complete ($t = 35$) is the printer ISR resumed. When that routine completes ($t = 40$), control finally returns to the user program.

I/O Function

Thus far, we have discussed the operation of the computer as controlled by the processor, and we have looked primarily at the interaction of processor and memory. The discussion has only alluded to the role of the I/O component. This role is discussed in detail in Chapter 7, but a brief summary is in order here.

An I/O module (e.g., a disk controller) can exchange data directly with the processor. Just as the processor can initiate a read or write with memory, designating the address of a specific location, the processor can also read data from or write data to an I/O module. In this latter case, the processor identifies a specific device that is controlled by a particular I/O module. Thus, an instruction sequence similar in form to that of Figure 3.5 could occur, with I/O instructions rather than memory-referencing instructions.

In some cases, it is desirable to allow I/O exchanges to occur directly with memory. In such a case, the processor grants to an I/O module the authority to read from or write to memory, so that the I/O-memory transfer can occur without tying up the processor. During such a transfer, the I/O module issues read or write commands to memory, relieving the processor of responsibility for the exchange. This operation is known as direct memory access (DMA) and is examined Chapter 7.

3.3 INTERCONNECTION STRUCTURES

A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other. In effect, a computer is a network of basic modules. Thus, there must be paths for connecting the modules.

The collection of paths connecting the various modules is called the *interconnection structure*. The design of this structure will depend on the exchanges that must be made among modules.

Figure 3.15 suggests the types of exchanges that are needed by indicating the major forms of input and output for each module type:[2]

- **Memory:** Typically, a memory module will consist of N words of equal length. Each word is assigned a unique numerical address $(0, 1, \ldots, N-1)$. A word of data can be read from or written into the memory. The nature of the operation is indicated by read and write control signals. The location for the operation is specified by an address.
- **I/O module:** From an internal (to the computer system) point of view, I/O is functionally similar to memory. There are two operations, read and write. Further, an I/O module may control more than one external device. We can refer to each of the interfaces to an external device as a *port* and give each a unique address (e.g., $0, 1, \ldots, M-1$). In addition, there are external data paths for the

[2]The wide arrows represent multiple signal lines carrying multiple bits of information in parallel. Each narrow arrows represents a single signal line.

Scanned by CamScanner

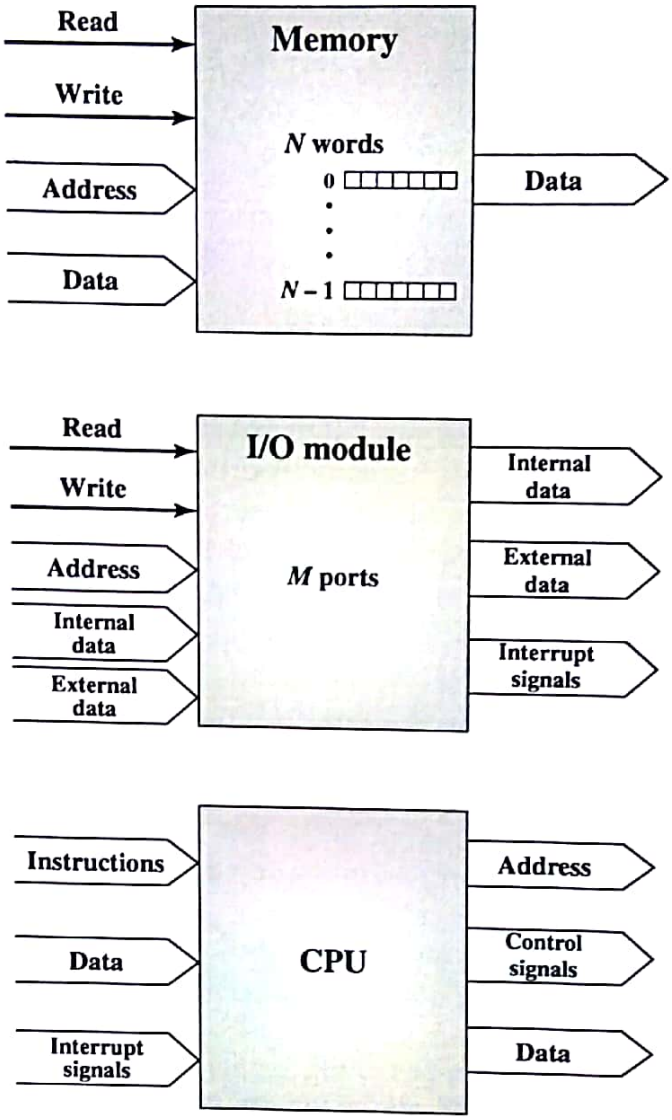


Figure 3.15 Computer Modules

input and output of data with an external device. Finally, an I/O module may be able to send interrupt signals to the processor.

- **Processor:** The processor reads in instructions and data, writes out data after processing, and uses control signals to control the overall operation of the system. It also receives interrupt signals.

The preceding list defines the data to be exchanged. The interconnection structure must support the following types of transfers:

- **Memory to processor:** The processor reads an instruction or a unit of data from memory.
- **Processor to memory:** The processor writes a unit of data to memory.
- **I/O to processor:** The processor reads data from an I/O device via an I/O module.
- **Processor to I/O:** The processor sends data to the I/O device.
- **I/O to or from memory:** For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

Scanned by CamScanner

Over the years, a number of interconnection structures have been tried. By far the most common is the bus and various multiple-bus structures. The remainder of this chapter is devoted to an assessment of bus structures.

3.4 BUS INTERCONNECTION

A bus is a communication pathway connecting two or more devices. A key characteristic of a bus is that it is a shared transmission medium. Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus. If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, only one device at a time can successfully transmit.

Typically, a bus consists of multiple communication pathways, or lines. Each line is capable of transmitting signals representing binary 1 and binary 0. Over time, a sequence of binary digits can be transmitted across a single line. Taken together, several lines of a bus can be used to transmit binary digits simultaneously (in parallel). For example, an 8-bit unit of data can be transmitted over eight bus lines.

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy. A bus that connects major computer components (processor, memory, I/O) is called a *system bus*. The most common computer interconnection structures are based on the use of one or more system buses.

Bus Structure

A system bus consists, typically, of from about 50 to hundreds of separate lines. Each line is assigned a particular meaning or function. Although there are many different bus designs, on any bus the lines can be classified into three functional groups (Figure 3.16): data, address, and control lines. In addition, there may be power distribution lines that supply power to the attached modules.

The **data lines** provide a path for moving data among system modules. These lines, collectively, are called the *data bus*. The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the *width* of the data bus. Because each line can carry only 1 bit at a time, the number of lines determines how many bits can be transferred at a time. The width of the data bus is a key

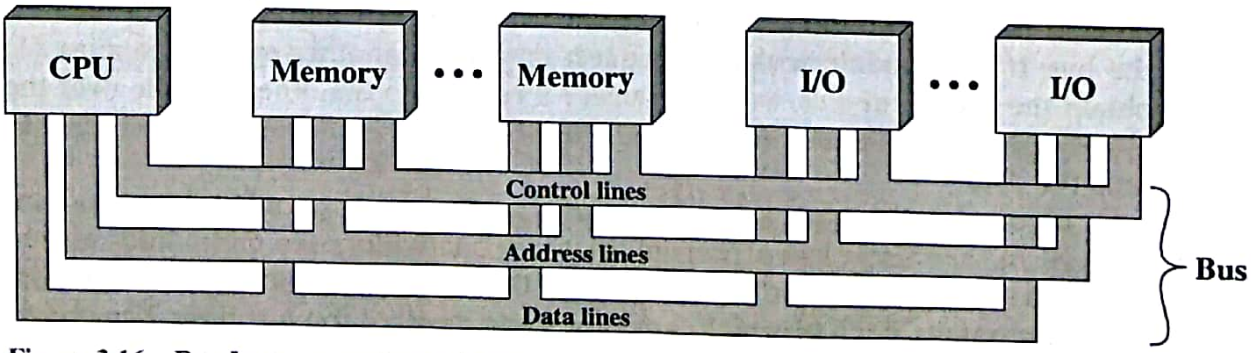


Figure 3.16 Bus Interconnection Scheme

Scanned by CamScanner

factor in determining overall system performance. For example, if the data bus is 32 bits wide and each instruction is 64 bits long, then the processor must access the memory module twice during each instruction cycle.

The **address lines** are used to designate the source or destination of the data on the data bus. For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines. Clearly, the width of the address bus determines the maximum possible memory capacity of the system. Furthermore, the address lines are generally also used to address I/O ports. Typically, the higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port within the module. For example, on an 8-bit address bus, address 01111111 and below might reference locations in a memory module (module 0) with 128 words of memory, and address 10000000 and above refer to devices attached to an I/O module (module 1).

The **control lines** are used to control the access to and the use of the data and address lines. Because the data and address lines are shared by all components, there must be a means of controlling their use. Control signals transmit both command and timing information among system modules. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Typical control lines include

- **Memory write:** Causes data on the bus to be written into the addressed location
- **Memory read:** Causes data from the addressed location to be placed on the bus
- **I/O write:** Causes data on the bus to be output to the addressed I/O port
- **I/O read:** Causes data from the addressed I/O port to be placed on the bus
- **Transfer ACK:** Indicates that data have been accepted from or placed on the bus
- **Bus request:** Indicates that a module needs to gain control of the bus
- **Bus grant:** Indicates that a requesting module has been granted control of the bus
- **Interrupt request:** Indicates that an interrupt is pending
- **Interrupt ACK:** Acknowledges that the pending interrupt has been recognized
- **Clock:** Is used to synchronize operations
- **Reset:** Initializes all modules

The operation of the bus is as follows. If one module wishes to send data to another, it must do two things: (1) obtain the use of the bus, and (2) transfer data via the bus. If one module wishes to request data from another module, it must (1) obtain the use of the bus, and (2) transfer a request to the other module over the appropriate control and address lines. It must then wait for that second module to send the data.

Physically, the system bus is actually a number of parallel electrical conductors. In the classic bus arrangement, these conductors are metal lines etched in a card or board (printed circuit board). The bus extends across all of the system components, each of which taps into some or all of the bus lines. The classic physical arrangement is depicted in Figure 3.17. In this example, the bus consists

Scanned by CamScanner

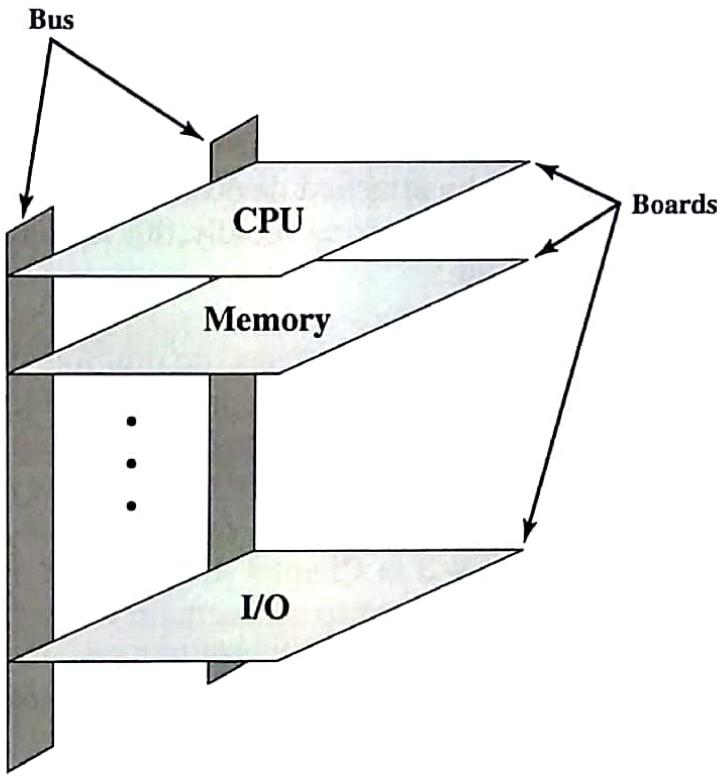


Figure 3.17 Typical Physical Realization of a Bus Architecture

of two vertical columns of conductors. At regular intervals along the columns, there are attachment points in the form of slots that extend out horizontally to support a printed circuit board. Each of the major system components occupies one or more boards and plugs into the bus at these slots. The entire arrangement is housed in a chassis. This scheme can still be used for some of the buses associated with a computer system. However, modern systems tend to have all of the major components on the same board with more elements on the same chip as the processor. Thus, an on-chip bus may connect the processor and cache memory, whereas an on-board bus may connect the processor to main memory and other components.

This arrangement is most convenient. A small computer system may be acquired and then expanded later (more memory, more I/O) by adding more boards. If a component on a board fails, that board can easily be removed and replaced.

Multiple-Bus Hierarchies

If a great number of devices are connected to the bus, performance will suffer. There are two main causes:

1. In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay. This delay determines the time it takes for devices to coordinate the use of the bus. When control of the bus passes from one device to another frequently, these propagation delays can noticeably affect performance.

Scanned by CamScanner

2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus. This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses (e.g., increasing the data bus from 32 to 64 bits). However, because the data rates generated by attached devices (e.g., graphics and video controllers, network interfaces) are growing rapidly, this is a race that a single bus is ultimately destined to lose.

Accordingly, most computer systems use multiple buses, generally laid out in a hierarchy. A typical traditional structure is shown in Figure 3.18a. There is a local bus that connects the processor to a cache memory and that may support one or more local devices. The cache memory controller connects the cache not only to this local bus, but to a system bus to which are attached all of the main memory modules. As will be discussed in Chapter 4, the use of a cache structure insulates the processor from a requirement to access main memory frequently. Hence, main memory can be moved off of the local bus onto a system bus. In this way, I/O transfers to and from the main memory across the system bus do not interfere with the processor's activity.

It is possible to connect I/O controllers directly onto the system bus. A more efficient solution is to make use of one or more expansion buses for this purpose. An expansion bus interface buffers data transfers between the system bus and the I/O controllers on the expansion bus. This arrangement allows the system to support a wide variety of I/O devices and at the same time insulate memory-to-processor traffic from I/O traffic.

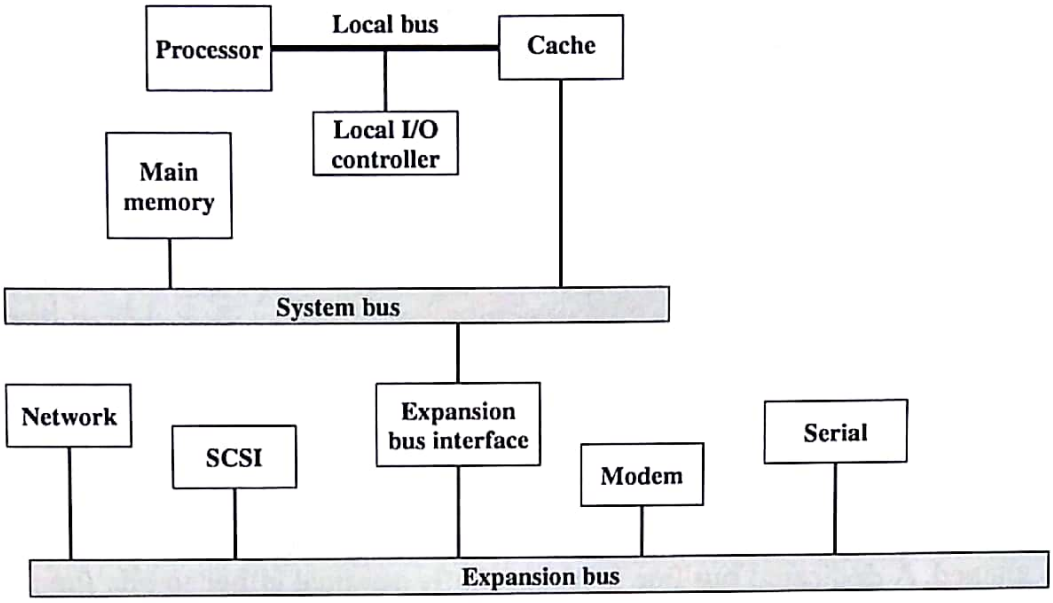
Figure 3.18a shows some typical examples of I/O devices that might be attached to the expansion bus. Network connections include local area networks (LANs) such as a 10-Mbps Ethernet and connections to wide area networks (WANs) such as a packet-switching network. SCSI (small computer system interface) is itself a type of bus used to support local disk drives and other peripherals. A serial port could be used to support a printer or scanner.

This traditional bus architecture is reasonably efficient but begins to break down as higher and higher performance is seen in the I/O devices. In response to these growing demands, a common approach taken by industry is to build a high-speed bus that is closely integrated with the rest of the system, requiring only a bridge between the processor's bus and the high-speed bus. This arrangement is sometimes known as a mezzanine architecture.

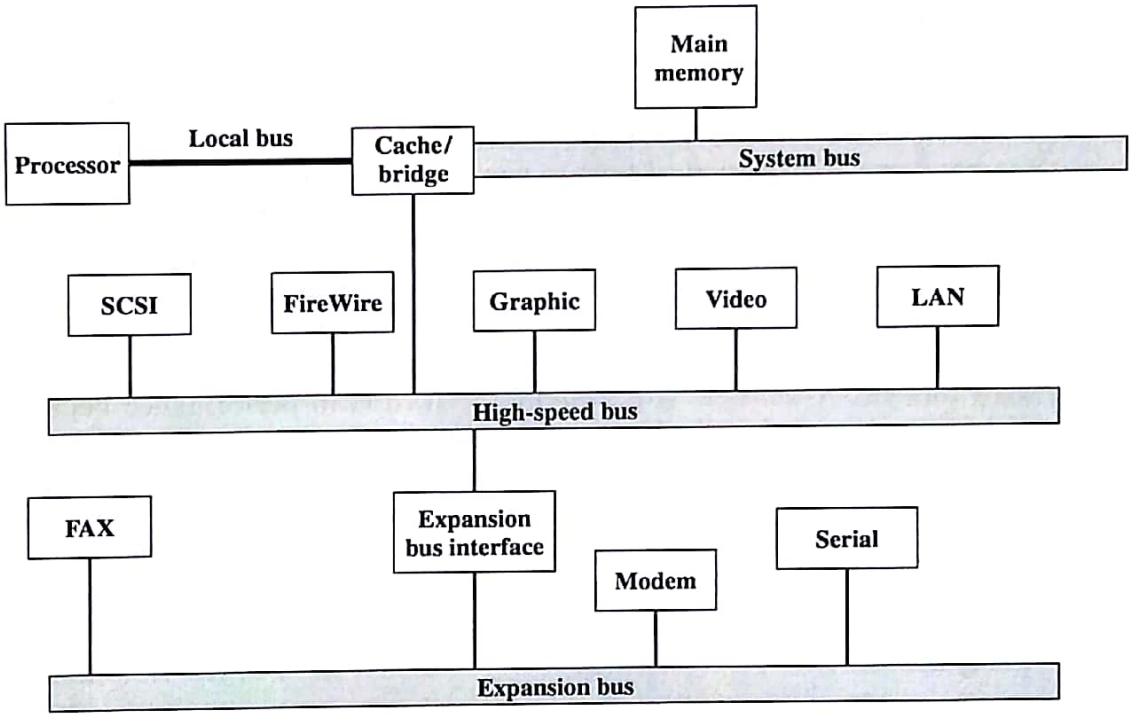
Figure 3.18b shows a typical realization of this approach. Again, there is a local bus that connects the processor to a cache controller, which is in turn connected to a system bus that supports main memory. The cache controller is integrated into a bridge, or buffering device, that connects to the high-speed bus. This bus supports connections to high-speed LANs, such as Fast Ethernet at 100 Mbps, video and graphics workstation controllers, as well as interface controllers to local peripheral buses, including SCSI and FireWire. The latter is a high-speed bus arrangement specifically designed to support high-capacity I/O devices. Lower-speed devices are still supported off an expansion bus, with an interface buffering traffic between the expansion bus and the high-speed bus.

The advantage of this arrangement is that the high-speed bus brings high-demand devices into closer integration with the processor and at the same time is

Scanned by CamScanner



(a) Traditional bus architecture



(b) High-performance architecture

Figure 3.18 Example Bus Configurations

independent of the processor. Thus, differences in processor and high-speed bus speeds and signal line definitions are tolerated. Changes in processor architecture do not affect the high-speed bus, and vice versa.

Elements of Bus Design

Although a variety of different bus implementations exist, there are a few basic parameters or design elements that serve to classify and differentiate buses. Table 3.2 lists key elements.

Scanned by CamScanner

Table 3.2 Elements of Bus Design

Type	**Bus Width**
Dedicated	Address
Multiplexed	Data
Method of Arbitration	**Data Transfer Type**
Centralized	Read
Distributed	Write
Timing	Read-modify-write
Synchronous	Read-after-write
Asynchronous	Block

BUS TYPES Bus lines can be separated into two generic types: dedicated and multiplexed. A dedicated bus line is permanently assigned either to one function or to a physical subset of computer components.

An example of functional dedication is the use of separate dedicated address and data lines, which is common on many buses. However, it is not essential. For example, address and data information may be transmitted over the same set of lines using an Address Valid control line. At the beginning of a data transfer, the address is placed on the bus and the Address Valid line is activated. At this point, each module has a specified period of time to copy the address and determine if it is the addressed module. The address is then removed from the bus, and the same bus connections are used for the subsequent read or write data transfer. This method of using the same lines for multiple purposes is known as *time multiplexing.*

The advantage of time multiplexing is the use of fewer lines, which saves space and, usually, cost. The disadvantage is that more complex circuitry is needed within each module. Also, there is a potential reduction in performance because certain events that share the same lines cannot take place in parallel.

Physical dedication refers to the use of multiple buses, each of which connects only a subset of modules. A typical example is the use of an I/O bus to interconnect all I/O modules; this bus is then connected to the main bus through some type of I/O adapter module. The potential advantage of physical dedication is high throughput, because there is less bus contention. A disadvantage is the increased size and cost of the system.

METHOD OF ARBITRATION In all but the simplest systems, more than one module may need control of the bus. For example, an I/O module may need to read or write directly to memory, without sending the data to the processor. Because only one unit at a time can successfully transmit over the bus, some method of arbitration is needed. The various methods can be roughly classified as being either centralized or distributed. In a centralized scheme, a single hardware device, referred to as a *bus controller* or *arbiter,* is responsible for allocating time on the bus. The device may be a separate module or part of the processor. In a distributed scheme, there is no central controller. Rather, each module contains access control logic and the modules act together to share the bus. With both methods of arbitration, the purpose is to designate one device, either the processor or an I/O module, as master. The master

Scanned by CamScanner

may then initiate a data transfer (e.g., read or write) with some other device, which acts as slave for this particular exchange.

TIMING Timing refers to the way in which events are coordinated on the bus. Buses use either synchronous timing or asynchronous timing.

With **synchronous timing**, the occurrence of events on the bus is determined by a clock. The bus includes a clock line upon which a clock transmits a regular sequence of alternating 1s and 0s of equal duration. A single 1–0 transmission is referred to as a *clock cycle* or *bus cycle* and defines a time slot. All other devices on the bus can read the clock line, and all events start at the beginning of a clock cycle. Figure 3.19 shows a typical, but simplified, timing diagram for synchronous read and write operations (see Appendix 3A for a description of timing diagrams). Other bus signals may change at the leading edge of the clock signal (with a slight reaction delay). Most events occupy a single clock cycle. In this simple example, the processor places a memory address on the address lines during the first

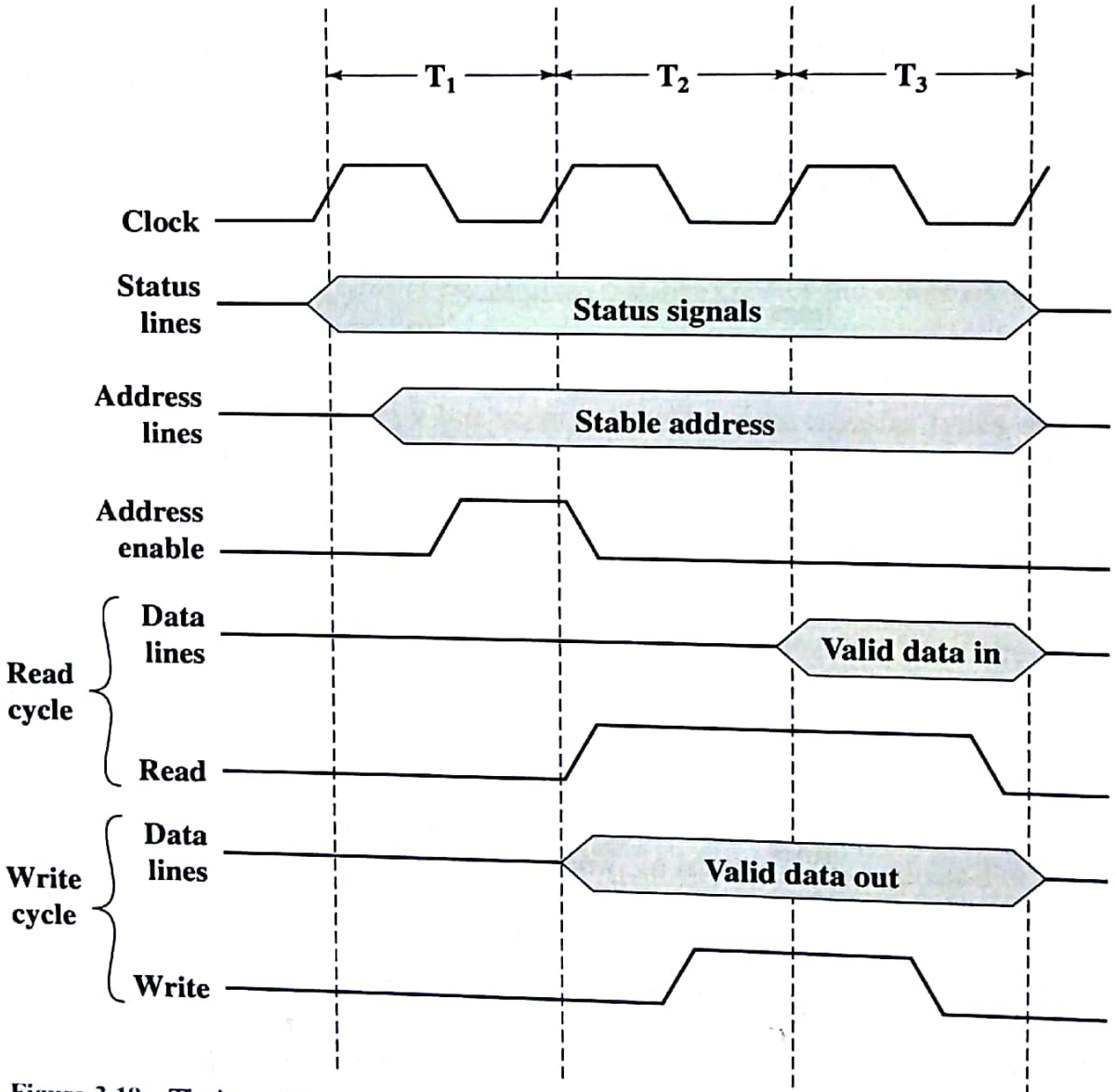
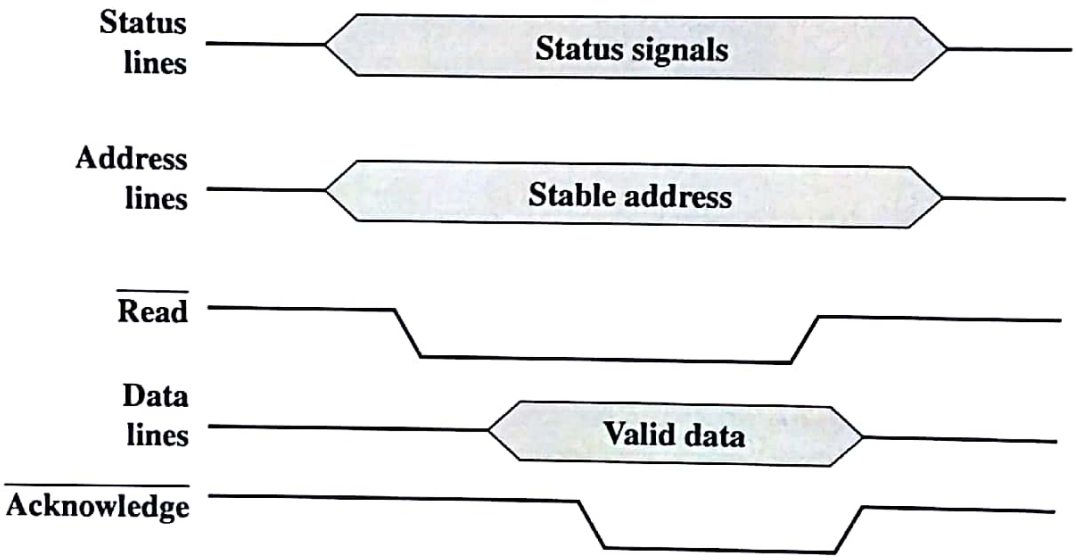


Figure 3.19 Timing of Synchronous Bus Operations

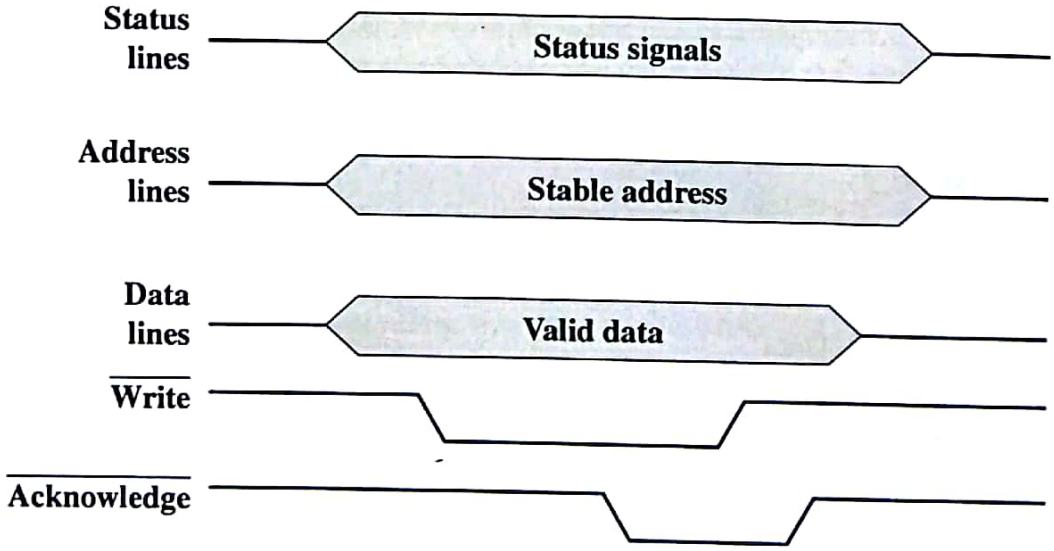
Scanned by CamScanner

clock cycle and may assert various status lines. Once the address lines have stabilized, the processor issues an address enable signal. For a read operation, the processor issues a read command at the start of the second cycle. A memory module recognizes the address and, after a delay of one cycle, places the data on the data lines. The processor reads the data from the data lines and drops the read signal. For a write operation, the processor puts the data on the data lines at the start of the second cycle, and issues a write command after the data lines have stabilized. The memory module copies the information from the data lines during the third clock cycle.

With **asynchronous timing**, the occurrence of one event on a bus follows and depends on the occurrence of a previous event. In the simple read example of Figure 3.20a, the processor places address and status signals on the bus. After



(a) System bus read cycle



(b) System bus write cycle

Figure 3.20 Timing of Asynchronous Bus Operations

Scanned by CamScanner

pausing for these signals to stabilize, it issues a read command, indicating the presence of valid address and control signals. The appropriate memory decodes the address and responds by placing the data on the data line. Once the data lines have stabilized, the memory module asserts the acknowledged line to signal the processor that the data are available. Once the master has read the data from the data lines, it deasserts the read signal. This causes the memory module to drop the data and acknowledge lines. Finally, once the acknowledge line is dropped, the master removes the address information.

Figure 3.20b shows a simple asynchronous write operation. In this case, the master places the data on the data line at the same time that is puts signals on the status and address lines. The memory module responds to the write command by copying the data from the data lines and then asserting the acknowledge line. The master then drops the write signal and the memory module drops the acknowledge signal.

Synchronous timing is simpler to implement and test. However, it is less flexible than asynchronous timing. Because all devices on a synchronous bus are tied to a fixed clock rate, the system cannot take advantage of advances in device performance. With asynchronous timing, a mixture of slow and fast devices, using older and newer technology, can share a bus.

BUS WIDTH We have already addressed the concept of bus width. The width of the data bus has an impact on system performance: The wider the data bus, the greater the number of bits transferred at one time. The width of the address bus has an impact on system capacity: the wider the address bus, the greater the range of locations that can be referenced.

DATA TRANSFER TYPE Finally, a bus supports various data transfer types, as illustrated in Figure 3.21. All buses support both write (master to slave) and read (slave to master) transfers. In the case of a multiplexed address/data bus, the bus is first used for specifying the address and then for transferring the data. For a read operation, there is typically a wait while the data are being fetched from the slave to be put on the bus. For either a read or a write, there may also be a delay if it is necessary to go through arbitration to gain control of the bus for the remainder of the operation (i.e., seize the bus to request a read or write, then seize the bus again to perform a read or write).

In the case of dedicated address and data buses, the address is put on the address bus and remains there while the data are put on the data bus. For a write operation, the master puts the data onto the data bus as soon as the address has stabilized and the slave has had the opportunity to recognize its address. For a read operation, the slave puts the data onto the data bus as soon as it has recognized its address and has fetched the data.

There are also several combination operations that some buses allow. A read–modify–write operation is simply a read followed immediately by a write to the same address. The address is only broadcast once at the beginning of the operation. The whole operation is typically indivisible to prevent any access to the data element by other potential bus masters. The principal purpose of this

Scanned by CamScanner

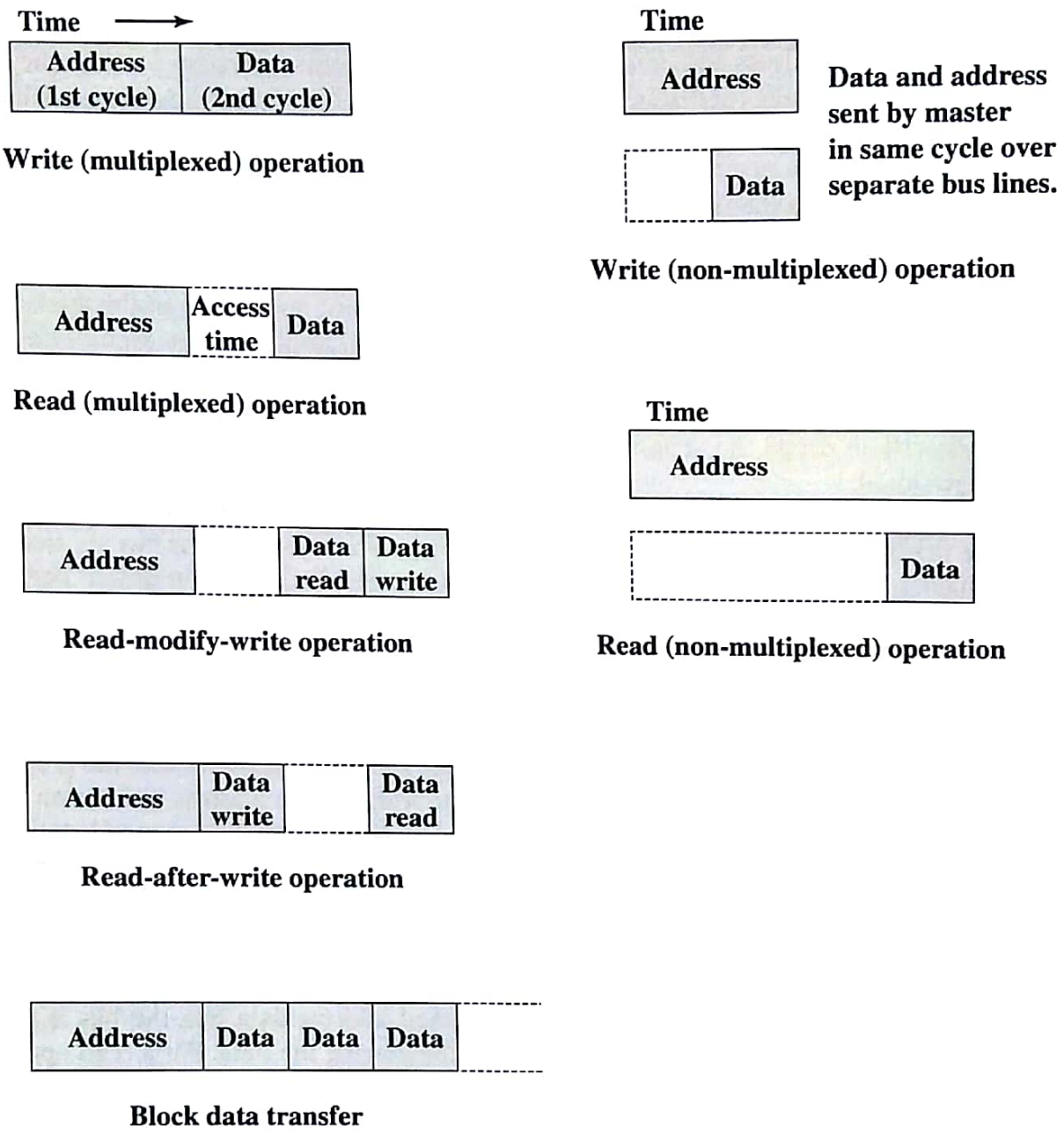


Figure 3.21 Bus Data Transfer Types

capability is to protect shared memory resources in a multiprogramming system (see Chapter 8).

Read-after-write is an indivisible operation consisting of a write followed immediately by a read from the same address. The read operation may be performed for checking purposes.

Some bus systems also support a block data transfer. In this case, one address cycle is followed by n data cycles. The first data item is transferred to or from the specified address; the remaining data items are transferred to or from subsequent addresses.

Scanned by CamScanner

3.5 PCI

The peripheral component interconnect (PCI) is a popular high-bandwidth, processor-independent bus that can function as a mezzanine or peripheral bus. Compared with other common bus specifications, PCI delivers better system performance for high-speed I/O subsystems (e.g., graphic display adapters, network interface controllers, disk controllers, and so on). The current standard allows the use of up to 64 data lines at 66 MHz, for a raw transfer rate of 528 MByte/s, or 4.224 Gbps. But it is not just a high speed that makes PCI attractive. PCI is specifically designed to meet economically the I/O requirements of modern systems; it requires very few chips to implement and supports other buses attached to the PCI bus.

Intel began work on PCI in 1990 for its Pentium-based systems. Intel soon released all the patents to the public domain and promoted the creation of an industry association, the PCI Special Interest Group (SIG), to develop further and maintain the compatibility of the PCI specifications. The result is that PCI has been widely adopted and is finding increasing use in personal computer, workstation, and server systems. Because the specification is in the public domain and is supported by a broad cross section of the microprocessor and peripheral industry, PCI products built by different vendors are compatible.

PCI is designed to support a variety of microprocessor-based configurations, including both single- and multiple-processor systems. Accordingly, it provides a general-purpose set of functions. It makes use of synchronous timing and a centralized arbitration scheme.

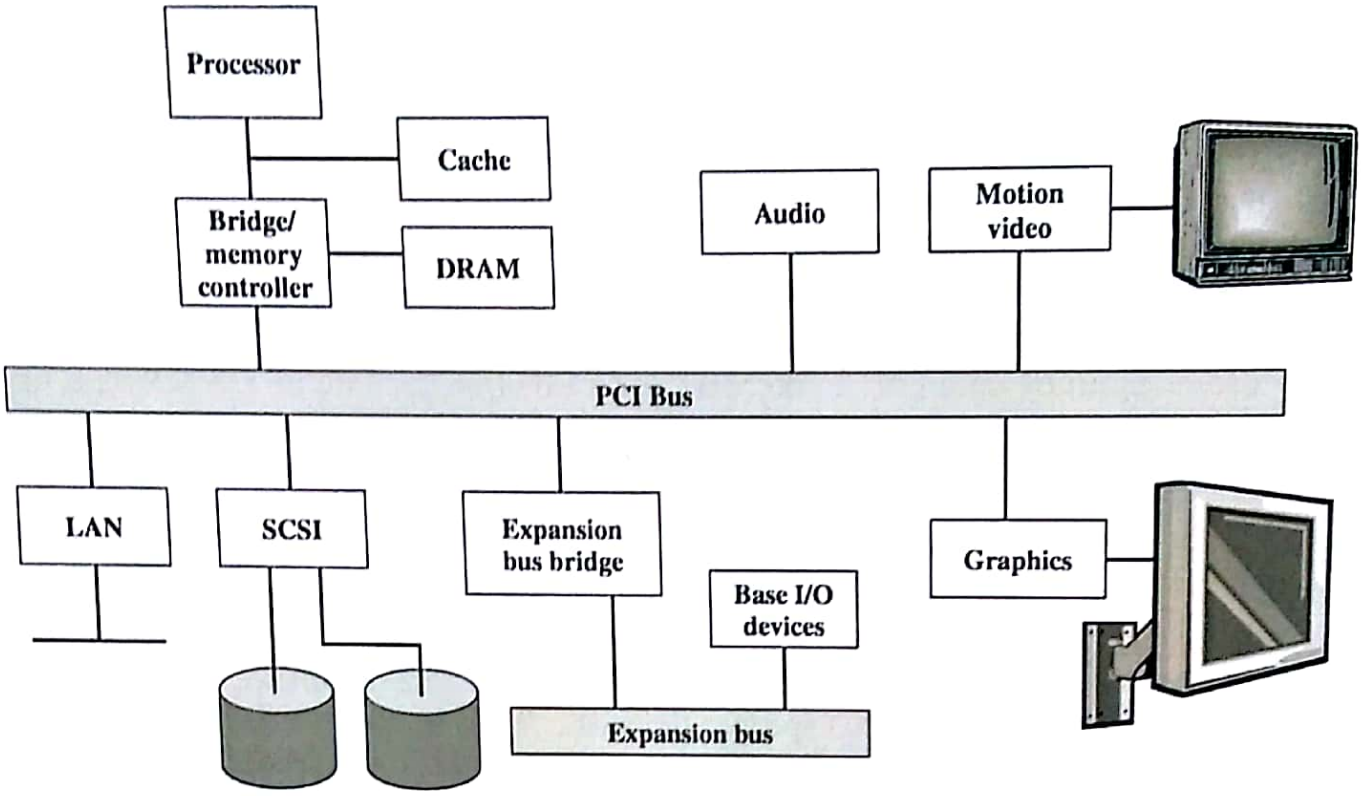
Figure 3.22a shows a typical use of PCI in a single-processor system. A combined DRAM controller and bridge to the PCI bus provides tight coupling with the processor and the ability to deliver data at high speeds. The bridge acts as a data buffer so that the speed of the PCI bus may differ from that of the processor's I/O capability. In a multiprocessor system (Figure 3.22b), one or more PCI configurations may be connected by bridges to the processor's system bus. The system bus supports only the processor/cache units, main memory, and the PCI bridges. Again, the use of bridges keeps the PCI independent of the processor speed yet provides the ability to receive and deliver data rapidly.

Bus Structure

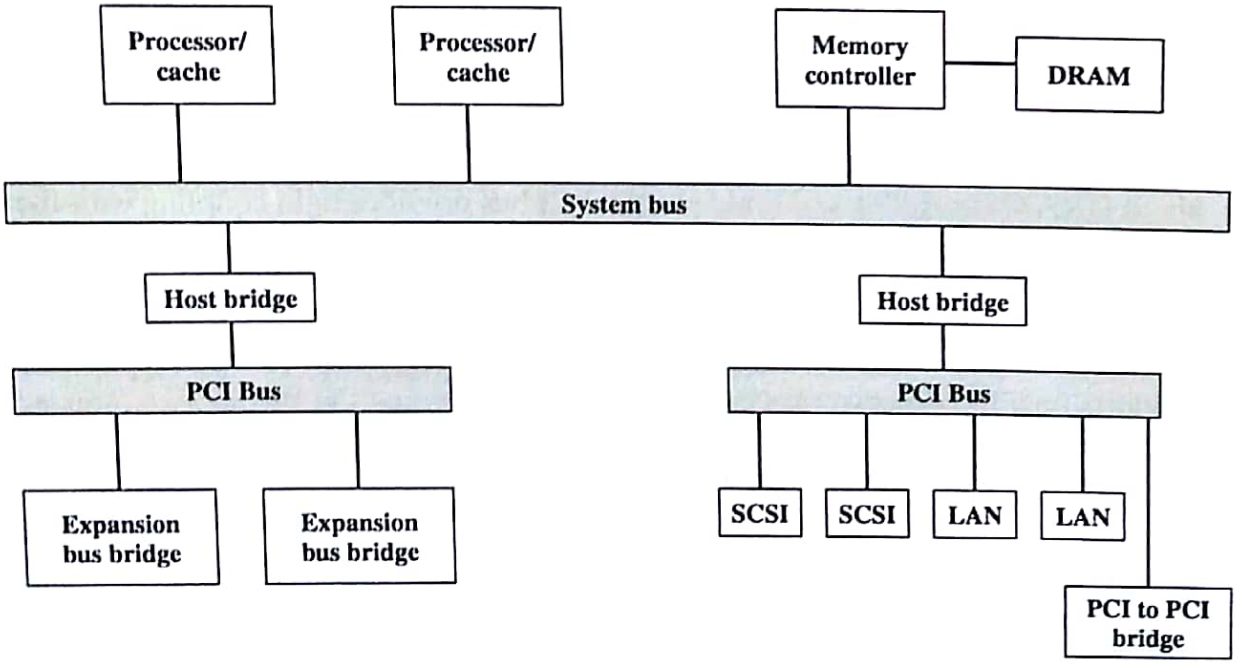
PCI may be configured as a 32- or 64-bit bus. Table 3.3 defines the 49 mandatory signal lines for PCI. These are divided into the following functional groups:

- **System pins:** Include the clock and reset pins.
- **Address and data pins:** Include 32 lines that are time multiplexed for addresses and data. The other lines in this group are used to interpret and validate the signal lines that carry the addresses and data.
- **Interface control pins:** Control the timing of transactions and provide coordination among initiators and targets.

Scanned by CamScanner



(a) Typical desktop system



(b) Typical server system

Figure 3.22 Example PCI Configurations

- **Arbitration pins:** Unlike the other PCI signal lines, these are not shared lines. Rather, each PCI master has its own pair of arbitration lines that connect it directly to the PCI bus arbiter.
- **Error reporting pins:** Used to report parity and other errors.

Scanned by CamScanner

Table 3.3 Mandatory PCI Signal Lines

Designation	Type	Description
		System Pins
CLK	in	Provides timing for all transactions and is sampled by all inputs on the rising edge. Clock rates up to 33 MHz are supported.
RST#	in	Forces all PCI-specific registers, sequencers, and signals to an initialized state.
		Address and Data Pins
AD[31::0]	t/s	Multiplexed lines used for address and data
C/BE[3::0]#	t/s	Multiplexed bus command and byte enable signals. During the data phase, the lines indicate which of the four byte lanes carry meaningful data.
PAR	t/s	Provides even parity across AD and C/BE lines one clock cycle later. The master drives PAR for address and write data phases; the target drive PAR for read data phases.
		Interface Control Pins
FRAME#	s/t/s	Driven by current master to indicate the start and duration of a transaction. It is asserted at the start and deasserted when the initiator is ready to begin the final data phase.
IRDY#	s/t/s	Initiator Ready. Driven by current bus master (initiator of transaction). During a read, indicates that the master is prepared to accept data; during a write, indicates that valid data are present on AD.
TRDY#	s/t/s	Target Ready. Driven by the target (selected device). During a read, indicates that valid data are present on AD; during a write, indicates that target is ready to accept data.
STOP#	s/t/s	Indicates that current target wishes the initiator to stop the current transaction.
IDSEL	in	Initialization Device Select. Used as a chip select during configuration read and write transactions.
DEVSEL#	in	Device Select. Asserted by target when it has recognized its address. Indicates to current initiator whether any device has been selected.
		Arbitration Pins
REQ#	t/s	Indicates to the arbiter that this device requires use of the bus. This is a device-specific point-to-point line.
GNT#	t/s	Indicates to the device that the arbiter has granted bus access. This is a device-specific point-to-point line.
		Error Reporting Pins
PERR#	s/t/s	Parity Error. Indicates a data parity error is detected by a target during a write data phase or by an initiator during a read data phase.
SERR#	o/d	System Error. May be pulsed by any device to report address parity errors and critical errors other than parity.

In addition, the PCI specification defines 51 optional signal lines (Table 3.4), divided into the following functional groups:

- **Interrupt pins:** These are provided for PCI devices that must generate requests for service. As with the arbitration pins, these are not shared lines. Rather, each PCI device has its own interrupt line or lines to an interrupt controller.

Scanned by CamScanner

Table 3.4 Optional PCI Signal Lines

Designation	Type	Description
Interrupt Pins		
INTA#	o/d	Used to request an interrupt.
INTB#	o/d	Used to request an interrupt; only has meaning on a multifunction device.
INTC#	o/d	Used to request an interrupt; only has meaning on a multifunction device.
INTD#	o/d	Used to request an interrupt; only has meaning on a multifunction device.
Cache Support Pins		
SBO#	in/out	Snoop Backoff. Indicates a hit to a modified line.
SDONE	in/out	Snoop Done. Indicates the status of the snoop for the current access. Asserted when snoop has been completed.
64-Bit Bus Extension Pins		
AD[63::32]	t/s	Multiplexed lines used for address and data to extend bus to 64 bits.
C/BE[7::4]#	t/s	Multiplexed bus command and byte enable signals. During the address phase, the lines provide additional bus commands. During the data phase, the lines indicate which of the four extended byte lanes carry meaningful data.
REQ64#	s/t/s	Used to request 64-bit transfer.
ACK64#	s/t/s	Indicates target is willing to perform 64-bit transfer.
PAR64	t/s	Provides even parity across extended AD and C/BE lines one clock cycle later.
JTAG/Boundary Scan Pins		
TCK	in	Test clock. Used to clock state information and test data into and out of the device during boundary scan.
TDI	in	Test input. Used to serially shift test data and instructions into the device.
TDO	out	Test output. Used to serially shift test data and instructions out of the device.
TMS	in	Test mode Select. Used to control state of test access port controller.
TRST#	in	Test reset. Used to initialize test access port controller.

in Input-only signal
out Output-only signal
t/s Bidirectional, tri-state, I/O signal
s/t/s Sustained tri-state signal driven by only one owner at a time
o/d Open drain: allows multiple devices to share as a wire-OR
Signal's active state occurs at low voltage

- **Cache support pins:** These pins are needed to support a memory on PCI that can be cached in the processor or another device. These pins support snoopy cache protocols (see Chapter 18 for a discussion of such protocols).
- **64-bit bus extension pins:** Include 32 lines that are time multiplexed for addresses and data and that are combined with the mandatory address/data lines to form a 64-bit address/data bus. Other lines in this group are used to interpret and validate the signal lines that carry the addresses and data. Finally, there are two lines that enable two PCI devices to agree to the use of the 64-bit capability.
- **JTAG/boundary scan pins:** These signal lines support testing procedures defined in IEEE Standard 1149.1.

Scanned by CamScanner

PCI Commands

Bus activity occurs in the form of transactions between an initiator, or master, and a target. When a bus master acquires control of the bus, it determines the type of transaction that will occur next. During the address phase of the transaction, the C/BE lines are used to signal the transaction type. The commands are as follows:

- Interrupt Acknowledge
- Special Cycle
- I/O Read
- I/O Write
- Memory Read
- Memory Read Line
- Memory Read Multiple
- Memory Write
- Memory Write and Invalidate
- Configuration Read
- Configuration Write
- Dual address Cycle

Interrupt Acknowledge is a read command intended for the device that functions as an interrupt controller on the PCI bus. The address lines are not used during the address phase, and the byte enable lines indicate the size of the interrupt identifier to be returned.

The Special Cycle command is used by the initiator to broadcast a message to one or more targets.

The I/O Read and Write commands are used to transfer data between the initiator and an I/O controller. Each I/O device has its own address space, and the address lines are used to indicate a particular device and to specify the data to be transferred to or from that device. The concept of I/O addresses is explored in Chapter 7.

The memory read and write commands are used to specify the transfer of a burst of data, occupying one or more clock cycles. The interpretation of these commands depends on whether or not the memory controller on the PCI bus supports the PCI protocol for transfers between memory and cache. If so, the transfer of data to and from the memory is typically in terms of cache lines, or blocks.[3] The three memory read commands have the uses outlined in Table 3.5. The Memory Write command is used to transfer data in one or more data cycles to memory. The Memory Write and Invalidate command transfers data in one or more cycles to memory. In addition, it guarantees that at least one cache line is written. This command supports the cache function of writing back a line to memory.

The two configuration commands enable a master to read and update configuration parameters in a device connected to the PCI. Each PCI device may include

[3]The fundamental principles of cache memory are described in Chapter 4; bus-based cache protocols are described in Chapter 17.

Scanned by CamScanner

Table 3.5 Interpretation of PCI Read Commands

Read Command Type	For Cachable Memory	For Noncachable Memory
Memory Read	Bursting one-half or less of a cache line	Bursting 2 data transfer cycles or less
Memory Read Line	Bursting more than one-half a cache line to three cache lines	Bursting 3 to 12 data transfers
Memory Read Multiple	Bursting more than three cache lines	Bursting more than 12 data transfers

up to 256 internal registers that are used during system initialization to configure that device.

The Dual Address Cycle command is used by an initiator to indicate that it is using 64-bit addressing.

Data Transfers

Every data transfer on the PCI bus is a single transaction consisting of one address phase and one or more data phases. In this discussion, we illustrate a typical read operation; a write operation proceeds similarly.

Figure 3.23 shows the timing of the read transaction. All events are synchronized to the falling transitions of the clock, which occur in the middle of each clock cycle. Bus devices sample the bus lines on the rising edge at the beginning of a bus cycle. The following are the significant events, labeled on the diagram:

a. Once a bus master has gained control of the bus, it may begin the transaction by asserting FRAME. This line remains asserted until the initiator is ready to complete the last data phase. The initiator also puts the start address on the address bus, and the read command on the C/BE lines.

b. At the start of clock 2, the target device will recognize its address on the AD lines.

c. The initiator ceases driving the AD bus. A turnaround cycle (indicated by the two circular arrows) is required on all signal lines that may be driven by more than one device, so that the dropping of the address signal will prepare the bus for use by the target device. The initiator changes the information on the C/BE lines to designate which AD lines are to be used for transfer for the currently addressed data (from 1 to 4 bytes). The initiator also asserts IRDY to indicate that it is ready for the first data item.

d. The selected target asserts DEVSEL to indicate that it has recognized its address and will respond. It places the requested data on the AD lines and asserts TRDY to indicate that valid data are present on the bus.

e. The initiator reads the data at the beginning of clock 4 and changes the byte enable lines as needed in preparation for the next read.

f. In this example, the target needs some time to prepare the second block of data for transmission. Therefore, it deasserts TRDY to signal the initiator that there will not be new data during the coming cycle. Accordingly, the initiator does not read the data lines at the beginning of the fifth clock cycle and does not change byte enable during that cycle. The block of data is read at beginning of clock 6.

Scanned by CamScanner

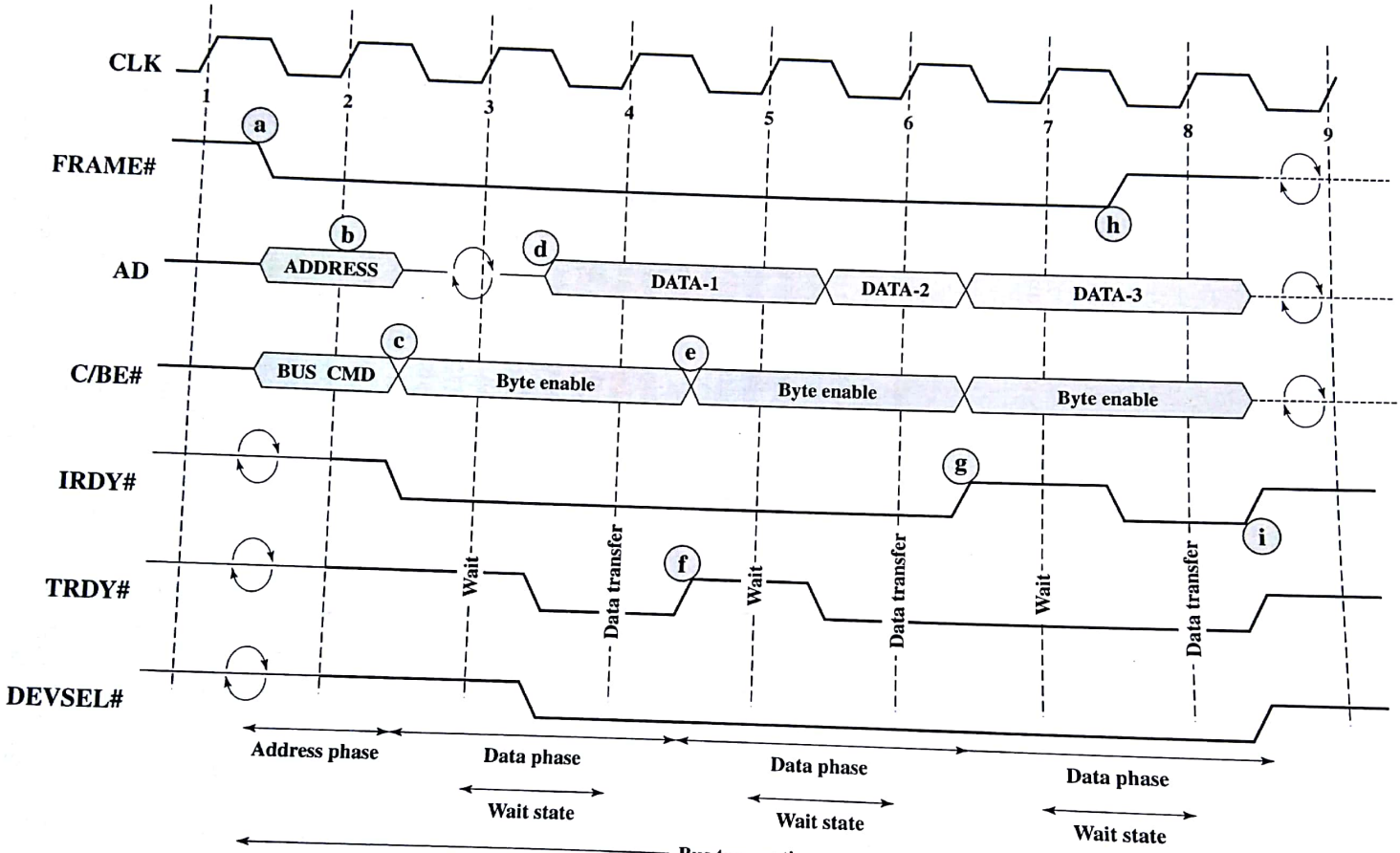


Figure 3.23 PCI Read Operation

Scanned by CamScanner

g. During clock 6, the target places the third data item on the bus. However, in this example, the initiator is not yet ready to read the data item (e.g., it has a temporary buffer full condition). It therefore deasserts IRDY. This will cause the target to maintain the third data item on the bus for an extra clock cycle.

h. The initiator knows that the third data transfer is the last, and so it deasserts FRAME to signal the target that this is the last data transfer. It also asserts IRDY to signal that it is ready to complete that transfer.

i. The initiator deasserts IRDY, returning the bus to the idle state, and the target deasserts TRDY and DEVSEL.

Arbitration

PCI makes use of a centralized, synchronous arbitration scheme in which each master has a unique request (REQ) and grant (GNT) signal. These signal lines are attached to a central arbiter (Figure 3.24) and a simple request–grant handshake is used to grant access to the bus.

The PCI specification does not dictate a particular arbitration algorithm. The arbiter can use a first-come-first-served approach, a round-robin approach, or some sort of priority scheme. A PCI master must arbitrate for each transaction that it wishes to perform, where a single transaction consists of an address phase followed by one or more contiguous data phases.

Figure 3.25 is an example in which devices A and B are arbitrating for the bus. The following sequence occurs:

a. At some point prior to the start of clock 1, A has asserted its REQ signal. The arbiter samples this signal at the beginning of clock cycle 1.

b. During clock cycle 1, B requests use of the bus by asserting its REQ signal.

c. At the same time, the arbiter asserts GNT-A to grant bus access to A.

d. Bus master A samples GNT-A at the beginning of clock 2 and learns that it has been granted bus access. It also finds IRDY and TRDY deasserted, indicating that the bus is idle. Accordingly, it asserts FRAME and places the address information on the address bus and the command on the C/BE bus (not shown). It also continues to assert REQ-A, because it has a second transaction to perform after this one.

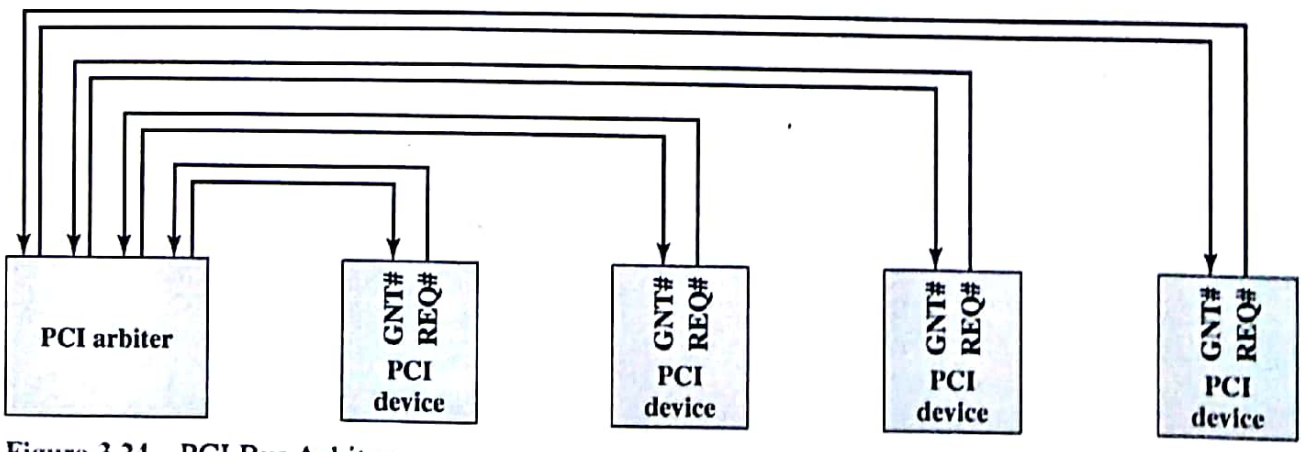


Figure 3.24 PCI Bus Arbiter

Scanned by CamScanner

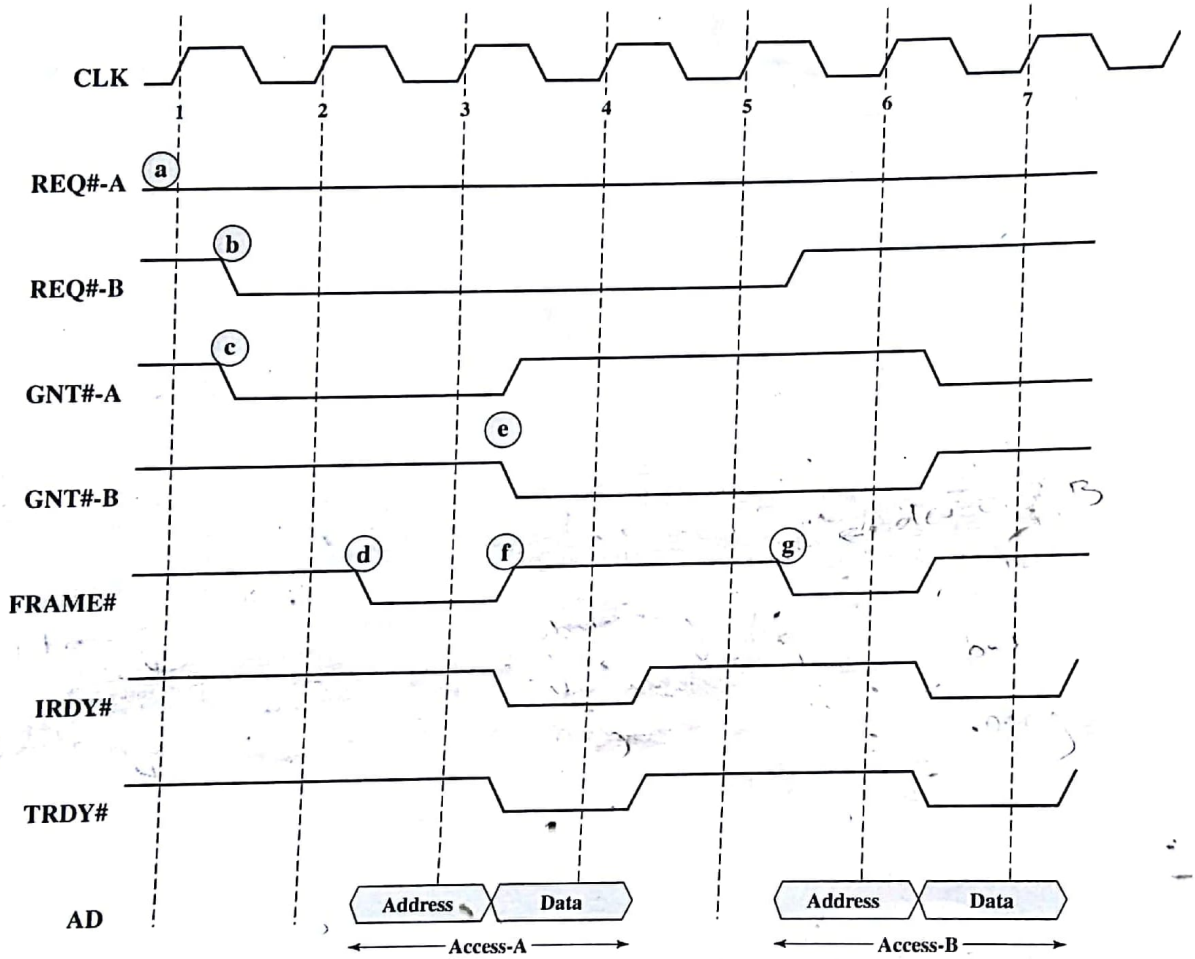


Figure 3.25 PCI Bus Arbitration between Two Masters

Scanned by CamScanner

e. The bus arbiter samples all REQ lines at the beginning of clock 3 and makes an arbitration decision to grant the bus to B for the next transaction. It then asserts GNT-B and deasserts GNT-A. B will not be able to use the bus until it returns to an idle state.

f. A deasserts FRAME to indicate that the last (and only) data transfer is in progress. It puts the data on the data bus and signals the target with IRDY. The target reads the data at the beginning of the next clock cycle.

g. At the beginning of clock 5, B finds IRDY and FRAME deasserted and so is able to take control of the bus by asserting FRAME. It also deasserts its REQ line, because it only wants to perform one transaction.

Subsequently, master A is granted access to the bus for its next transaction.

Notice that arbitration can take place at the same time that the current bus master is performing a data transfer. Therefore, no bus cycles are lost in performing arbitration. This is referred to as *hidden arbitration.*

3.6 RECOMMENDED READING AND WEB SITES

The clearest book-length description of PCI is [SHAN99]. [ABBO04] also contains a lot of solid information on PCI.

ABBO04 Abbot, D. *PCI Bus Demystified.* New York: Elsevier, 2004.

SHAN99 Shanley, T., and Anderson, D. *PCI Systems Architecture.* Richardson, TX: Mindshare Press, 1999.

Recommended Web sites:

- **PCI Special Interest Group:** Information about PCI specifications and products
- **PCI Pointers:** Links to PCI vendors and other sources of information

3.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

address bus	distributed arbitration	memory address register (MAR)
asynchronous timing	instruction cycle	memory buffer register (MBR)
bus	instruction execute	peripheral component interconnect (PCI)
bus arbitration	instruction fetch	synchronous timing
bus width	interrupt	system bus
centralized arbitration	interrupt handler	
data bus	interrupt service routine	
disabled interrupt		

Scanned by CamScanner

Review Questions

3.1 What general categories of functions are specified by computer instructions?

3.2 List and briefly define the possible states that define an instruction execution.

3.3 List and briefly define two approaches to dealing with multiple interrupts.

3.4 What types of transfers must a computer's interconnection structure (e.g., bus) support?

3.5 What is the benefit of using a multiple-bus architecture compared to a single-bus architecture?

3.6 List and briefly define the functional groups of signal lines for PCI.

Problems

3.1 The hypothetical machine of Figure 3.4 also has two I/O instructions:

0011 = Load AC from I/O
0011 = Store AC to I/O

In these cases, the 12-bit address identifies a particular I/O device. Show the program execution (using the format of Figure 3.5) for the following program:

1. Load AC from device 5.
2. Add contents of memory location 940.
3. Store AC to device 6.

Assume that the next value retrieved from device 5 is 3 and that location 940 contains a value of 2.

3.2 The program execution of Figure 3.5 is described in the text using six steps. Expand this description to show the use of the MAR and MBR.

3.3 Consider a hypothetical 32-bit microprocessor having 32-bit instructions composed of two fields: the first byte contains the opcode and the remainder the immediate operand or an operand address.

a. What is the maximum directly addressable memory capacity (in bytes)?
b. Discuss the impact on the system speed if the microprocessor bus has
   1. a 32-bit local address bus and a 16-bit local data bus, or
   2. a 16-bit local address bus and a 16-bit local data bus.
c. How many bits are needed for the program counter and the instruction register?

3.4 Consider a hypothetical microprocessor generating a 16-bit address (for example, assume that the program counter and the address registers are 16 bits wide) and having a 16-bit data bus.

a. What is the maximum memory address space that the processor can access directly if it is connected to a "16-bit memory"?
b. What is the maximum memory address space that the processor can access directly if it is connected to an "8-bit memory"?
c. What architectural features will allow this microprocessor to access a separate "I/O space"?
d. If an input and an output instruction can specify an 8-bit I/O port number, how many 8-bit I/O ports can the microprocessor support? How many 16-bit I/O ports? Explain.

3.5 Consider a 32-bit microprocessor, with a 16-bit external data bus, driven by an 8-MHz input clock. Assume that this microprocessor has a bus cycle whose minimum duration equals four input clock cycles. What is the maximum data transfer rate across the bus that this microprocessor can sustain, in bytes/s? To increase its performance, would it be better to make its external data bus 32 bits or to double the external clock frequency supplied to the microprocessor? State any other assumptions

Scanned by CamScanner