3)	1000xx

The bit patterns identify the six LSBs of the word; the other bits are not tested in the system. Note that a pattern of all zeros is not a legal instruction. Also, the actions that do not require tape movement (transport select, intermpt enable, interrupt disable) all have a 1 in the sixth position. This will simplify some of the hardware of the system. The bit patterns used to specify the action of machine interface modules should have some correlation between the defined patterns and the hardware requirements of the interface modules. This is just one of the many examples where communication between users of computers (programmers, systems personnel. etc.) and builders of computers should communicate requirements and preferences.

The above commands are given to the interface module by writing the appropriate bit pattern to the command register. The status of the tape drive is obtained by reading the status register. This information is obtained from the signal lines identified in Table 6.2. Other signals are available, but this set will be sufficient to demonstrate the elements of our design. From those signals, as well as from signals generated by the control of the interface module, we will configure a status register as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BSY	SMB	ERR			INTE	RDW	WRP	PE	BOT	EOT	EOF	Trans			

The bits are defined as follows:

- BSY: Busy bit, derived directly form the TAPE–READY signal from the tape controller.
- SMB: State machine busy, indicates when the controller of the interface module is not in the idle state.
- ERR: Record length error, which will occur when the number of words read fmm a block on the tape does not agree with the expected number.
- INTE: Interrupt enable bit, which is a 1 when the interrupt facility of the interface module has been enabled.
- RDW: Rewinding, set when the selected transport is in the process of rewinding.
- WRP: Write pmtect, which is a 1 when the selected transport docs not detect the presense of a write ring on the tape.
- PE: Parity error, which is set when the Last operation detected a parity error.
- BOT: Beginning of tape, indicates that the tape is located at the beginning of tape marker.

- EOT: End of tape, indicates that the tape is located at the end of tape marker.
- Trans: These 4 bits indicate which of the four transports will be controlled by the interface module.

Reading the status causes these values to be loaded into a register, so that if they should change while rhe instruction is being executed that change will not cause problems with the instruction itself.

The list of commands for the tape system does not have a direct correspondence with the signal lines given for the tape controller. Thus, the designer must identify the desired action and assert the control lines accordingly. For the controller used here, the functions identified above are obtained by asserting the lines according to the following table:

	Read	*Write*	*Write File Mark*	*File Search Forward*	*File Search Reverse*	*Rewind*
SETX-L	X	X	X	X	X	
WRITE-L		X	X			
READ-L	X			X	X	
FILESRCHF-L				X		
FILESRCHR-L					X	
SYNCFWD-L	X	X	X	X		
SYNCRVS-L					X	
WRITEFM-L			X			
REWIND-L						X

Note that the SETX line is to be asserted for all motion commands, except rewind. which requires a pulse on only the REWIND line. The INPUTX and OUTPUTX signals are pulses that activate the data transfers; that is, when the tape controller needs (or has) data for transfer, it will request this information. The interface module must respond by providing (or accepting) data on the byte-wide set of data lines to the tape drive and asserting OUTPUTX (or INPUTX). All of the other lines can be levels, and our design will treat them as such.

The interface module must be electrically compatible with both the bus with which it is working and with the tape controller. The UNIBUS requires specific set of interface chips that provide a minimal load on the bus; sample gates used by the interface module are shown in Figure 6.17(a). The assertion level on the bus is low, and these chips convert from the high assertion levels used in the interface module to the low assertion levels used on the bus. Other bus systems may use standard tri-state devices, or have other requirements, but the design process must adhere to the specification of the bus. We will not include all of the individual gates in the drawings shown in this section, but we will assume that rhese gates are used to match the electrical requirements needed by the signals. The drawings in this section will include the major blocks and some of the control signals involved. A more complete set of schematics can be found in Appendix B.

The tape controller also has a specification for driving and receiving the control, status. and data lines. This specifically calls for open collector drivers for the signal lines going to the controller. and resistor networks (220Ω to +5, 330Ω to ground) on the signals arriving from the controller. The effective impedance of this combination (220Ω in parallel with 330Ω) is about 130Ω, which is a

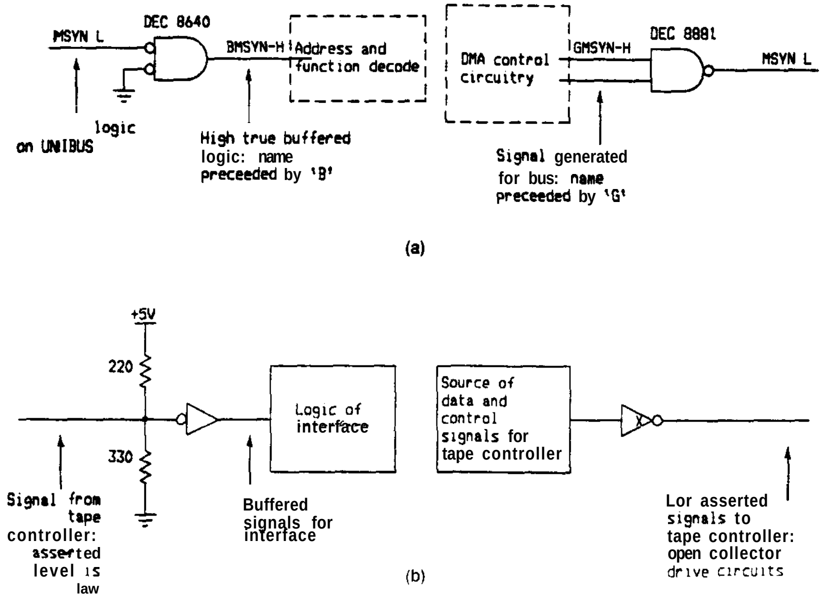
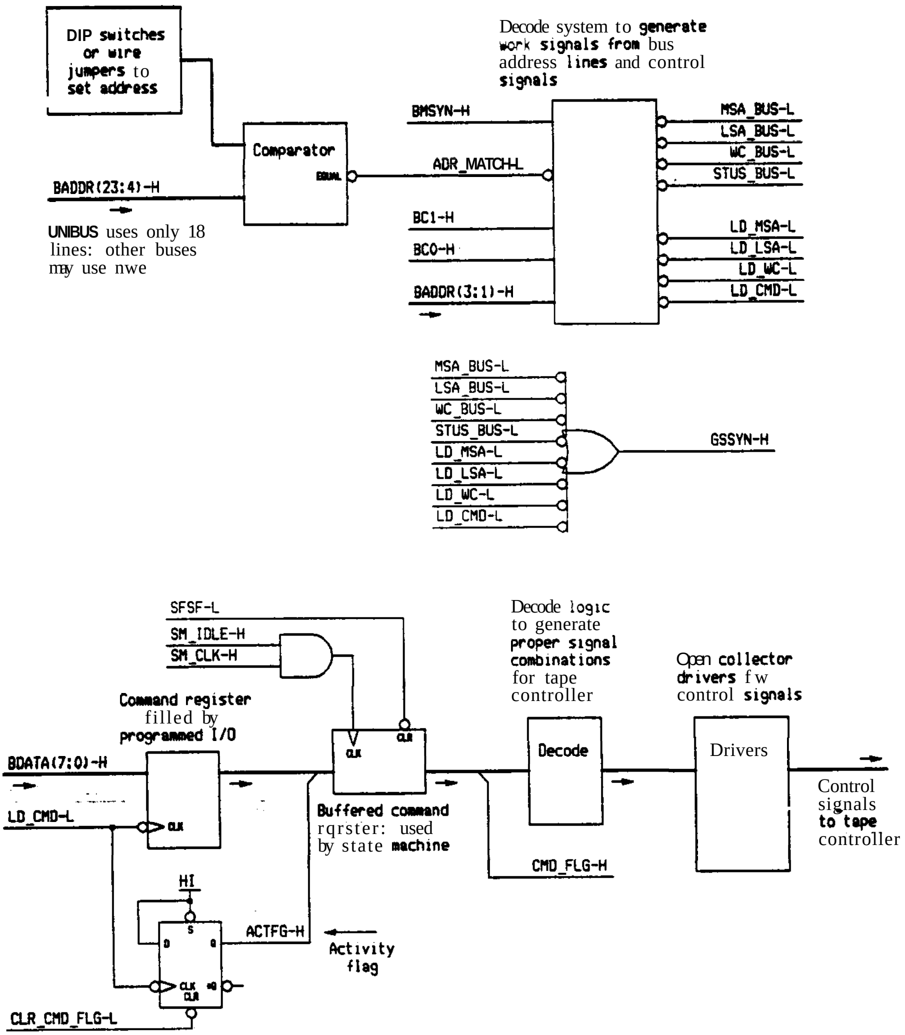

DEC 8640
MSYN L
BMSYN-H
Address and function decode
logic on UNIBUS
High true buffered logic: name preceeded by 'B'
DMA control circuitry
GMSYN-H
DEC 8881
MSYN L
Signal generated for bus: name preceeded by 'G'
(a)
+5V
220
330
Logic of interface
Source of data and control signals for tape controller
Signal from tape controller: asserted level is law
Buffered signals for interface
Lor asserted signals to tape controller: open collector drive circuits
(b)

Figure 6.17. Interface Gates Used for Bus, Controller.

reasonable match for many signal transmission mechanisms. For this reason. it has been used for many years as the method for terminating signals, as shown in Figure 6.17(b).

The diagrams for the interface module are included as Figure 6.18. and we will describe the various sections and their responsibilities. The address decode and programmed I/O control signals are found in Figure 6.18(a). Gates have been provided to minimize the load presented to the address bus, and the buffered lines are labeled BADDR. These lines are used to compare the address against an addressed set up by the user. This mechanism allows the address to be determined at installation time rather than design time. The UNIBUS uses only 18 address lines, so the other lines indicated in the figure are supcrfluous; however, other bus systems use up to 32 bits in the address. The least significant address lines and buffered control lines from the bus are used to create signals used in the interface module. These signals allow the sequentiality of the bus protocol to provide the timing necessary to read and write registers under programmed I/O control. The request line for the UNIBUS is called MSYN, and its buffered version is shown in Figure 6.18(a). The acknowledge is identified as SSYN, and it is shown before being sent to the bus with the required bus matching gates.

Also included in Figure 6.18(a) is the command register. The action of filling the command register also sets a flag (ACTFG), which will be tested by the state machine that directs the interaction with the tape. controller. The contents of 'the command register and the activity flag are inputs to a second register, which is labeled the buffered command register. This register, which is clocked whenever the system is in an idle state, has two purposes. The first is to synchronize the



DIP switches or wire jumpers to set address
Comparator
EQUAL
BADDR(23:4)-H
UNIBUS uses only 18 lines: other buses may use more
Decode system to generate work signals from bus address lines and control signals
BMSYN-H
ADR_MATCH-L
BC1-H
BC0-H
BADDR(3:1)-H
MSA_BUS-L
LSA_BUS-L
WC_BUS-L
STUS_BUS-L
LD_MSA-L
LD_LSA-L
LD_WC-L
LD_CMD-L
GSSYN-H
SFSF-L
SM_IDLE-H
SM_CLK-H
Command register filled by programmed I/O
BDATA(7:0)-H
LD_CMD-L
CLK
CLR
Buffered command register: used by state machine
Decode logic to generate proper signal combinations for tape controller
Decode
Open collector drivers for control signals
Drivers
Control signals to tape controller
CMD_FLG-H
HI
S
D
Q
CLK
CLR
ACTFG-H
Activity flag
CLR_CMD_FLG-L

Figure 6.18(a). Programmed I/O Control and Command Registers.

filling of the command register with the clock of the state machine system. Without this mechanism the system would fail when the contents of the command register changed during the sensitive time before the active edge of the state machine clock. The second reason for the second register is to prevent any change in the command register from affecting a function in progress. The contents of the buffered command register are decoded and the appropriate control lines asserted to the tape controller. The commands and levels will be determined by this logic; the required pulses will be generated by the state machine.

The word count register and the address register are found in Figure 6.18(b). These registers must act as registers to be filled by pmgrammed I/O instructions, and as counters to be decremented when under the control of the state machine. This is accomplished by using two clocking sources. When the state machine is idle, SM_IDLE will be asserted, and the clock is derived from the programmed I/O signals. However, when the state machine is not idle, the registers are decremented by a signal fmm the state machine itself (WC_DEC-L). One feature of the system that will not be used in the normal action of transfers is the ability to read the contents of the word count register. This is provided by the tri-state drivers included with the word count and address registers; when the appropriate enable

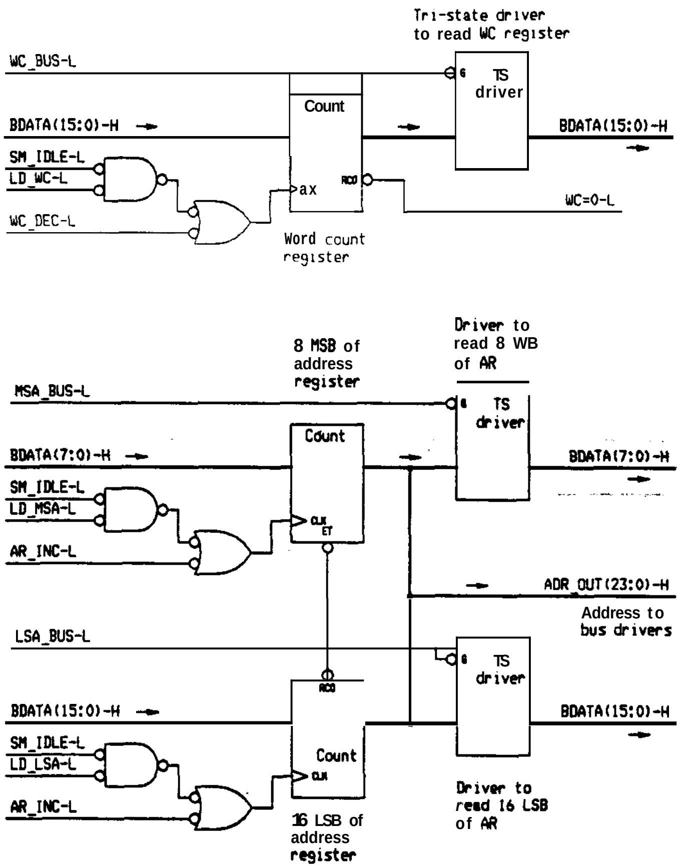


Figure 6.18(b). Word Count Register and Address Register.

signal is asserted, the information is enabled onto the internal data bus (BDATA) and then to the UNIBUS.

The data path is included in Figure 6.18(c). The path to and from the bus is provided by a pair of bidirectional registers. These devices contain two registers, one for each direction. During a write operation, information from the bus is obtained 16 bits at a time, and loaded into the register by a signal derived from the request line (DMA_OUT_CLK). This information is then fed one byte at a time to the controller, using the TAPE-DATA lines. The selection of the byte to send to the controller is handled by the enable lines (LSB_ENBL, MSB_ENBL), which are alternately enabled during a write operation. The timing signals for loading and reading the registers are generated using control signals from the tape unit and the state machine.

During a read operation, the data path is reversed. The register in the reverse direction is loaded by a signal derived from the control lines of the tape controller (TAPE-MSB, TAPE_LSB). The resulting values are enabled onto the internal data bus when needed during the DMA operation. As noted earlier, the data lines to the controller are asserted with open collector drivers, and the lines from the controller are received with resister networks. The state machine controller is responsible for asserting the appropriate information onto the TAPE_DATA lines, from one of the DMA output registers or the data in lines.

The status register is also included in Figure 6.18(c). This register monitors signals from the tape controller and from latches internal to the interface module itself. such as the interrupt enable bit. When a programmed I/O instruction requests this information, it is loaded into a register to keep it stable during the read operation. The register has tri-state outputs that directly connect to the internal data bus. This status information will be enabled onto the UNIBUS at the appropriate time by the DMA system.

The logic for controlling the interface module is shown in Figure 6.18(d). There are two state machines in this implementation. The first is for the action of the interface module itself, the second is for the DMA controller. The controller specification calls for command pulses which are a minimum of 200 nsec. For that reason, the state machine controlling the action of the interface module is clocked at 5 MHz, which provides a 200 nsec state time. We will describe the state diagrams for the system later in this section. The state machine used to control the interface module is constructed from two registered PROMs, each of which contains 2,048 words of 8 bits. This requires 11 bits of address. Five of the 11 bits are provided by the present state of the system: the remaining 6 bits are derived from the inputs to the system. Since more than 6 inputs are required to control the state machine, the 6 used at any one time are selected with a multiplexer network. The outputs of the registered PROMs provide the needed control and state information. The two devices together have 16 outputs: five of these are used for present state information, and 11 are used for control signals.

The state machine which is used for DMA and interrupt requests is also included in Figure 6.18(d). This consists of two parts: a synchronizing register and a registered PAL. These parts need to be capable of fairly high speed, since the cycle time for the unit is 50 nsec. The function of the DMA controller is to control the interaction with the bus for direct access to memory. The outputs of the PAL drive both the bus signals and the internal registers involved in the DMA transactions. Also included with the control circuitry are some flags that handle communication between the two state machines, and a timer used to create a rewind signal that is longer than 2 μsec.

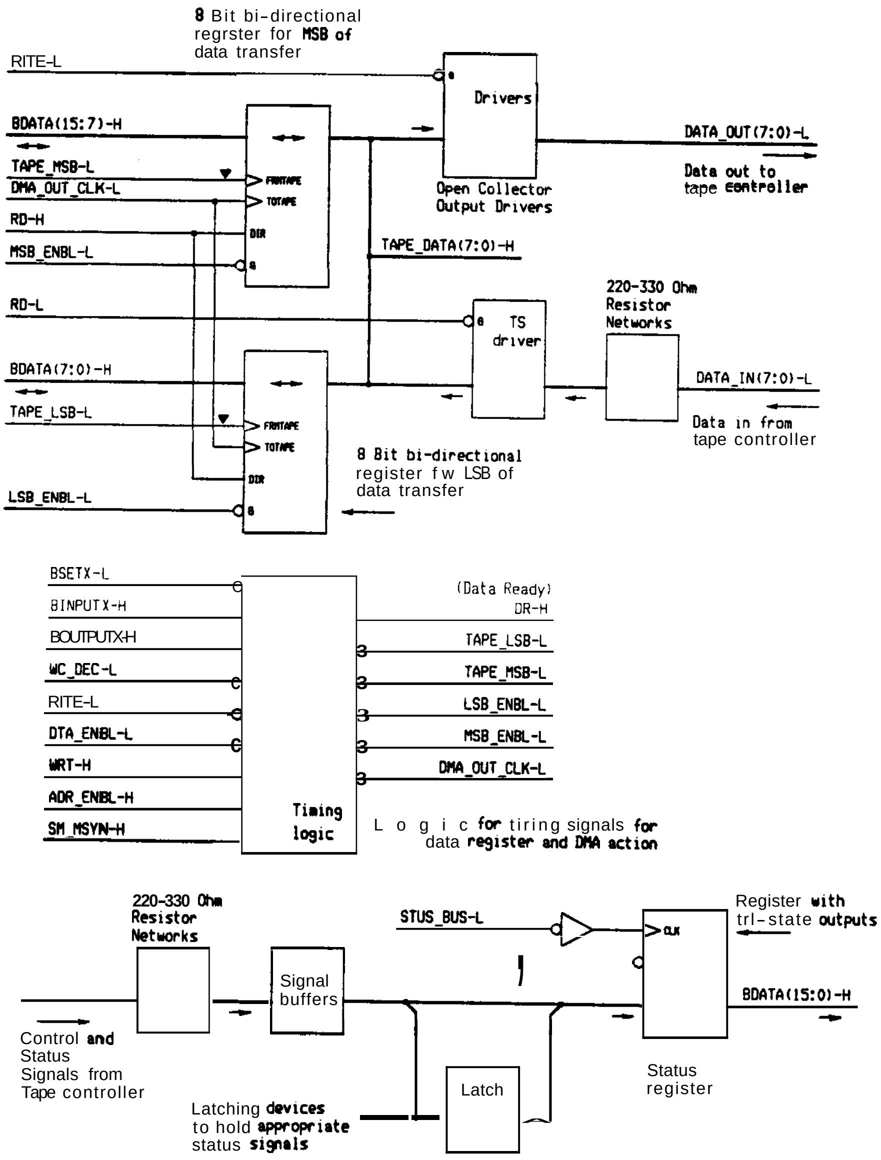

8 Bit bi-directional regrster for MSB of data transfer
RITE-L
Drivers
BDATA(15:7)-H
TAPE_MSB-L
DMA_OUT_CLK-L
RD-H
MSB_ENBL-L
FRMTAPE
TOTAPE
DIR
DATA_OUT(7:0)-L
Data out to tape controller
Open Collector Output Drivers
TAPE_DATA(7:0)-H
220-330 Ohm Resistor Networks
RD-L
TS driver
BDATA(7:0)-H
TAPE_LSB-L
DATA_IN(7:0)-L
Data in from tape controller
8 Bit bi-directional register fw LSB of data transfer
LSB_ENBL-L
BSETX-L
BINPUTX-H
BOUTPUTX-H
WC_DEC-L
RITE-L
DTA_ENBL-L
WRT-H
ADR_ENBL-H
SM_MSYN-H
(Data Ready) DR-H
TAPE_LSB-L
TAPE_MSB-L
LSB_ENBL-L
MSB_ENBL-L
DMA_OUT_CLK-L
Timing logic
L o g i c for tiring signals for data register and DMA action
220-330 Ohm Resistor Networks
STUS_BUS-L
Register with trl-state outputs
CLK
Signal buffers
BDATA(15:0)-H
Control and Status Signals from Tape controller
Latch
Status register
Latching devices to hold appropriate status signals

Figure 6.18(c). Data Path to and from Tape Controller and Status Register.

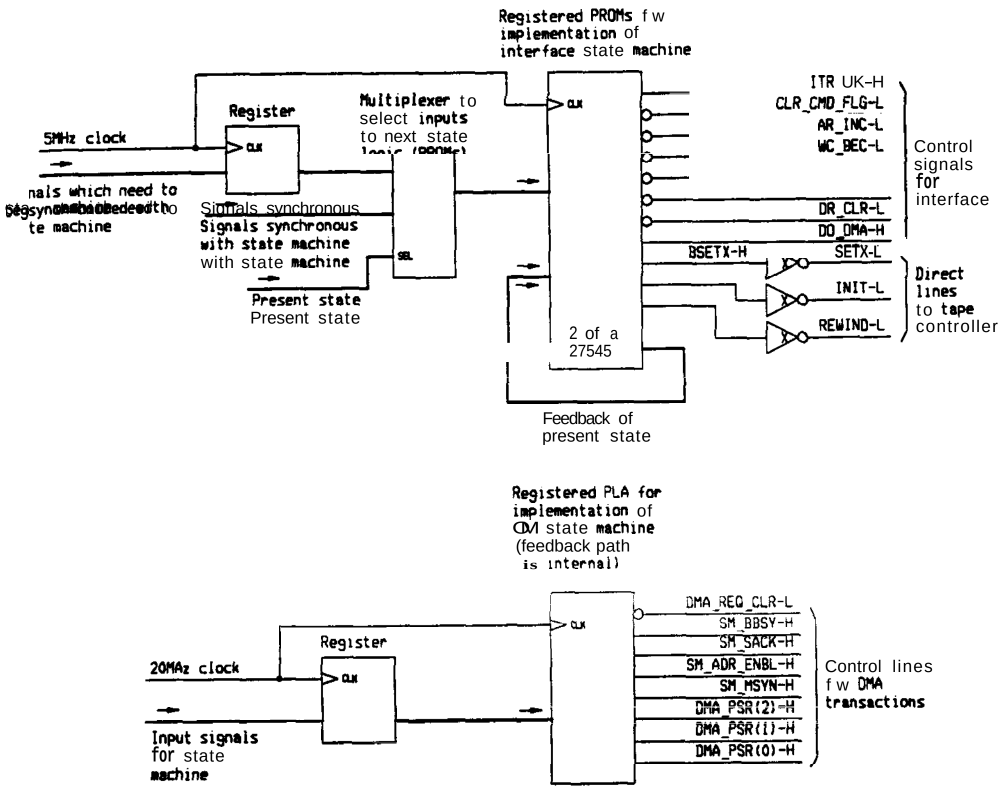

Registered PROMs for implementation of interface state machine
Multiplexer to select inputs to next state logic (PROMs)
Register
5MHz clock
CLK
Signals which need to be synchronized to state machine
Signals synchronous with state machine
Present state
SEL
2 of a 27S45
Feedback of present state
ITR UK-H
CLR_CMD_FLG-L
AR_INC-L
WC_BEC-L
DR_CLR-L
DO_DMA-H
BSETX-H
SETX-L
INIT-L
REWIND-L
Control signals for interface
Direct lines to tape controller
Registered PLA for implementation of DMA state machine (feedback path is internal)
Register
20MHz clock
Input signals for state machine
DMA_REQ_CLR-L
SM_BBSY-H
SM_SACK-H
SM_ADR_ENBL-H
SM_MSYN-H
DMA_PSR(2)-H
DMA_PSR(1)-H
DMA_PSR(0)-H
Control lines for DMA transactions

Figure 6.18(d). State Machine Controllers.

The last of the diagrams, Figure 6.18(e), contains the logic needed to connect to the bus, with the requisite gates matching the bus requirements. The address and data buses are provided with both receivers and drivers to present a minimal load to the bus as required by the bus specification. The interrupt vector address can be specified by the user, and the interface module will assert this information onto the bus at the appropriate time.

The action of the interface module is described by the state diagrams included as Figure 6.19. As mentioned above, the minimum time for the tape controller is 200 nsec, whereas the UNIBUS will be most effectively used if the state times are much less than 200 nsec. Thus, the state time for the controller of the interface module itself is 200 nsec, while the DMA controller operates with a 50 nsec clock. It would be possible to combine the two state machines, but that would result in a much larger system. Hence, the decision was made to use two different state machines.

Figure 6.19(a) deals with the behavior of the interface module itself. The states used to perform the work of directing the tape controller are identified, and the signals that need to be asserted are identified in each state. A description of

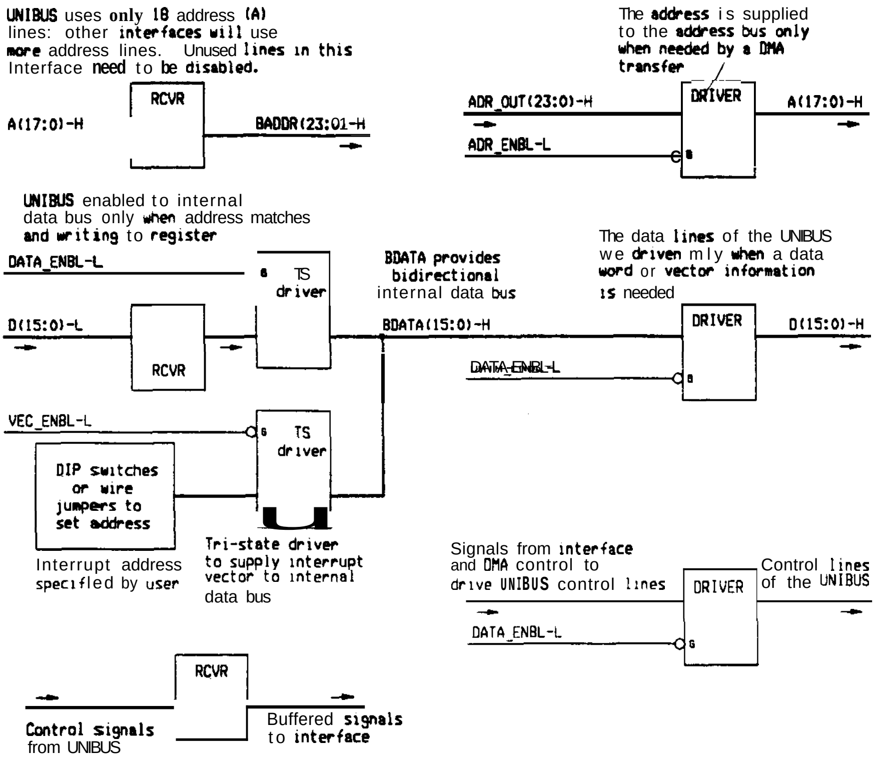

UNIBUS uses only 18 address (A) lines: other interfaces will use more address lines. Unused lines in this interface need to be disabled.
RCVR
A(17:0)-H
BADDR(23:0)-H
The address is supplied to the address bus only when needed by a DMA transfer
ADR_OUT(23:0)-H
ADR_ENBL-L
DRIVER
A(17:0)-H
UNIBUS enabled to internal data bus only when address matches and writing to register
DATA_ENBL-L
TS driver
BDATA provides bidirectional internal data bus
The data lines of the UNIBUS are driven only when a data word or vector information is needed
D(15:0)-L
RCVR
BDATA(15:0)-H
DRIVER
D(15:0)-H
DATA_ENBL-L
VEC_ENBL-L
TS driver
DIP switches or wire jumpers to set address
Interrupt address specified by user
Tri-state driver to supply interrupt vector to internal data bus
Signals from interface and DMA control to drive UNIBUS control lines
DRIVER
Control lines of the UNIBUS
DATA_ENBL-L
RCVR
Control signals from UNIBUS
Buffered signals to interface

Figure 6.18(e). Circuitry for Bus Interaction.

the purpose of each state is included in Appendix B: here we will briefly describe some of the action generated by the state machine.

The interface module is initialized by forcing the present state to zero, since that is a relatively easy thing to do with the present state register. This is used to initialize both the electronics of the interface module and the tape controller. Once the initialization has occurred. the interface moves to the idle state, where it will await further direction.

If the instruction which is received by the interface module does not require tape movement, then State 3 is visited. This causes the appropriate information to be clocked into the retaining registers and the action flag to be cleared; then the system returns to the idle state.

If a command that requires tape movement is received, then the system moves to State 4. If a rewind is required, then the system moves to State 20 to issue a long enough pulse, then to State 23 to await then completion of the tape movement. If a write file mark (WFM) command is desired, the system moves to State 6 to create the SETX pulse, then to State 23 to await the completion of the tape movement. Note that the appropriate command lines to the tape controller

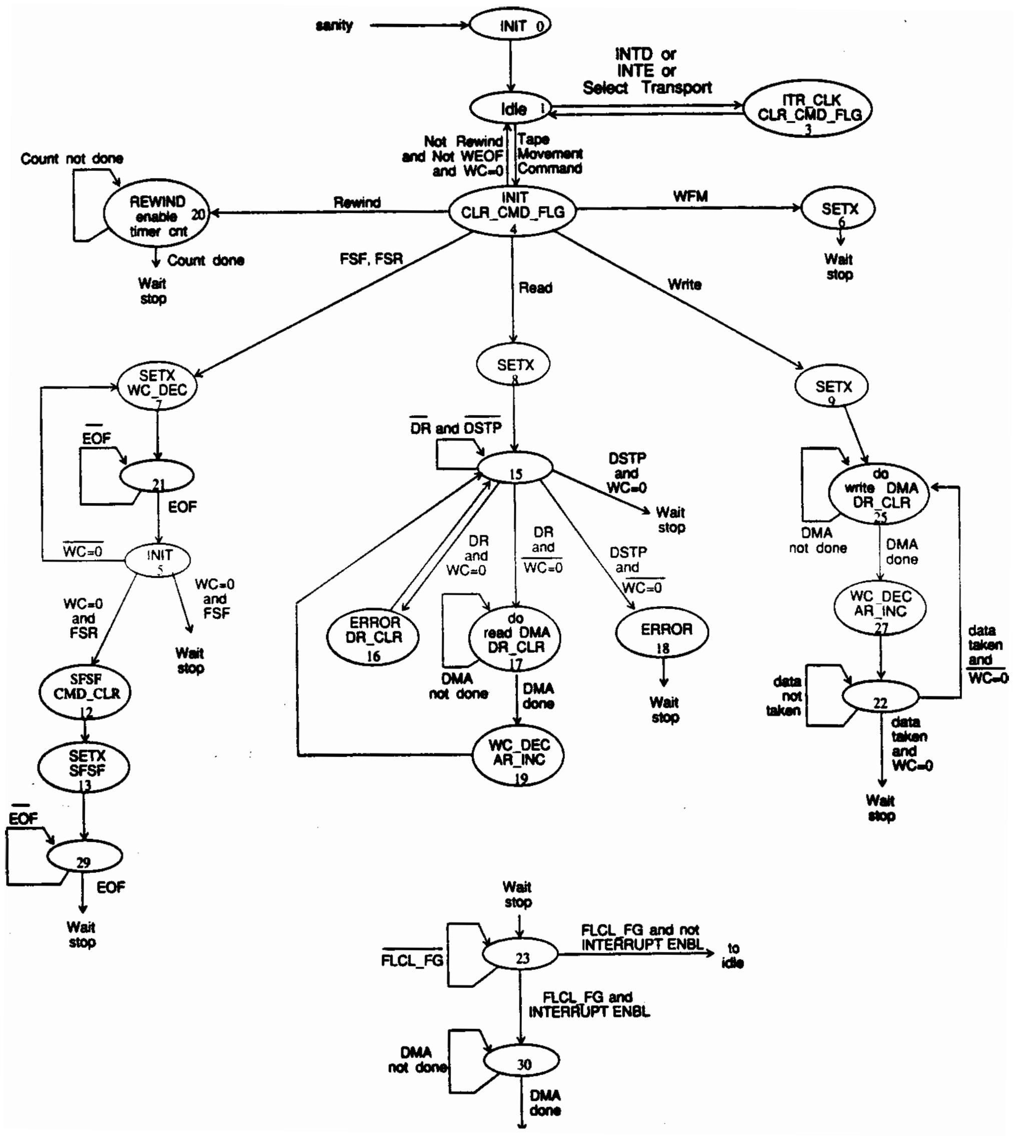


Figure 6.19(a). State Diagram ſa Tape Controller Interface Module.

are generated by the logic associated with the command register. and that the state machine is used only to create the pulses needed.

The remaining commands are file search forward (FSF), file search reverse (FSR), read, and write. All of these commands require a nonzero word count register, so if that condition does not exist in State 4, the state machine returns immediately to the idle state. If, however, the word count register is nonzero, then the action can begin.

The file search commands assert the SETX pulse, decrement the word count, and then wait for the controller to indicate that it has seen an end of file mark (EOF). This is repeated until the word count register is equal to zero. If the specified action was a FSF command, then the desired movement is complete, and the action of the interface module moves to state 23 to wait for the tape movement to stop. If the specified action was a FSR command. then the state machine causes one more file search command. this one in the forward direction. This action leaves the tape at the beginning of a file. rather than at the end of a file.

The write command starts the tape movement and then requests that the DMA state machine perform a DMA transfer to get the information to write onto the tape. Then the address register is incremented and the word count is decremented, and the interface module waits for the controller to take the data. The activity of the data path, while the controller takes the data. is coordinated by pulses from the controller itself, rather than from the tape machine. This maintains synchronization between the devices in the data path and the tape controller. When the information has been taken by the tape controller. the interface module checks to see if more information is needed (is WC equal to zero?). If the transfer is complete. the interface module waits for the tape movement to stop.

The action of the read command is initiated by the SETX pulse, then the interface module waits until data is ready. This condition will exist when the controller has extracted 2 bytes from the tape and placed them into the two registers on the data path. When this has occurred, then a flag is set (data ready, DR) and the appropriate action can be requested by the interface module. If the word count has not reached zero, then a DMA transfer is requested to place the information into memory. This also results in decrementing the word count and incrementing the address register. However, if the word count register has reached zero, then more data is being extracted from the tape than expected. The result here is to not write the information into memory; rather, an error flag is set and the data ignored. When the read action is completed, the controller will send a stop indication (DSTP). If the word count register has not reached zero at this time, then fewer words than expected were received from the tape, and this also causes the error flag to be set.

The final portion of the state machine of the interface module is used to wait for the controller to signal the completion of the tape movement, which is indicated by the FLCL_FG flag. At that time, an interrupt is requested if the interrupt flag is set in the status register. The final action of the state machine is to return to the idle state to await the next instruction from the CPU.

The interaction between the two state machines is handled with a simple flag arrangement, and when a DMA interaction is needed, the DMA_REQ flag is set. The DMA state machine is then enabled to direct the interaction with the UNIBUS. This interaction is shown in Figure 6.19(b).

The DMA state machine remains in the idle state until a DMA transfer is required. It then asserts the bus request signal (BR) to gain access to the bus. When the arbitration system grants access, then the SACK signal is asserted, and

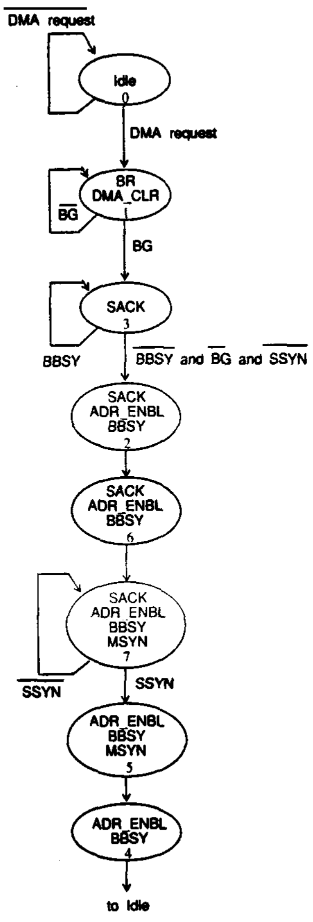


Figure 6.19(b). State Diagram for DMA Bus Interaction.

the state machine waits for the previous bus transfer to complete. When this condition is detected, *then the* transfer is performed: the address is enabled onto the address lines (States 2, 6), the request line is asserted (MSYN), and the system waits for the acknowledge line to be asserted in response (SSYN). When the acknowledge is detected, the DMA state machine returns to the idle state releasing the asserted signals in the appropriate order. Also, the return to the idle state sets a flag that is detected by the state machine of the interface module to indicate that the requested transfer is complete.

If the action is an interrupt sequence rather than a data transfer, then the same action is needed, but not all of the same signals are used. Thus, the appropriate control of the gates and tranceivers in Figure 6.18(e) allows interrupt and DMA transfers to be controlled by the DMA sequencer. For example, the request signal (MSYN) is not used for the interrupt sequence, and hence the bus driver for that signal is disabled during that operation.

The interface module presented here is a relatively straightforward implementation that utilizes the concepts of bus interaction and sequential circuits. The system can be made much more complex in its interaction by including additional instructions and expanding the state diagram. For example. the controller has the capability to read and write when the tape motion is in reverse. This ability can be harnessed by including appropriate instructions in the definition of the interface system, and then including appropriate action definitions in the state machine. Other action, such as block searchs and unloading the tape, an also possible with a more complex system.

This interface system is an example of the application of the techniques discussed in earlier portions of this book. The details of the interface module were determined by a thorough examination of all of the applicable information. The electrical requirements and protocol specifications of the bus used in the system were determined. Also, the electrical requirements and protocol specifications of the tape controller were determined. And the specific action of the programmed I/O instructions of the system was determined. Once this information had been obtained, then a data path block diagram of the system could be generated, and the design of the control system performed. The design required combinational techniques to create many of the signals and conditions that were not tied to the pulses generated by the state machine. Combinational circuits were also applicable in those areas where the sequentiality of action was determined by other systems, such as tilling registers from the bus. Finally. the sequential action of the interface module was defined by state machines and implemented with simple programmable logic devices.

6.6. VLSI Devices for Interface Systems

The example of Section 6.5 included individual TTL devices for every aspect of the system, from address registers to bus controller. However, newer technology has resulted in a variety of devices that place portions of an complete interface module into VLSI devices. The designer of an interface system is then required to ascertain the capabilities of the devices and apply them in a reasonable manner to the systems at hand.

The manufacturers of microprocessor systems have recognized that users of the microprocessors would almost always be desirous of interfacing the microprocessor to physical devices of one kind or another. Thus, they have provided a variety of interface devices to work with their systems. Perhaps one of the first available devices was the 8255, a block diagram of which is shown in Figure 6.20. This device was created to work with the Intel 8080, and has been used not only in 8080 systems, but many other types of systems as well. This device contains logic sufficient for 24 bidirectional lines. The control logic internal to the 8255 specifies the mode of operation for the external lines, whether they an inputs or outputs, and when to accept (supply) the information from *(to)* the bus. The bidirectional data bus lines allow the device to connect directly to buses of a microprocessor system, as shown in Figure 6.21. If the data bus of the microprocessor system is 8 bits wide. then the 8255s are accessed one at a time. If the data

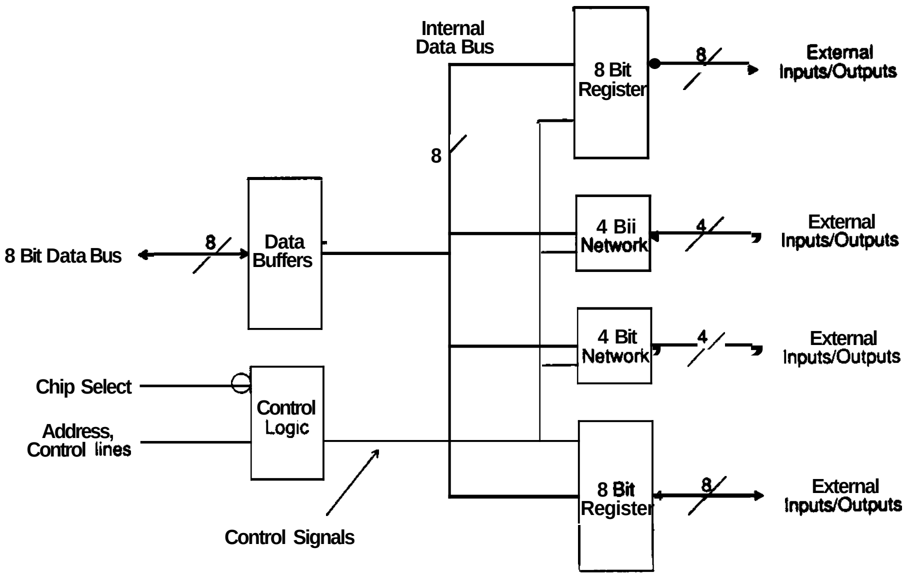


Figure 6.20. Block Diagram of the 8255.

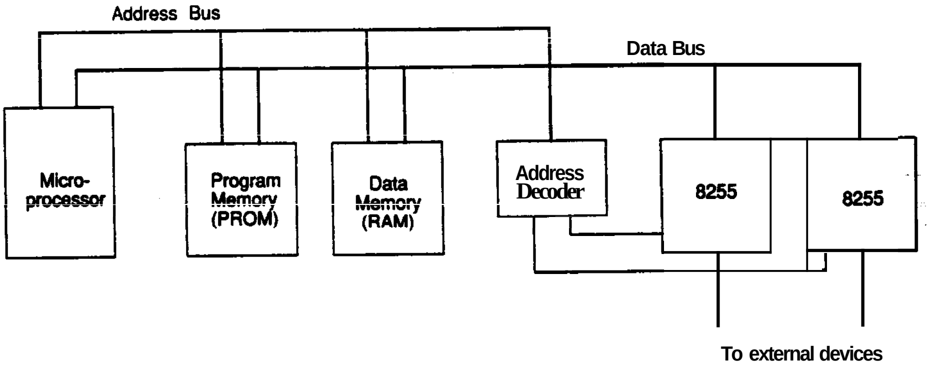


Figure 631. Microprocessor **System** with 8255 Interface Chips.

bus of the system is 16 bits wide, then both 8255s can be accessed **simultaneously.** Or, they could be **byte-addressable** and accessed uniquely. **The** versatility of the device, which allows using the device in any of three basic modes on each of the interface elements, permits configurations that fit the needs of many applications. Howeva, the basic **system** matches the buses discussed **here:** the **address** *decoder* is **responsible** for identifying **when** the devices are to be **accessed,** and the **other timing** signals **control** the actual **transfers.**

One of the functions that is a prime candidate for inclusion in a single integrated circuit is the circuitry required for a DMA operation. Many manufacturers provide controllers for different types of microprocessor systems. A block diagram of the Signetics SBC68438 is shown in Figure 6.22. As indicated in the figure, the chip contains all of the logic needed to perform the DMA operations with a 68000 system bus. This includes registers for storing the word count and the address, as well as interrupt logic, daisy chain priority logic. and isolation gates for the data and address buses. The data bus is also connected to the device controlled by the SCB68430, so that when the DMA controller directs the peripheral device to do so. the data is directed to or extracted from the data bus. Thus, the connection between the DMA controller and the peripheral device allows the peripheral to signal the DMA controller when a bus transfer is needed, and the controller to indicate to the peripheral when the data transfer should take place.

The DMA controller can be used with any peripheral that needs to perform high speed transfers with a 68000 system. Such a system configuration is shown in Figure 6.23. A DMA peripheral device. such as the tape controller of the previous section. is connected to the data bus for transfers of data. and to the SCB60430 to control the data transfers. In addition. the device must be controlled by the processor, and therefore a programmed I/O connection is provided.

The use of DMA controllers in microprocessor systems greatly reduces the number of integrated circuit chips required for controlling peripherals that need the DMA capability. A number of other such devices are available from other manufacturers. Among these are the 8237A from Intel. which is designed to work with X-bit buses and contains logic for four DMA channels. The Am9516, which is available from Advanced Micro Devices, is designed to work with 16-bit

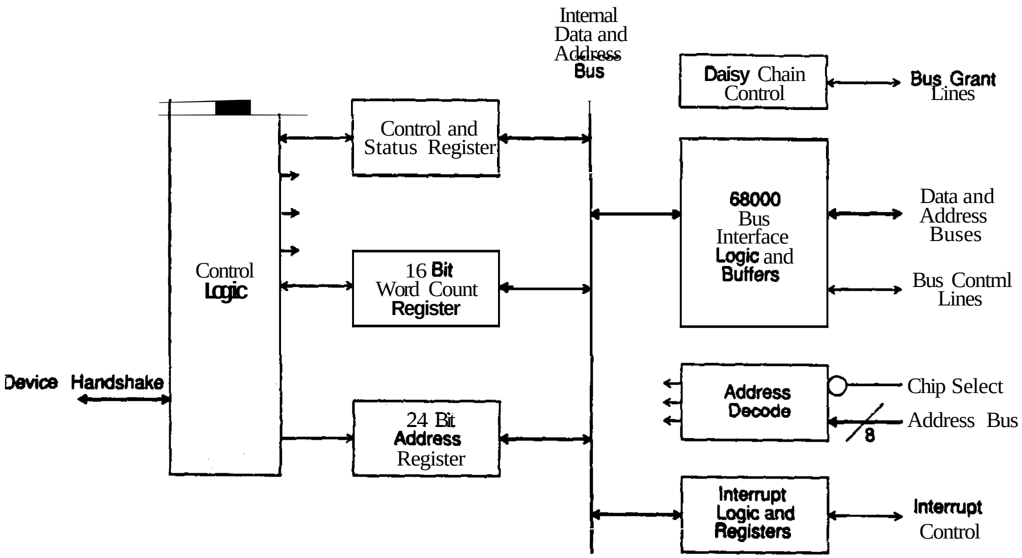


Figure 6.22. Block Diagram of the SCB68430.

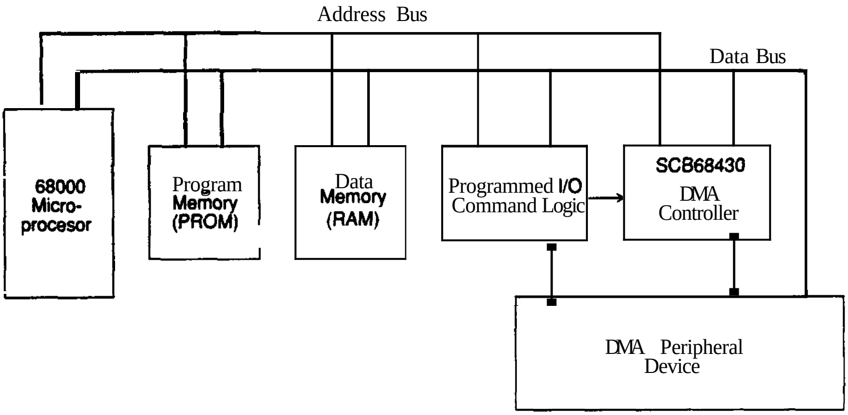


Figure 6.23. Microprocessor System with SCB68430 DMA Controller.

microprocessors, and contains two separate DMA channels. And the NS32203 from National Semiconductor is designed to work with the time multiplexed 32032 bus system, and it contains logic and registers for four separate DMA channels. In each case, the integrated circuit contains a great deal of logic to control the bus transfers needed for DMA action, but the user is required to provide the programmed I/O commands needed to control the action of the DMA peripheral.

Additional capabilities can be added to integrated circuits to further reduce the number of chips required to do particular functions. One such example is the DP8466 Disk Data Controller (DDC) fmm National Semiconductor Corporation. The DDC not only contains the logic needed for DMA operation, but also the logic for providing most of the interface functions to the data stream of a disk system. A basic block diagram of the device is shown in Figure 6.24. Internal to the device are registers that control the DMA action (word count. address, etc), and also registers that control the activity of the serial data stream. In this manner, different types of disk interface specifications can be handled by the same type of device. The bus connection presents a tri-state interface to the system for transfer of both data and address information. And the bus timing circuits allow transfers into the device (e.g., programmed I/O set up of registers) as well as out of the device. The FIFO permits storing of up to 32 bytes of information in the system. This allows data transfers to be performed in a burst mode: once control of the bus is obtained, data can be rapidly transferred to/from memory. The remaining logic is used to perform the functions needed to convert between the serial formats used on a disk and the parallel format of the computer system.

The DDC not onlv has the abilitv to encode and decode the information according to the serial protocols used in disk systems, but it also has capability for certain types of error detection and correction. As interface systems become more complex, one of the functions that must be provided is the ability to detect errors and, under the proper circumstances, correct them. We discussed simpk error detection with parity codes in Chapter 2. as well as error correction with Hamming codes. Serial codes can use parity techniques, but often they also use polynomial codes to provide a different form of error capabilities. With the amount of

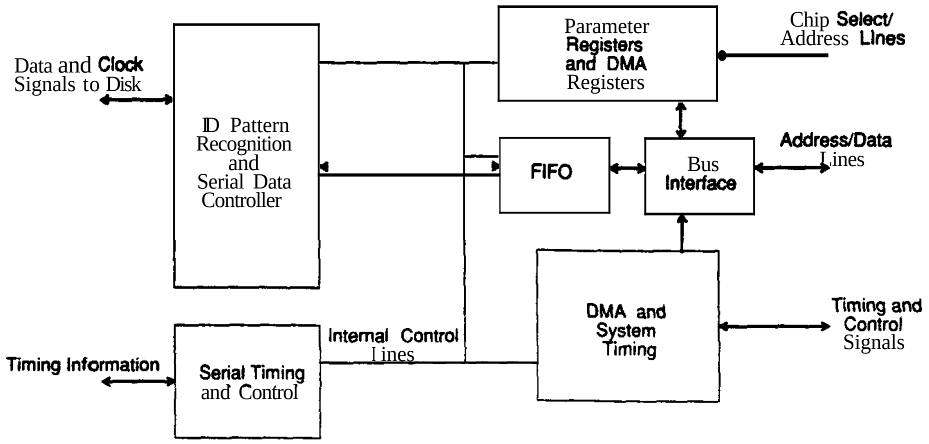

Data and Clock Signals to Disk
ID Pattern Recognition and Serial Data Controller
Parameter Registers and DMA Registers
Chip Select/ Address Lines
FIFO
Bus Interface
Address/Data Lines
DMA and System Timing
Timing and Control Signals
Internal Control Lines
Timing Information
Serial Timing and Control

Figure 6.24. Block diagram of the DP8466.

logic available on integrated circuits, the use of these mechanisms can be included in the chips as shown by the DDC.

As with the DMA controller, the DDC can be used to control data flow in systems, but the control of the disk itself is left up to the user. Consider the block diagram shown in Figure 6.15. Much of the system is identical to the systems shown in earlier figures. The programmed I/O signals are used to control action in the DDC as well as the disk itself. The DP8466 is connected to address and data buses: some buffers, which are not shown in the figure, are required for this connection. The data and control paths to the SMD disk require differential line drivers and receivers, which minimize the effects of noise on the common data lines. For other types of interface specifications, such as the ST506, National also provides a data separator and a data synchronizer. The net effect is to have a family of integrated circuits that connect to general microprocessor bus systems and control disk systems. With this capability, a user can develop a disk system to meet a variety of needs.

We will include one final example of an integrated circuit I/O controller, which is the 7990 Local Area Network Controller for Ethernet (LANCE) of Advanced Micro Devices. Other manufacturers (Intel, National Semiconductor, etc.) have similar Ethernet devices. The LANCE chip connects to a microprocessor system in a manner similar to the other interface systems indicated in this section, as shown in Figure 6.26. The only difference here is that a second chip is required, the serial interface adapter. This chip provides the needed connection for the 7990 to connect to Ethernet systems.

Internal to the LANCE chip a number of functions are performed. A basic block diagram of the device is given in Figure 6.27. Like the other devices we have examined in this section, there is a set of isolating gates to handle the data and address lines of the bus. In addition, a number of registers are included in the system to control the action of the device. These include the normal DMA type of registers, as well as registers that control the Ethernet connection itself. However, the interaction with the memory of the LANCE is more complicated than other

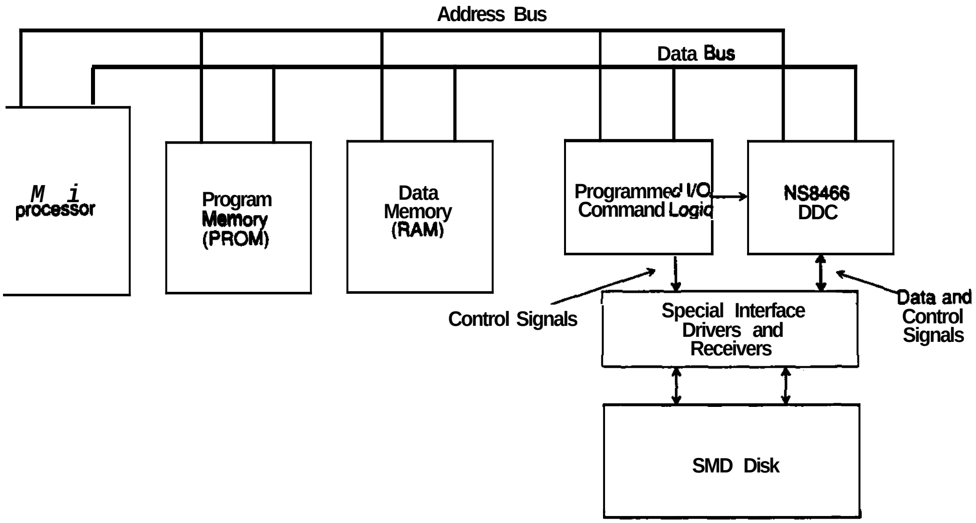


Figure 6.25. Microprocessor System with DP8466 Disk Interface

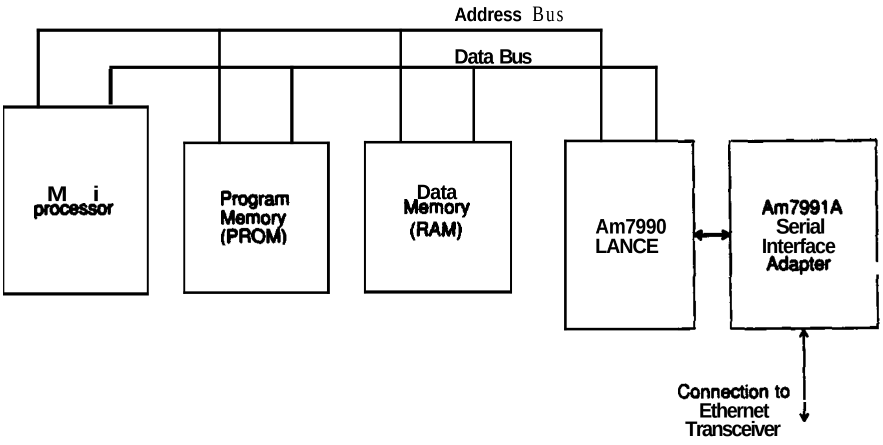


Figure 6.26. Connection of LANCE in Microprocessor System.

systems we have considered. The LANCE operates by both building and examining data structures in the memory areas of the processor. Thus, in addition to transferring data to and from memory, this unit also uses the ability to look at memory to control the activity of the Ethernet d o n .

In addition to the devices described in this section, manufacturers also provide a number of other functions. These include real time clocks for keeping

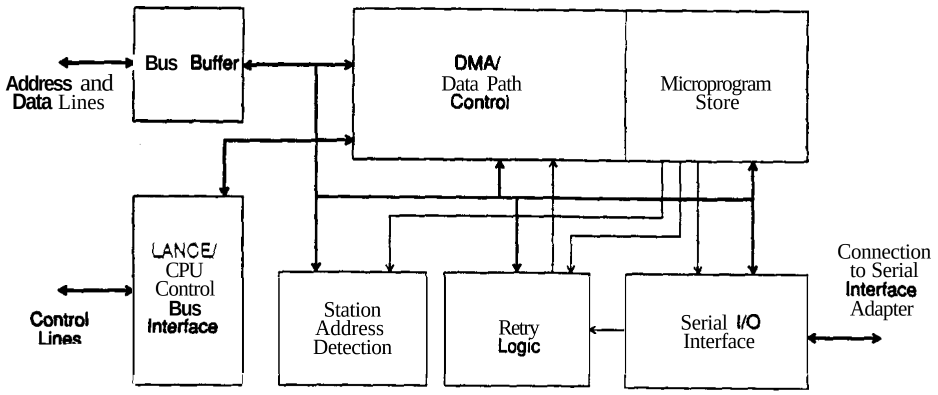


Figure 6.27. Block diagram of the Am7990.

track of the time, event timers to ascertain the time required for internal and external events, serial communications controllers, network interface systems. fiber optic interface modules, and error handling devices, to name a few. In all cases, the user must provide some programmed I/O capability to control some of the basic functions, and the interface unit handles as much of the automatic data movement as feasible. One of the challenges of system architects and designers is to use these devices in reasonable ways in useful systems.

6.7. I/O Channels and I/O Processors

The action of transferring information to and from a computer can take many forms, as we have seen. The I/O mechanisms used as examples have been limited to interaction between a processor and an I/O device connected by a common bus. Indeed, this is the normal connection mechanism for bus-oriented systems used in minicomputers, workstations, and microprocessor systems. Another method of dividing the work of the computer system is to remove from the CPU the responsibility for detailed control of I/O devices, and limit the CPU to computing and controlling. Logically, this resembles the situation depicted by Figure 6.28. The CPU operates normally, executing programs found in main store and manipulating data according to the instructions found there. However, when interaction with an I/O device is required, the CPU requests this interaction by sending a command directly to an I/O device controller. This unit is specifically designed to pmvide control for I/O devices, which it proceeds to do according to the instructions of the CPU. Since the I/O device controller has its own connection to main store, the data transfers occur directly to locations in memory.

The I/O device controller shown in Figure 6.28 is sometimes called a channel, and different types of channels are used in different computer systems. The channel is essentially a special purpose processing element designed to do one thing: control I/O devices. In general, the programs executed by the channel reside in main store, just as the programs executed by the CPU. The CPU indicates to the channel the work to be done by creating programs for the channel to

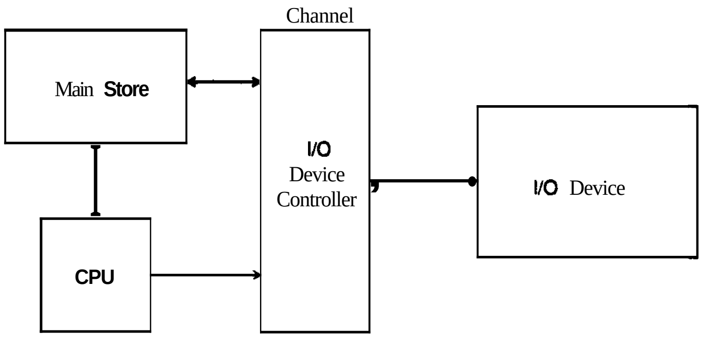


Figure 6.28. Control of I/O Devices with a Channel

execute from the set of operations available to a channel. In some systems these I/O commands are called channel command words (CCWs). After the action is initiated by a direct command from the CPU, the channel will assert the proper signals to cause the transfer of information from the I/O device to the memory. However. the channel has more capabilities than a simple I/O interface module, such as that presented in Section *6.5*. The channel may perform data conversion on data moving in the system. as well as handle error checking and correcting. Also. the channel may interrupt the CPU at any time during the transfer, if the situation requires it. Also. the CPU may request information concerning the status of the transfer at any time. and the channel will respond.

Although many different types of channels are used, channels are sometimes grouped into the classifications used by IBM. With this classification method, channels are grouped into three categories: multiplexer channels, block multiplexer channels, and selector channels. These are shown in Figure 6.29. A multiplexer channel, as its name indicates, multiplexes between a number of I/O devices. Each transfer has associated with it an I/O device address and a byte of information. Each device will have a specific address associated with it in main store, and the multiplexer channel must maintain the correlation between the physical device and its associated storage area in memory. Thus, the multiplexer channel maintains a number of addresses and other information about the physical devices over which it has control. One of the basic requirements for the devices connected to a multiplexer channel is that they are slow enough to allow the channel to switch between them as needed. since they all share the same communication path. Thus, these devices are generally of a nature conducive to the slower speeds: terminals, modems, electromechanical devices, CRTs, and so on.

The selector channel is designed to provide high speed transfers from an external device and the memory of the system. As such, it is very much like a DMA controller: once the system has designated the device to use and the location in memory of the information, the selector channel executes that transfer or control operation before initiating another. This is true even if the operation is merely a track-to-sack seek of a disk or other movement command. However, because of the creation of programs consisting of channel command words in memory, the selector channel may move on to a second transfer as soon as the last data movement of the first transfer has been completed

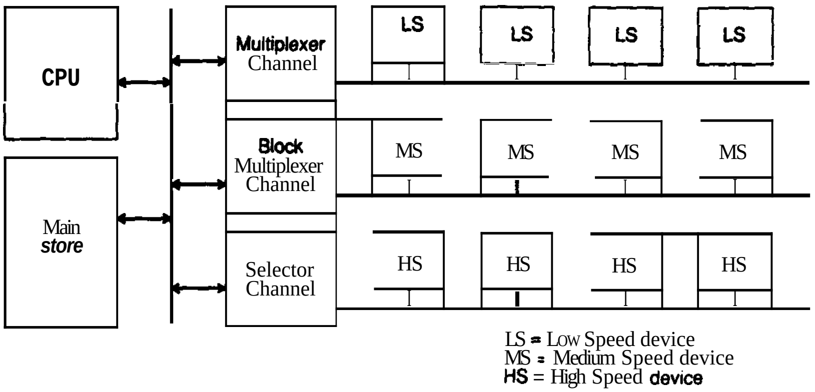


Figure 6.29. Computer System with Multiple Channels.

The block multiplexer channel is designed to have some of the characteristics of both the multiplexer channel and the selector channel. The block multiplexer channel is capable of multiplexing between devices. as the multiplexer channel. but the basic unit of information is no longer a byte. but rather a block of information. Thus. once the transfer of a block of information is started. the channel will maintain the logical connection between the device and its associated location in memory. When the transfer of the block has been completed, then the channel can move on to another device.

A channel provides a mechanism for the processor to off-load the burden of I/O control to a device specifically designed to handle the interaction. The channel controls the interaction with the I/O devices over the channel bus, which is an 8-bit transfer path. The devices that connect to the channel bus have the same problem examined earlier in the chapter: transfers are made over a shared data path, and the interface modules must be designed to permit this to happen in a uniform manner. However, the interface problem is somewhat simplified, since the channel is always in control of the bus. Once the channel action has been initiated, no further action is required on the part of the CPU until the transfer is complete. This leaves the processor free more of the time to do what it does best: compute.

When a computer system is configured with a number of channels, the system architect includes a sufficient number of channels to provide the I/O capability needed by the system. The transfer rate of the memory systems used in large computer systems is sufficient to allow several channel systems to operate simultaneously. Therefore, the architect is free to utilize enough channels to meet the maximum transfer rate required, or to use a small number of channels to provide capability at a minimal cost.

Channels are one example of an input/output processor (IOP). Figure 6.30 shows a system configured with a number of processing elements and IOPs. The figure indicates that the IOPs are dedicated to specific functions, such as disk or tape systems. This need not necessarily be the case. The basic requirement for an IOP is that it be capable of controlling a device and interfacing to another system. Thus, the IOPs shown in the figure each perform a designated task, and present the

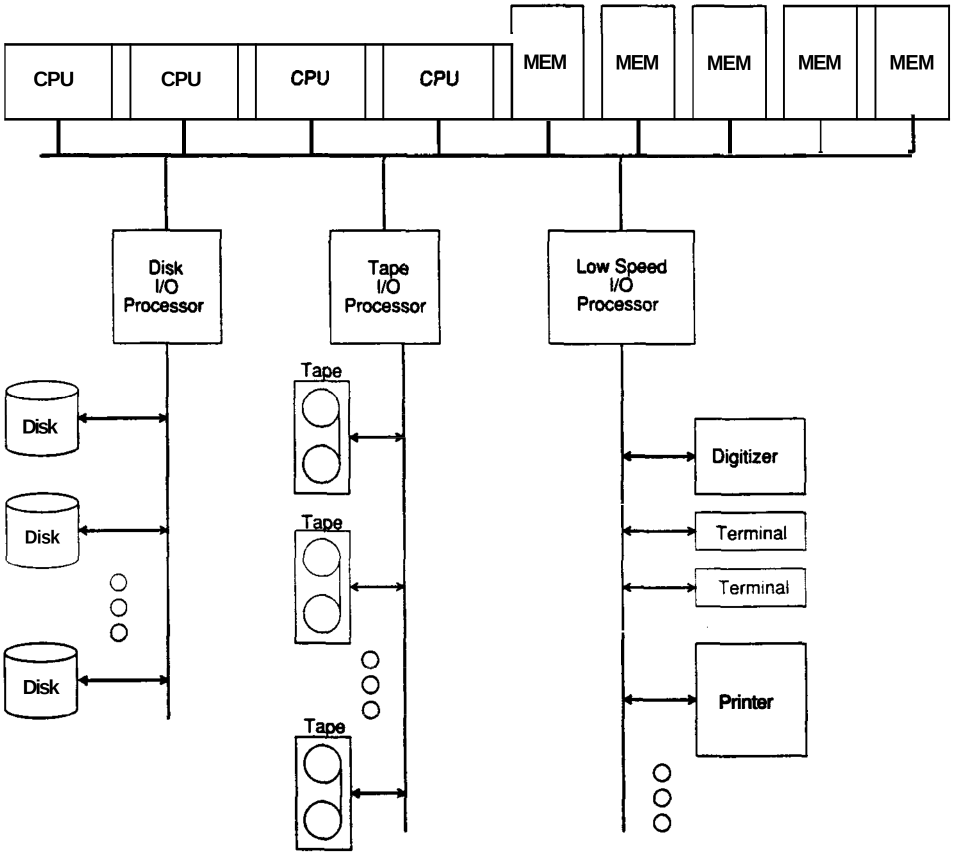
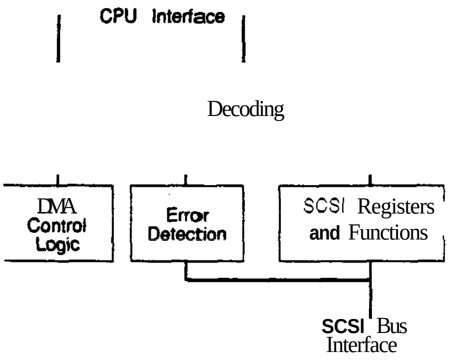


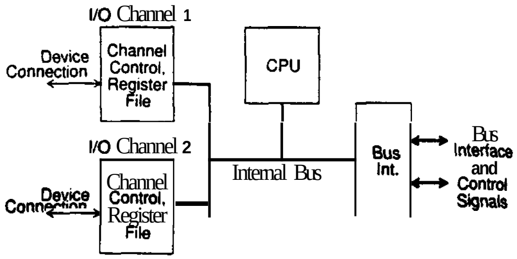
Figure 6.30. System with Multiple Processors and IOPs.

results to the larger computer system. In this context, many of the units described above have the characteristics of IOPs.

Additional systems that fall into this classification include the Am5380 SCSI (Small Computer Systems Interface) Interface Controller, made by Advanced Micro Devices, and the 8089 I/O processor, made by Intel. Block diagrams of these systems are shown in Figure 6.31. Also included in Figure 6.31 is a diagram of the 8044 remote universal peripheral interface.

The SCSI interface definition provides an 8-bit data path to peripherals. and a number of disks and tape units have been designed to be connected to computer systems by using this protocol. To the controlling CPU, the Am5380 appears as a set of eight registers; these could be located in the memory space as memory mapped W or in a separate W space. The controlling CPU monitors activity on the SCSI bus and requests appropriate action by reading or writing to these




I/O Channel 1
Device Connection
Channel Control, Register File
I/O Channel 2
Device Connection
Channel Control, Register File
CPU
Internal Bus
Bus Int.
Bus Interface and Control Signals

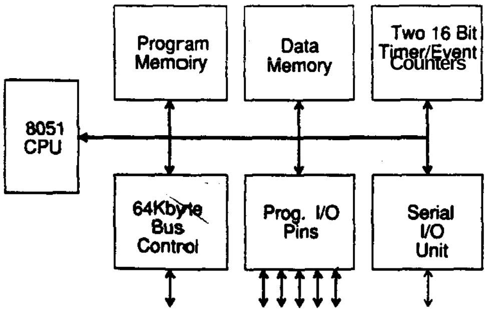


Figure 631. IOPs: (a) SCSI Interface Controller; (b) I/O Processor, and; (c) Remote Universal **Peripheral Interface.**

registers. This device must be utilized in conjunction with other devices to perform the DMA transfers required for high speed operation. In this type of a configuration, the Am5380 is used to provide the SCSI bus connection. and the other portion of the circuitry controls DMA interaction with the host. To control a number of SCSI transfers simultaneously. a system could be configured with several Am5380 devices. Each of these units would be capable of transferring information directly into the memory system under DMA control. The Am5380 can also be utilized in the design of peripheral units that connect to a SCSI bus, as it *can* be a target as well as an initiator on the bus.

The 8089 **I/O processor** is a device that contains a microprocessor capable of controlling interaction with two I/O devices. The unit is compatible with 8086- and 8088-type microprocessors, and provides high speed DMA capabilities for two **separate devices.** It is designed to intelligently control transceivers connecting I/O devices to a microprocessor system. The system was created to be utilized in the memory space of the host CPU; communication between host and I/O processor are accomplished by passing messages in the memory space. The instruction set of the 8089 has been created to function efficiently in its role as data mover, and the instructions include a number of load, store, and move capabilities, as well as conditional and unconditional branches, and minimal arithmetic capabilities. Devices of this nature can be used to remove from the host CPU some of the mundane action needed for I/O transfers, allowing the CPU to concentrate on the computational aspects of the system.

In addition to LSI devices. such as the Am5380 and the 8089 which control bus interaction. other devices are available to provide lines that can be connected directly to the control lines of I/O units to control the interaction. One such unit is the 8044 remote universal peripheral interface. also shown in Figure 6.31. This unit contains an 8051 CPU capable of asserting lines needed for control of I/O devices. The 8044 provides 24 programmable pins, so that a designer could create signals for controlling the action of peripheral devices, and interfacing those devices to a processor system.

Channels and IOPs provide a mechanism whereby a system can divide the tasks that are required — computation and communication — between processors that are more appropriately configured to the task. Moving I/O-oriented tasks to separate. specialized processors has two immediate benetits. First. the transfers required by I/O units are in general much slower than memory transfers, since limits are imposed by the electrical and mechanical nature of the I/O systems. This means that the IOPs can be constructed with medium speed technology and devices. The second benefit is the release of the time commitment from the CPU, since it no longer has primary responsibility for every command given to I/O devices. This allows the apparent system speed to increase.

6.8. Conclusion

The communication mechanism between the processing element and the external world is a very important pan of any computer system. By this mechanism data is obtained by the CPU for use within the system, and results of the operations are made available to peripheral units, whether those units are computer systems, or disks, tapes or other peripheral devices.

The I/O mechanisms are an important part of the functionings of a computer system. To assess the impact of the I/O system. a thorough analysis of the system should be performed. This will allow evaluation of alternative utilizations of the busing schemes and other I/O mechanisms. matching the interconnection features with the characteristics of the processor(s) and peripherals.

Busing systems allow different modules to communicate with one another over the common communication medium. Asynchronous bus communication protocols allow the transfers to proceed. controlled by signals generated by both sender and receiver. This allows the transaction to seek its natural transfer rate for the bus. Asynchronous mechanisms can be used with buses that have separate address and data lines, as well as buses which time multiplex data and address on the same set of lines.

Information can also be transferred on a bus in a synchronous manner. The protocols for synchronous bus systems allow multiple operations, such as arbitration, transfer, and acknowledge, to occur simultaneously. For this reason, the data rates for synchronous bus systems is generally higher than a rate for an asynchronous bus.

The task of identifying the controller of a bus ~~system is the~~ responsibility of an arbitration system. The arbitration mechanism can be parallel in nature, which allows for high speed arbitration based on algorithms of arbitrary complexity. Another arbitration mechanism is serial in nature, with each module cooperating by passing a grant signal if access to the bus is not required. This method is necessarily slower than the parallel system, since decisions are made in a serial fashion. Another arbitration mechanism is polling, which is not used for bus ownership recognition. but is used in identifying active I/O devices.

Control of activity of peripheral devices is achieved by specialized I/O instructions, or by using memory mapped I/O techniques. By using instructions that control the action of peripheral devices. a processor can initiate transfers and monitor the status of the system. The complexity of the interface module between the processor and the peripheral units determines the responsibility of the CPU. If minimal capability exists within the interface module, then the CPU must monitor the status of the peripheral and cause all action with programmed I/O instructions. If the interface module is capable of interrupting the processor, then the CPU can continue processing and service the I/O device only when action is needed. Finally, if the interface module contains the ability to interact directly with the memory, then the CPU can initiate a transfer and he interrupted only when the action is complete. This direct memory access minimizes the time required by the CPU for controlling I/O functions.

Channels and I/O processors are specialized processing elements designed to remove the elemental I/O concerns from the CPU. These processors directly control peripheral elements to perform the data transfers and other functions required of I/O devices. With the byte multiplexing technique, the channel switches between I/O devices as needed and tags each byte as it is obtained. This allows many slow speed devices to be attached to a single channel. A block multiplexer channel operates on a similar principle, but the units of transfer are blocks of data rather than bytes. A selector channel selects one I/O device, and transfers data at high rates to or from that device before being switched to a different peripheral unit.

All of the techniques mentioned above — bus systems, arbitration systems, programmed I/O mechanisms. direct memory access, interrupts, channels, and IOPs — are utilized to transfer information to and from a computer system. By using the various mechanisms as called for by the peripheral devices, computer systems, and desired data rates, an effective processing system can be configured that will not only compute, but will also make available the results of the computations.

6.9. Problems

6.1 For a bus with handshake protocol shown in Figure 6.3, design a byte swap register that functions at address 776504_8. That is, writing to the specified address will fill a register, and reading from that same location will present the data in a byte swapped manner, with the data written on the most significant eight lines now available on the least significant lines, and vice versa The system has an **18-bit** address bus and a **16-bit data** bus.

6.2 Design a hardware multiplier that will operate on an asynchronous bus system with a 24-bit address bus and a **16-bit** data bus. The multiplier must use the shift and add algorithm shown in **Figure** 3.12 (and in Appendix B). The unit must respond to the following **addresses** on the bus (read and write are from point of view of CPU):

Address	*Read Action*	*Write Action*
777640_8	Read from multiplier register.	Write to multiplier register.
777642_8	Read from multiplicand register.	Write to multiplicand register.
777644_8	Read 16 least significant bits of **result.**	No action.
777646_8	Read 16 most significant **bits** of result.	No action.

This multiplier will function for positive numbers only. The interaction with the multiplier and multiplicand registers can be accomplished by using combinational circuits to interact with the control signals of the bus. When the location of the least significant bits is accessed, a sequential controller should **perform** the multiply on the data in the input registers, and present the result when the multiplication process is finished. When the most significant result location is accessed. the bits in the most significant bits of the product register should be made available. without going through another multiplication process.

6 3 One of the operations that proves to be very beneficial in the algorithm known as the fast **fourier** transform (FFT) is a bit reversal, **where** the most and least significant bits are exchanged, the second most and the second least significant bits are exchanged, and so on. Design a bit reversal register operating on a bus that **uses** the time multiplexed **asynchronous** protocol of Figure 6.5. When the **address** 17777605_8 is written to, a register is filled. When that same location is read, the bits in the register are presented to the bus in bit reversed *order.* The bus lines involved are multiplexed **between** a **22-bit** address and a **16-bit** data value. The number of bits needed for the reversal operation depends on the size of the FFT. What modifications would be needed to allow a different number of bits to be involved in the bit **reversal? That** is, what changes in the design would be **required** in the definition of the unit, and what logic complications would result?

6.4 Design a bit rotator for a time multiplexed asynchronous bus that operates **according** to the **protocol** shown in Figure 6.5. This bus multiplexes the common lines **between** a 22-bit **address** and a **16-bit** data value. **The rotator works** at the following addresses:

Address	*Read Action*	*Write Action*
17777642_8	**Read rotate value.**	**Write rotate** value.
17777644_8	**Read** position value.	write **position** value.
17777646_8	**Read value in rotate register rotated left** number of bit positions **specified** by position value.	**No action.**

The rotator has two registers: a 16-bit rotate register, which contains the value to be rotated, and a 4-bit position register, which identifies how many bits (to the left) to mtate the value Located in the rotate register. The position reading and writing the mtate register and the position register simply involve the bits in the registers in question. When a rotated value is requested, then the value in the rotate register is loaded into a separate shift register, which is configured as a mtator, and this register is rotated the amount specified by the four bits of the position register. When the rotate has been completed, the value is supplied to the bus, and the transaction can terminate.

6.5 Three types of arbitration mechanisms are discussed in this chapter. Give a brief description of each of the mechanisms, along with an explanation of what are the good characteristics (and why) and bad characteristics (and why) of each mechanism.

6.6 Design a parallel arbitration mechanism for eight master modules that operates on the round robin principle. That is, once a master module has been granted access to the bus, the module with the highest priority for the next bus grant is the module with the next highest number (mod 8).

6.7 Both synchronous and asynchronous bus communication pmtocols are discussed in this chapter. Identify the salient characteristics of each type of protocol. and describe the good and bad features of each. Which communication mechanism is faster? Why?

6.8 An interface is to he designed to control a data logger and provide the data to a computer. The computer is organized around an asynchronous data bus, such as the UNIBUS or MULTIBUS. Give a simple block diagram of the interface, identifying the major data paths/registers and the principal contml boxes. What information is transferred under program control? What is transferred on a cycle stealing basis?

6.9 Consider a computer system with the following characteristics:

500 nsec memory cycle time, both read and write

2 microsec instruction time for all instructions (very strange computer, since most clear instructions take less time than multiplies, or a CALL)

memory mapped I/O

standard instruction set

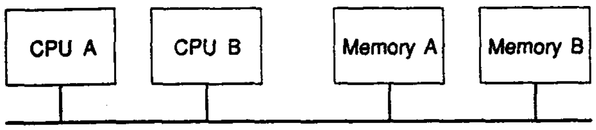
Create appropriate code segments to control I/O transfers, and determine the peak transfer rate and the average transfer rate for blocks of 512 words, for

a. interrupt driven I/O

b. straight programmed I/O

c. DMA

6.10 Consider the partial system diagram shown below, which contains two CPUs and two memories. Using the synchronous protocol of Figure 6.11, what is the shortest amount of time in which the two CPUs could write two words (each) to the memories, where CPU A is writing to MEM A, and CPU B is writing to MEM B. Plot the timing relationship of the transactions, showing



Synchmnous Bus System (arbitration lines, data and address lines, acknowledge lines)

Figure P6.10. Partial System Diagram with Synchronous Bus.

the relationship between the arbitration lines, data/address lines. and the acknowledge lines. If the words are each 4 bytes, and the cycle time is 100 nsec, what is the data rate?

6.11 Repeat Problem 6.10 for read cycles, instead of write cycles. Assume that the bus cycle time is 100 nsec. and that the memory modules require five cycles to obtain the requested data.

6.12 Create a diagram similar to Figure 6.12 that explains the time relationship between commanders and responders on the SBI. Show with the diagram both reads and writes, and configure the transactions to demonstrate a maximum transfer rate. Name three different techniques that could be used to increase the bandwidth of the bus.

6.13 An A/D converter is configured as shown in Figure P6.13. Design a DMA interface that will input data to an asynchronous bus system from this A/D convener. The definition of the behavior of the interface is as follows: The interface is idle until a bit (the GO bit. which is the most significant bit of the command register) is set in the command register. When the GO bit is set, the interface will clear the DATA_AVAILABLE-H flag and wait for the A/D convener to generate a new data sample. When the sample is available, the interface will request a bus transaction by asserting a bus request line (BR-H). When the bus grant is asserted (BG-H), then the interface releases the bus request and asserts the address and data lines. After a 50 nsec delay, the interface asserts the request line (REQ-H) and awaits the assertion of the acknowledge line (ACK-H). The master then releases the request line, and when the master sees that the acknowledge line has. been released, it will release the address lines and decrement a word count register. This process

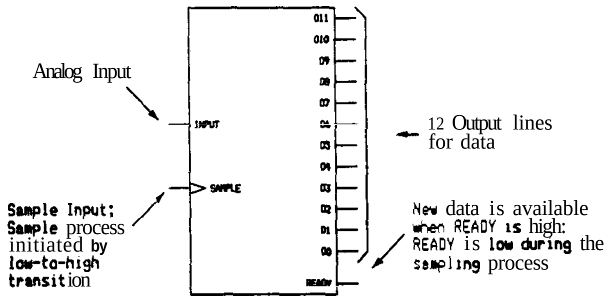

Analog Input
INPUT
SAMPLE
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
READY
12 Output lines for data
Sample Input; Sample process initiated by low-to-high transition
New data is available when READY is high: READY is low during the sampling process

Figure P6.13. A/D Convener for Problem 6.13.

continues until the word count reaches zero. The interface contains a status register readable by the CPU, which consists simply of a busy bit (which is the most significant bit on the bus). When the interface is in the process of transferring information to the computer the bit is a one; otherwise. the bit is a zero. Thus, the busy bit will be set by the action of setting the GO bit of the command register, and reset when the word count reaches zero, and the transfer process is completed For this system, provide:

a. a programmer's interface definition: the registers that a programmer can reach and their definitions

b. a data path block diagram that identifies all of the necessary components for the data transfers and their interconnection

c. a set of control signals that can be used to control the interface (the control signals of the data path \

d. a state diagram of the interaction

e. logic diagrams for the system

6.14 The bus transactions described in Problem 6.13 are used to transfer information to a memory module. Modify the protocol description to read information from a memory. Then design a D/A converter interface with the following behavior: When a SEND-DATA flag is set, the interface extracts a value from the location identified by the address register and sends it to the D/A convener. and also resets the tlag and decrements the word count. When the word count re aches zero, the word count is returned to an initial word count value and the address register is returned to an initial address value, and the process is repeated. This system could be used to draw waveforms on an oscilloscope. For this system give:

a. a programmer's interface definition: the registers that a programmer can reach and their definitions. Note that the programmer will not be able to reach all of the registers in the system.

b. a data path block diagram that identifies all of the necessary components for the data transfers and their interconnection

c. a set of control signals that can be used to control the interface (the control signals of the data path

d. a state diagram of the interaction

e. logic diagrams for the system

6.15 Obtain the data manual for a DP8466 disk data controller (DDC), and using that device design a disk interface for a 16-bit asynchronous bus. initiate the design by identifying the possible transfers between the DDC and the bus. Then identify the additional signals required to control the action of the disk. Then, complete the design process by creating an appropriate data path block diagram, specifying the control system, and creating the appropriate logic diagrams.

6.16 Obtain a data manual for an Am9516 direct memory access controller (DMAC), and using a pair of devices design a high speed communication channel that connects two 16-bit asynchronous bus systems. These are essentially independent computer systems that are physically close together, and information is to be exchanged between the two over the DMA channels.

6.10. References and Readings

[AMD85] Advanced Micro Devices, *Bipolar Microprocessor Logic and Interface Data Book.* Sunnyvale, CA: Advanced Micro Devices, 1985.

[Baer84] Baer, J. L., "Computer Architecture," *Computer.* Vol. 17. No. 10, October 1984, pp. 77–87.

[Baer80] Baer. J. L., *Computer System Architecture.* Rockville, MD: Computer Science Press, 1980.

[Bart85] Bartee, T. C., *Digital Computer Fundamentals. 6th* edition, New York: McGraw Hill Book Company, 1985.

[BeNe71] Bell, C. G. and A. Newell, *Computer Structures: Readings and Examples.* New York: McGraw Hill Book Company. 1971.

[Chen74] Chen. R. C. H. "Bus Communications Systems," Ph.D. Dissertation. Pittsburg, PA: Department of Computer Science, Carnegie-Mellon University, 1974.

[Clui82] Cluley, J. C., *Minicomputer and Microprocessor Interfacing,* New York: Crane, Russak, 1982.

[DEC82] Digital Equipment Corporation, *VAX Hardware Handbook.* Maynard, MA: Digital Equipment Corporation, 1982.

[Dext86] Dexter. A. L., *Microcomputer Bus Structures and Bus Interface Design.* New York: M. Dekker. 1986.

[Egge83] Eggebrecht. L. C.. *Interfacing to the IBM Personal Computer.* Indianapolis. IN: H. W. Sams, 1983.

[Flet80] Fletcher, W. I., *An Engineering Approach to Digital Design.* Englewood Cliffs, NJ: Rntice Hall, 1980.

[IEEE75] Institute of Electrical and Electronics Engineers, "IEEE Standard Digital Interface for Programmable Instrumentation," IEEE Std. 488-1975. The Institute of Electrical and Electronics Engineers, Inc., October 1975.

Intel, *Microsystem Components Handbook.* Intel Corporation, 1984.

[LaZa84] Lazowska, E. D., J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance.* Englewood Cliffs. NJ: Prentice Hall. 1984.

[Lang82] Langdon, G. G., Jr., *Computer Design.* San Jose, CA: Computeach Press Inc, 1982.

[Lipo88] Lipovski, G. J., *Single- and Multiple-Chip Microcomputer Interfacing.* Englewood Cliffs, NJ: Rntice Hall, 1988.

[Mati80] Matick, R. E., "Memory and Storage," in [Ston80], pp. 205–274.

[Mati77] Matick, R. E, *Computer Storage Systems and Technology.* New York: John Wiley & Sons, 1977.

[Moto85] Motorola, *The VMEbus Specification.* 1985.

[PaWa81] Parker, A. C., and J. J. Wallace, "An I/O Hardware Descriptive Language," IEEE *Transactions on Computers.* Vol. C-30, No. 6 June 1981, pp. 423–439.

[Poll83] Pollard, L. H, "Fault Tolerant Bus Communication Protocols for Computer Systems," Ph. D. Dissertation. Champaign-Urbana, IL: University of Illinois, 1983.

[Shiv85] Shiva, S. G.. *Computer Design and Architecture.* Boston, MA: Little, Brown, 1985.

[SiBe82] Siewiorek, D. P., C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples*. New York: McGraw Hill Book Company, 1982.

[Stal87] Stallings, W., *Computer Organization and Architecture*. New York: Macmillan Publishing Co., 1987.

[TI85] Texas Instruments, The *TTL Data* Book. Volume 2. Dallas, TX: Texas Instruments, 1985.

[ThJe72] Thurber, K. J., E. D. Jensen, et al., "A Systematic Approach to the Design of Digital Bussing Structures," *AFIPS Conference Proceedings — Fall Joint Computer Conference*. 1972, pp. 719–740.

[ThMa79] Thurber, K. J., and G. M. Masson, "Bus Structures," in Distributed Processor Communication Architecture, Lexington. MA: Lexington Books, 1979. pp. 131–174.

[TiLa82] Titus. C. A. J. A. Titus, and D. G. Larson, *STD Bus Interfacing*. Indianapolis. IN: H. W. Sams, 1982.

[TsSi82] Tseng, C. L. and D. P. Siewiorek. "The Modeling and Synthesis of Bus Systems," Technical Report DRC-18-42-82, Design Research Center. Pittsburg. PA: Carnegie-Mellon University, 1982.

[Wilk87] Wilkinson, B., *Digital System Design*. Englewood Cliffs, NJ: Prentice Hall International. 1987.