UNIT – V I/O ORGANIZATION

Accessing I/O devices – Interrupts – Direct Memory Access – Buses – Interface Circuits – Standard I/O Interfaces (PCI, SCSI, USB).

5.1 Accessing I/O Devices

- A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement as shown in below fig.

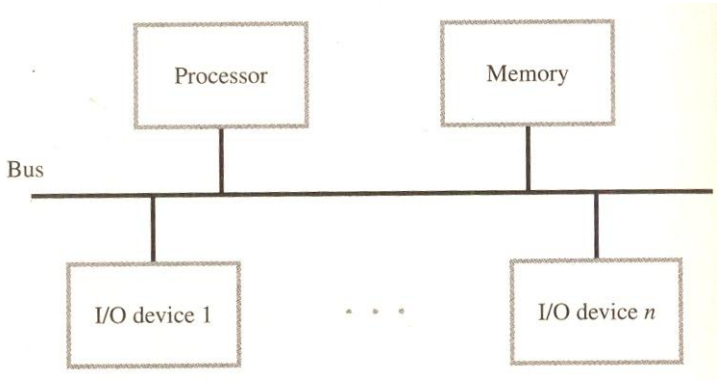

Fig: Single Bus Structure

- Three sets of lines used to carry address, data and control signals. Each I/O device is assigned a unique set of addresses.
- When I/O devices and the memory share the same address space, the arrangement is called **memory-mapped I/O**.
- With memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device.

Eg: Move DATAIN,R0 → Reads the data from DATAIN and stores them into Processor register Ro.
Move R0,DATAOUT → Sends the contents of Register R0 to location DATAOUT.

- When building a computer system based on Intel processors, the designer has the option of connecting I/O devices to use the special I/O address space or simply incorporating them as part of the memory address space.

- A separate I/O address space does not necessarily mean that the I/O address lines are physically separate from the memory address lines.
- A separate signal on the bus indicates that the requested read or write transfer is an I/O operation. When this signal is asserted, the memory unit ignores the requested transfer.

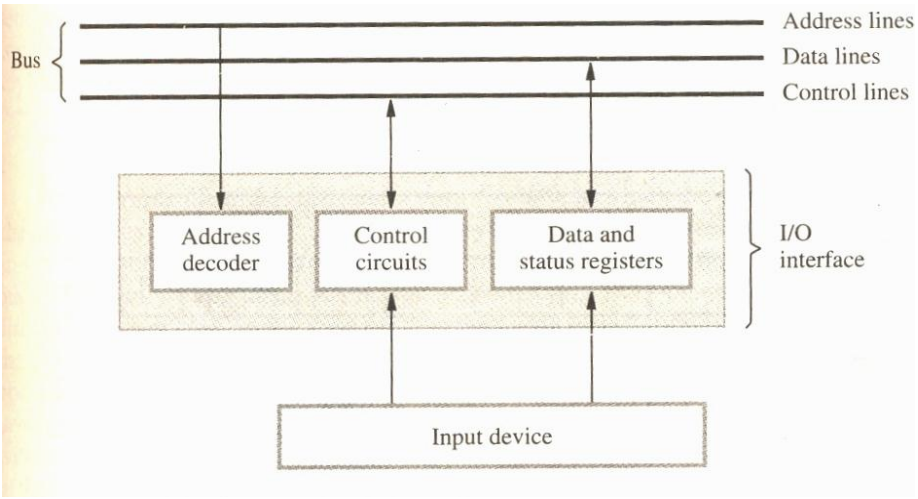

Fig: I/O interface for an input device

The fig.4.2 illustrates the hardware required to connect an I/O device to the bus.

Address Decoder: Enables the device to recognize its address when this address appears on the address lines.

Data Register: This holds the data being transferred to or appears on the address lines.

Status Register: Contains information relevant to the operation of the I/O device.

- Both the data and status Registers are connected to the data bus and assigned unique addresses. The address decoder, the data and status register and the control circuitry required to coordinate I/O transfers.
- An instruction that's reads a character from the keyboard should be executed only when a character is available in the input buffer of the keyboard interface. Also, we must make sure than an input character is read only once.
- For an input device such as a keyboard, a status flag, SIN, is included in the interface circuit as part of the status register.

- This flag is set to 1 when a character is entered at the keyboard and cleared to 0 once this character is read by the processor.
- Hence, by checking the SIN flag, the software can ensure that it is always reading valid data and the program loop repeatedly reads the status register and checks the state of SIN and when SIN becomes 1, the program reads the input data register.
- A similar procedure can be used to control output operations using an output status flag, SOUT.

There are two other commonly used mechanisms for implementing I/O operations: **Interrupts and Direct Memory Access.**

- In case of ***interrupts***, synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation.
- ***Direct Memory Access*** is a technique used for high-speed I/O devices. It involves the device interface data directly to or form the memory without involvement by the processor.

5.2 INTERRUPTS

- The processor has to wait for a long time for the concerned I/O device to be ready either for reception or transmission of a data and must check the status of the I/O device repeatedly.
- Alternate approach is I/O device can send the alert processor when it becomes ready. This is done by sending an **interrupt** to the processor.
- One of the lines in bus is dedicated to pass interrupt; this line is named as **interrupt-request line.**
- The routine executed in response to an interrupt request is called the **interrupt-service routine**.
- Assume that some interrupt occurs while executing the ith instruction. The processor first completes execution of instruction *i.* Then, it loads the program counter with the address of the first instruction of the interrupt-service routine.
- After execution of the interrupt-service routine, the processor has to come back to instruction $i + 1$.

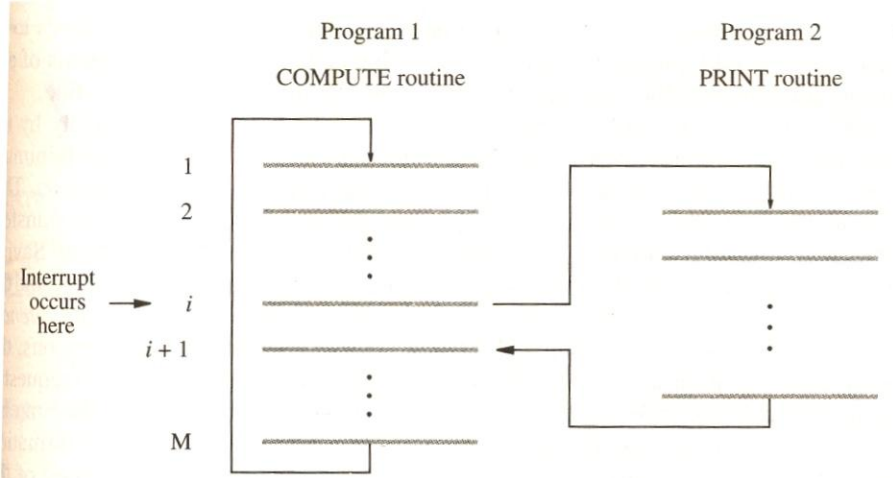

Figure 4.5 Transfer of control through the use of interrupts.

- Therefore, when an interrupt occurs, the current contents of the PC, which point to instruction $i + 1$, must be put in temporary storage. At the end of the interrupt-service routine reloads the PC from that temporary storage location, causing execution to resume at instruction $i + 1$.
- The task of saving and restoring information can be done automatically by the processor or by program instructions. Most modern processors save only the minimum amount of information needed to maintain the integrity of program execution.
- Saving registers also increases the delay between the time an interrupt request is received and the start of execution of the interrupt -service routine. This delay is called ***interrupt latency***.
- Typically, the processor saves only the contents of the program counter and the processor status register. Any additional information that needs to be saved must be saved by program instructions at the beginning of the interrupt-service routine and restored at the end of the routine.
- The concept of interrupts is used in operating systems and in many control applications where processing of certain routines must be accurately timed relative to external events. This application is referred to as ***real-time processing.***

5.2.1 INTERRUPT HARDWARE

A single interrupt-request line may be used to serve *n* devices as depicted in Figure 4.6.

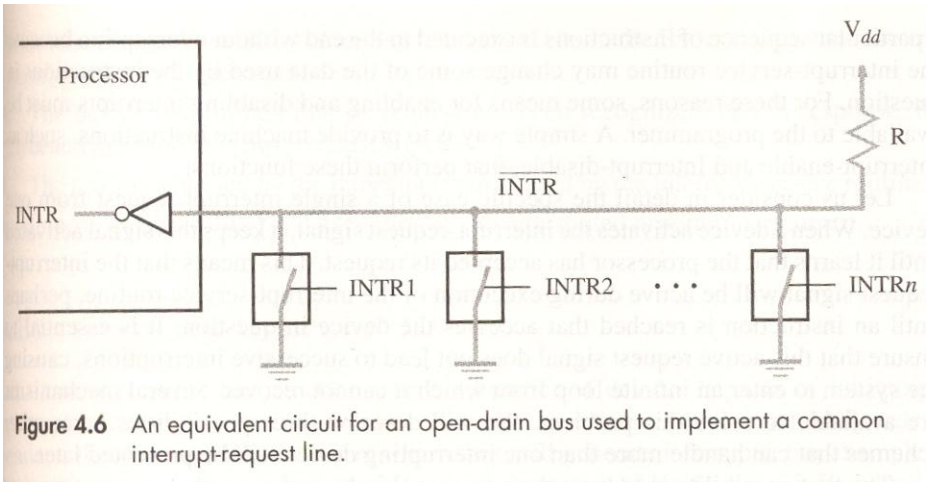

Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

- All devices are connected to the line via switches to ground. To request an interrupt, a device closes its associated switch. Thus, if all interrupt-request signals INTR1 to INTRn are inactive, that is, if all switches are open, the voltage on the interrupt-request line will be equal to V_{dd}. This is the inactive state of the line.
- When a device requests an interrupt by closing its switch, the voltage on the line drops to 0, causing the interrupt-request signal, INTR, received by the processor to go to 1.
- Since the closing of one or more switches will cause the line voltage to drop to 0, the value of INTR is the logical OR of the requests from individual devices, that is,

$$INTR = INTR1 + ... + INTRn$$

- Resistor R is called a *pull-up resistor* because it pulls the line voltage up to the high-voltage state when the switches are open.

5.2.2 ENABLING AND DISABLING INTERRUPTS

The interrupts can arrive at any time, they may alter the sequence of events from that expected by the programmer.

A fundamental facility found in all computers is the ability to enable and disable such interruptions as desired, some means for enabling and disabling interrupts must be

available to the programmer.

We will describe three possibilities here;.

- A simple way is to provide machine instructions, such as Interrupt-enable and Interrupt-disable. By using an Interrupt-disable instruction as the first instruction in the interrupt-service routine, the programmer can ensure that no further interruptions will occur until an Interrupt-enable instruction is executed. Typically, the Interrupt enable instruction will be the last instruction in the interrupt-service routine before the Return-from-interrupt instruction. The processor must guarantee that execution of the Return-from-interrupt instruction is completed before further interruption can occur.

- The second option is to have the processor automatically disable interrupts before starting the execution of the interrupt -service routine. After saving the contents of the PC and the processor status register (PS) on the stack, the processor performs the equivalent of executing an Interrupt-disable instruction. It is often the case that one bit in the PS register, called *Interrupt-enable*, indicates whether interrupts are enabled. When Interrupt-enable bit is 1 it can accept the interrupts. If it accepts it clears the bit that means no other interrupt can use the Interrupt request line. Once the Service routine is processed Bit is set to accept new interrupts.

- In the third option, the processor has a special interrupt-request line for which the interrupt-handling circuit responds only to the leading edge of the signal. Such a line is said to be *edge-triggered.* In this case, the processor will receive only one request regardless of how long the line is activated. Hence, there is no danger of multiple interruptions and no need to explicitly disable interrupt requests from this line.

Let us summarize **the sequence of events** involved in **handling an interrupt request from a single device.**

Assuming that interrupts are enabled, the following is a typical scenario:

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the PS (except in the case of edge-triggered interrupts).
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. The action requested by the interrupt is performed by the interrupt-service routine.
6. Interrupts are enabled and execution of the interrupted program is resumed.

5.2.3 HANDLING MULTIPLE DEVICES

Numbers of devices capable of initiating interrupts are connected to the processor. This gives rise to a number of questions:

1. How can the processor recognize the device requesting an interrupt?
2. Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?
4. How should two or more simultaneous interrupt requests be handled?

Recognizing device:

- When a request is received over the common interrupt-request line, additional information is needed to identify the particular device that activated the line.
- If two devices have activated the line at the same time, it must be possible to break the tie and select one of the two requests for service.
- When the interrupt service routine for the selected device has been completed, the second request can be serviced.
- The information needed to determine whether a device is requesting an interrupt is available in its status register.
- When a device raises an interrupt request, it sets to 1 one of the bits in its status register, which we will call the IRQ bit.
- The first device encountered with its IRQ bit set is the device that should be

serviced. Its main **disadvantage** is the time spent interrogating the IRQ bits of all the devices that may not be requesting any service. An alternative approach is to use vectored interrupts.

Starting address prediction for a service routine:

Vectored Interrupts

- A device requesting an interrupt can identify itself by sending a special code to the processor over the bus. This enables the processor to identify individual devices even if they share a single interrupt-request line.
- The code supplied by the device may represent the starting address of the interrupt-service routine for that device. The code length is typically in the range of 4 to 8 bits.
- In many computers, this is done automatically by the interrupt-handling mechanism. The location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine. The processor reads this address, called the *interrupt vector*, and loads it into the Pc. The interrupt vector may also include a new value for the processor status register.

Interrupt Nesting

- A **multiple-level priority organization** means that during execution of an interrupt-service routine, interrupt requests will be accepted from some devices but not from others, depending upon the device's priority .
- To implement this scheme, we can assign a priority level to the processor that can be changed under program control. The priority level of the processor is the priority of the program that is currently being executed.
- The processor priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the PS. These are privileged instruction, which can be executed only while the processor is running in the supervisor mode.
- A multiple-priority scheme can be implemented easily by using separate interrupt-request and interrupt-ack lines for each device as shown in fig.4.7

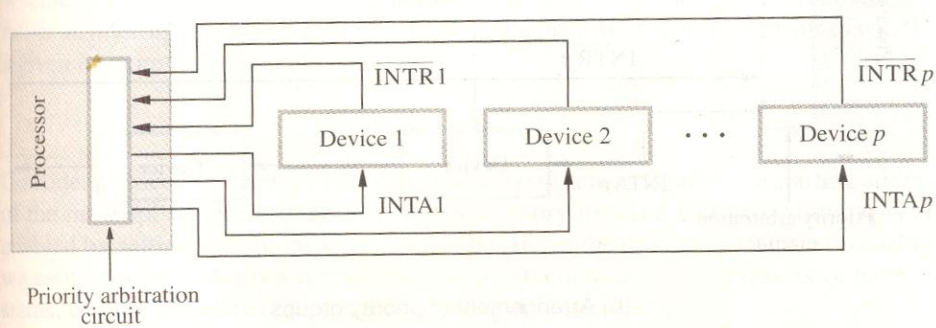

Figure 4.7 Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

- Each of the interrupt-request lines is assigned a different priority level. Interrupt request receive over these lines are sent to a priority arbitration circuit in the processor.
- A request is accepted only if it has a higher priority level than that currently assigned to the processor.

Simultaneous Requests:

- When simultaneous interrupts arrive from two or more devices the processor must device which device request must service first. A widely used scheme is to connect the devices to form a daisy chain, as shown below.

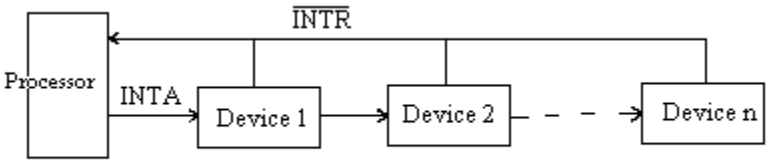

Fig: Daisy Chain

- The interrupt request line $\overline{INTR}$ is common to all devices. The interrupt-acknowledgement line, INTA is connected in a daisy-chain fashion, such that INTA signal propagates serially through the devices.
- If processor is available, when $\overline{INTR}$ line is activated it responds by setting INTA line to 1. This signal is received by device 1. Device 1 passes the signal on to the device 2 only it does not require any service. If device 1 has a pending request for interrupt, it blocks the INTA signal and proceeds to put its identifying code on the data lines.
- Therefore in daisy-chain arrangement, the device that is electrically close to the

processor has the highest priority. The second device along the chain has second highest priority and so on.

- The disadvantage is the processor accepts the interrupt request of some devices but not from others depending on the priorities. To eliminate this, the devices are organized in groups and each group is connected at a different priority level. Within a group, devices are connected in a daisy chain as shown below:

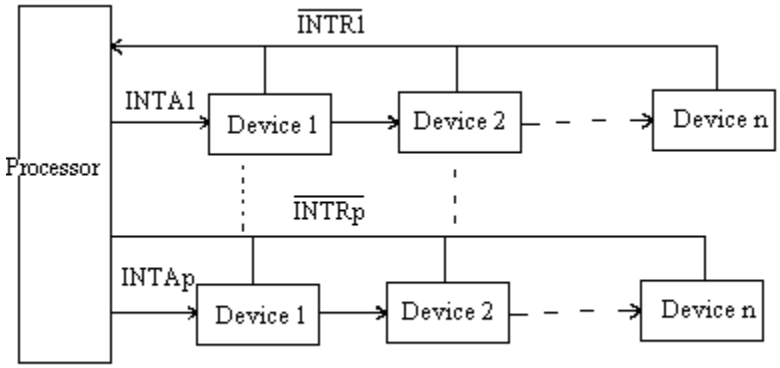

Fig: Arragement of Priority Groups

4.2.4 CONTROLLING DEVICE REQUESTS:

Idle devices must not be allowed to generate interrupt requests, even though they are ready to participate in I/O transfer operations.

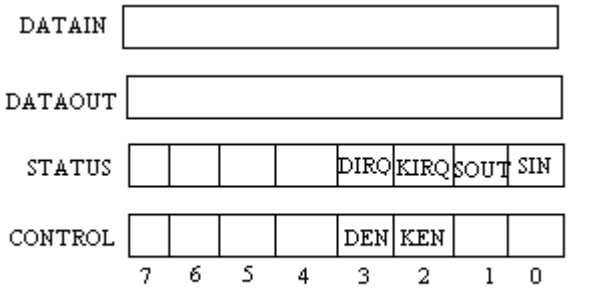

Fig: Registers in keyboard and display devices

KEN and DEN flags in the register CONTROL are the interrupt enable bit, itf either of these bits is set, the interface circuit generates an interrupt request whenever the corresponding status flag in the register STATUS is set.

The interface circuit sets bit KIRQ or DIRQ to indicate that the keyboard or display unit, respectively, is requesting an interrupt.

If the interrupt enable bit is equal to 0, the interface circuit will not generate an interrupt request, regardless of the state of the status flag.

4.2.5 EXCEPTIONS

The term exception refers to any event that causes an interruption.

Recovery from errors:

If an error occurs, the control hardware detects it and informs the processor by raising an interrupt. When processor detects an error, it suspends the program being executed and starts an exception service routine. This routine takes appropriate action to recover from the error, or to inform the user about it.

Debugging:

System software usually includes a program called a debugger, which helps the programmer to find the errors. Debugger provides two important facilities

1. Trace
2. Breakpoints

Trace mode:

- Exception occurs after every execution of every execution of instruction, using the debugging program as exception service routine.
- The debugging program enables the user to examine the contents of the registers memory locations and so on.
- On return from the debugging program, the next instruction in the program being debugged is executed, and then the debugging program is activated again.

Breakpoints Mode:

The program being debugged is interrupted only at specific points selected by the user. An instruction called Trap or Software-interrupt is usually provided for this purpose.

Privilege Exception:

To protect the operating system of a computer from being corrupted by the users programs, certain instructions can be executed only while the processor in the supervisor mode. These are called privileged instructions.

For example, when the processor is running in the user mode, it will not execute an instruction that changes the priority level of the processor or that enables a user program to access areas in the computer memory that have been allocated to other users. An attempt to execute such an instruction will produce a privilege exception, causing the processor to switch to the supervisor mode and begin executing an appropriate routine in the operating system.

5.3 DIRECT MEMORY ACCESS

A special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called **direct memory access or DMA.**

- DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a *DMA controller.*
- The DMA controller performs the functions that would normally be carried out by the processor when accessing the main memory.
- For each word transferred, it provides the memory address and all the Bus signals that control data transfer. Since it has to transfer blocks of data, the DMA controller must increment the memory address for successive words and keep track of he number of transfers.
- To initiate the transfer of a block of words, the processor sends the starting address, the number of words in the block, and the direction of the transfer. On receiving this information, the DMA controller proceeds to perform the requested operation. When the entire block has been transferred, the controller informs the processor by raising an interrupt signal.
- IO operations are always performed by the operating system of the computer in response to a request from an application program.
- The OS is also responsible for suspending the execution of one program and starting another. Thus, for an I/O operation involving DMA, the OS puts the program that requested the transfer in the Blocked state and initiates the DMA

operation, and starts the execution of another program.

- When the transfer is completed, the DMA controller informs the processor by sending an interrupt request. In response, the OS puts the suspended program in the Runnable state so that it can be selected by the scheduler to continue execution.

Figure 4.18 shows an example of the DMA controller registers that are accessed by the processor to initiate transfer operations.

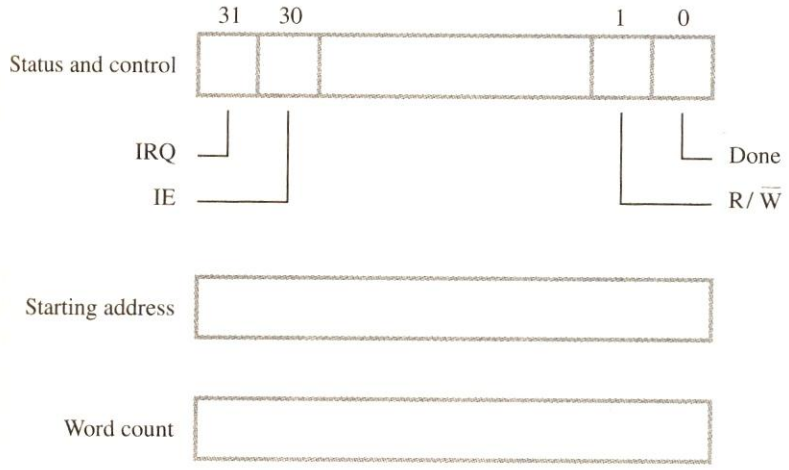

Figure 4.18 Registers in a DMA interface.

- Two registers are used for storing the starting address and the word count. The third register contains status and control flags.
- The R/W bit determines the direction of the transfer. When this bit is set to 1 by a program instruction, the controller performs a read operation, that is, it transfers data from the memory to the I/O device. Otherwise, it performs a write operation.
- When the controller has completed transferring a block of data and is ready to receive another command, it sets the Done flag to 1.
- Bit 30 is the Interrupt-enable flag, IE. When this flag is set to 1, it causes the controller to raise an interrupt after it has completed transferring a block of data. Finally, the controller sets the IRQ bit to 1 when it has requested an interrupt.

Figure 4.19 showing how DMA controllers may be used.

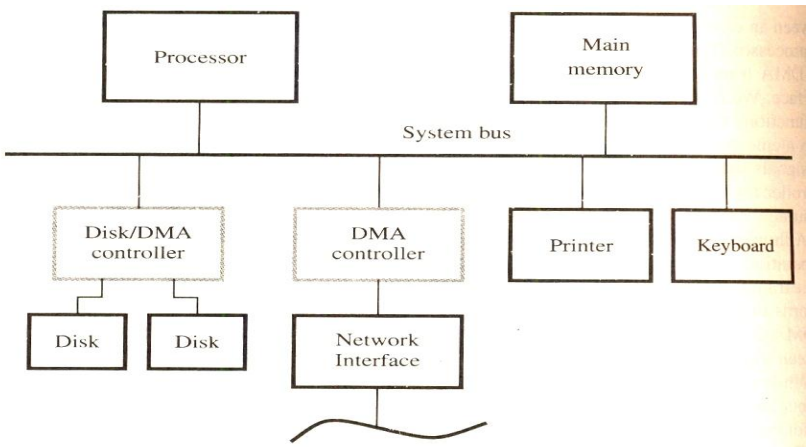

Figure 4.19 Use of DMA controllers in a computer system.

- A DMA controller connects a high-speed network to the computer bus. The disk controller, which controls two disks, also has DMA capability and provides two DMA channels.
- It can perform two independent DMA operations, as if each disk had its own DMA controller. The registers needed to store the memory address, the word count, and so on are duplicated, so that one set can be used with each device.
- To start a DMA transfer of a block of data from the main memory to one of the disks, a program writes the address and word count information into the registers of the corresponding channel of the disk controller.
- When the DMA transfer is completed, the fact is recorded in the status and control register of the DMA channel by setting the Done bit. At the same time, if the IE bit is set, the controller sends an interrupt request to the processor and sets the IRQ bit. The status register can also be used to record other information, such as whether the transfer took place correctly or errors occurred.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, or a graphics display device.
- Since the processor originates most memory access cycles, the DMA controller can be said to "steal" memory cycles from the processor. Hence, this interweaving technique is usually called *cycle stealing.* Alternatively, the DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as *block* or *burst* mode.

- A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve these conflicts, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.

5.3.1 BUS ARBITRATION

- The device that is allowed to initiate data transfers on the bus at any given time is called the ***bus master***.
- ***Bus arbitration*** is the process by which the next device to become the bus master is selected and bus mastership is transferred to it. The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus.
- There are two approaches to bus arbitration:
  - ***Centralized***: In centralized arbitration, a single *bus arbiter* performs the required arbitration
  - ***Distributed:*** In distributed arbitration, all devices participate in the selection of the next bus master.

Centralized Arbitration

- The bus arbiter may be the processor or a separate unit connected to the bus.
- Figure 4.20 illustrates a basic arrangement in which the processor contains the bus arbitration circuitry.

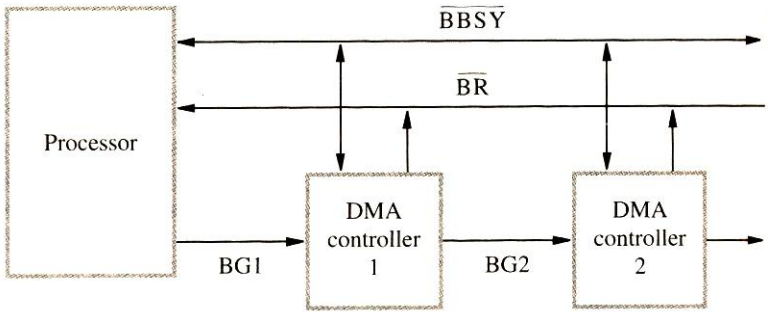

Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

- The processor is normally the bus master unless it grants bus mastership to one of the DMA controllers.

- A DMA controller indicates that it needs to become the bus master by activating the Bus-Request line, BR.
- The signal on the Bus-Request line is the logical OR of the bus requests from all the devices connected to it. When Bus-Request is activated, the processor activates the Bus-Grant signal, BG1, indicating to the DMA controllers that they may use the bus when it becomes free. This signal is connected to all DMA controllers using a daisy-chain arrangement.
- Thus, if DMA controller 1 is requesting the bus, it blocks the propagation of the grant signal to other devices. Otherwise, it passes the grant downstream by asserting BG2.
- The current bus master indicates to all devices that it is using the bus by activating another open-collector line called Bus-Busy, BBSY.
- Hence, after receiving the Bus-Grant signal, a DMA controller waits for Bus-Busy to become inactive, and then assumes mastership of the bus. At this time, it activates Bus-Busy to prevent other devices from using the bus at the same time.
- The timing diagram in Figure 4.21 shows the sequence of events for the devices as DMA controller 2 requests and acquires bus mastership and later releases the bus. During its tenure as the bus master, it may perform one or more data transfer operations, depending on whether it is operating in the cycle stealing or block model.

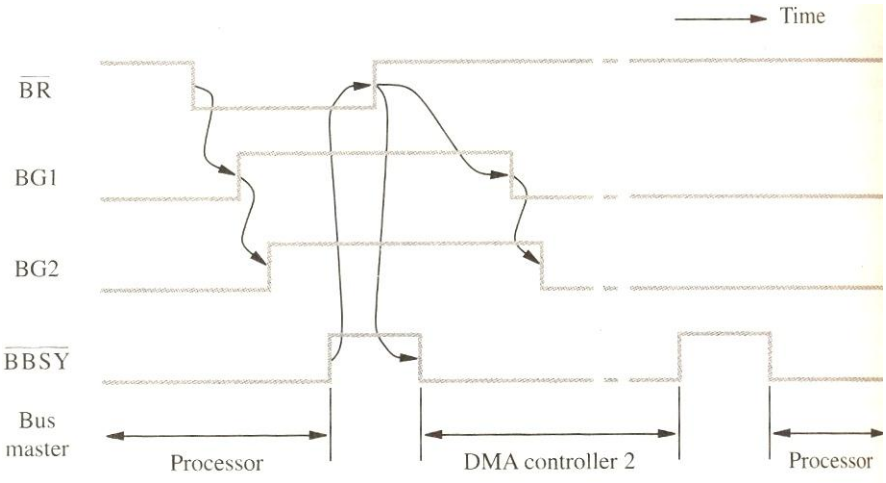

Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

- This figure shows the causal relationships among the signals involved in the arbitration process.

Distributed Arbitration

- ***Distributed arbitration*** means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter.
- A simple method for distributed arbitration is illustrated in Figure 4.22. Each device on the bus is assigned a 4-bit identification number.

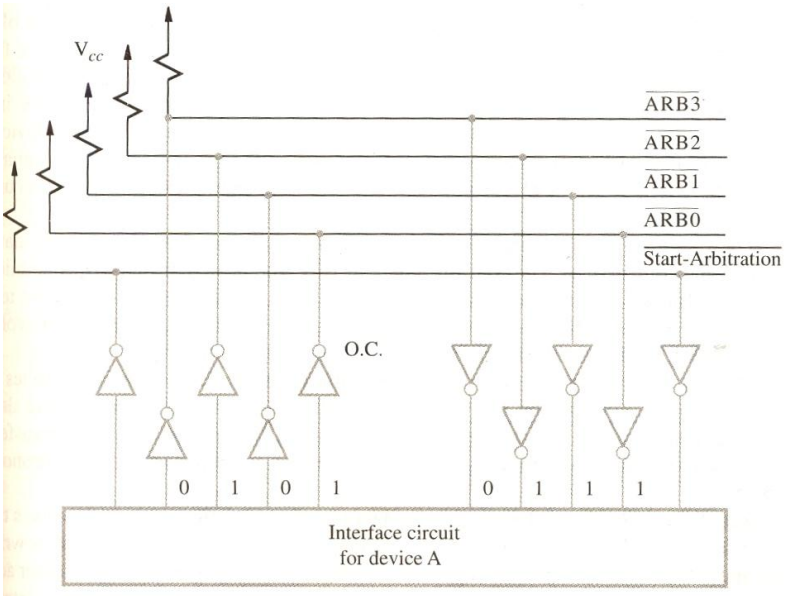

Figure 4.22 A distributed arbitration scheme.

- When one or more devices request the bus, they assert the Start-Arbitration signal and place their 4-bit ID numbers on four open-collector lines, ARB0 through ARB3.
- The net outcome is that the code on the four lines represents the request that has the highest ID number.
- The drivers are of the open-collector type. Hence, if the input to one driver is equal to one and the input to another driver connected to the same bus line is equal to 0 the bus will be in the low-voltage state. In other words, the connection performs an OR function in which logic 1 wins.

Example

Assume that two devices, A and B, having ID numbers 5 and 6, respectively, are requesting the use of the bus. Device A transmits the pattern 0101, and device B transmits the pattern 0110. The code seen by both devices is 0111. Each device compares the pattern on the arbitration lines to its own ID, starting from the most significant bit. If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower-order bits. It does so by placing a 0 at the input of these drivers. In the case of our example, device A detects a difference on line ARB 1. Hence, it disables its drivers on lines ARB 1 and ARBO. This causes the pattern on the arbitration lines to change to 0110, which means that B has won the contention. Note that, since the code on the priority lines is 0111 for a short period, device B may temporarily disable its driver on line ARBO. However, it will enable this driver again once it sees a 0 on line ARB 1 resulting from the action by device A.

- Decentralized arbitration has the advantage of offering higher reliability, because operation of the bus is not dependent on any single device.

5.4 BUSES

- The processor, main memory, and I/O devices can be interconnected by means of a common bus whose primary function is to provide a communications path for the transfer of data. The bus includes the lines needed to support interrupts and arbitration. The main features of the bus protocols used for transferring data are discussed.
- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, and so on. After describing bus protocols, we will present examples of interface circuits that use these protocols.
- The bus lines used for transferring data may be grouped into three types: ***data, address, and control lines***.
- The control signals specify whether a read or a write operation is to be performed. Usually, a single R/W line is used. It specifies Read when set to 1 and Write when set to 0. When several operand sizes are possible, such as byte, word, or long word, the required size of data is indicated.
- The bus control signals also carry timing information. They specify the times at which the processor and the I/O devices may place data on the bus or receive data from the bus. The schemes that have been devised for the timing of data transfers over a bus are broadly classified as either ***synchronous or asynchronous*** schemes.
- In any data transfer operation, one device plays the role of a ***master***. This is the device that initiates data transfers by issuing read or write commands on the bus; hence, it may be called an ***initiator***.
- Normally, the processor acts as the master, but other devices with DMA capability may also become bus masters. The device addressed by the master is referred to as a ***slave*** **or** ***target***.

5.4.1 SYNCHRONOUS BUS

In a *synchronous* bus, all devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time intervals. In the simplest form of a synchronous bus, each of these intervals constitutes a ***bus cycle*** during which one data transfer can take place. The address and data lines in this and subsequent figures are shown as high and low at the same time.

This is a common convention indicating that some lines are high and some low, depending on the particular address or data pattern being transmitted. The crossing points indicate the times at which these patterns change.

A signal line in an indeterminate or high impedance state is represented by an intermediate level half-way between the low and high signal levels.

The sequence of events during an input (read) operation.

- At time t_0, the master places the device address on the address lines and sends an appropriate command on the control lines. In this case, the command will indicate an input operation and specify the length of the operand to be read, if necessary.
- Information travels over the bus at a speed determined by its physical and electrical characteristics. The clock pulse width, $t_1 - t_0$, must be longer than the maximum propagation delay between two devices connected to the bus.
- It also has to be long enough to allow all devices to decode the address and control signals so that the addressed device (the slave) can respond at time t_1. It is important that slaves take no action or place any data on the bus before *ti*. The information on the bus is unreliable during the period t_0 to t_1 because signals are changing state.
- The addressed slave places the requested input data on the data lines at time t_1. At the end of the clock cycle, at time *t2,* the master *strobes* the data on the data lines into its input buffer. In this context, "strobe" means to capture the values of the data at a given instant and store them into a buffer.
- For data to be loaded correctly into any storage device, such as a register built with flip-flops, the data must be available at the input of that device for a period greater than the setup time of the device. Hence, the period *t2 – t1* must be

greater than the maximum propagation time on the bus plus the setup time of the input buffer register of the master.

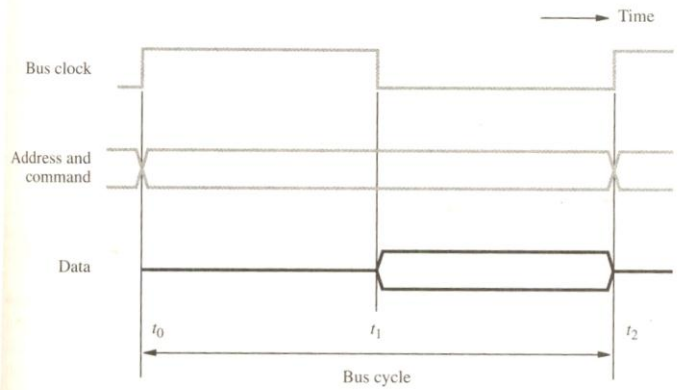

Figure 4.23 Timing of an input transfer on a synchronous bus.

A similar procedure is followed for an output operation.

- The master places the output data on the data lines when it transmits the address and command information. At time *t2*, the addressed device strobes the data lines and loads the data into its data buffer.
- The timing diagram in Figure 4.23 is an idealized representation of the actions that take place on the bus lines. The exact times at which signals actually change state are somewhat different from those shown because of propagation delays on bus wires and in the circuits of the devices.
- Figure 4.24 shows two views of each signal, except the clock. Because signals take time to travel from one device to another, a given signal transition is seen by different devices at different times. One view shows the signal as seen by the master and the other as seen by the slave.

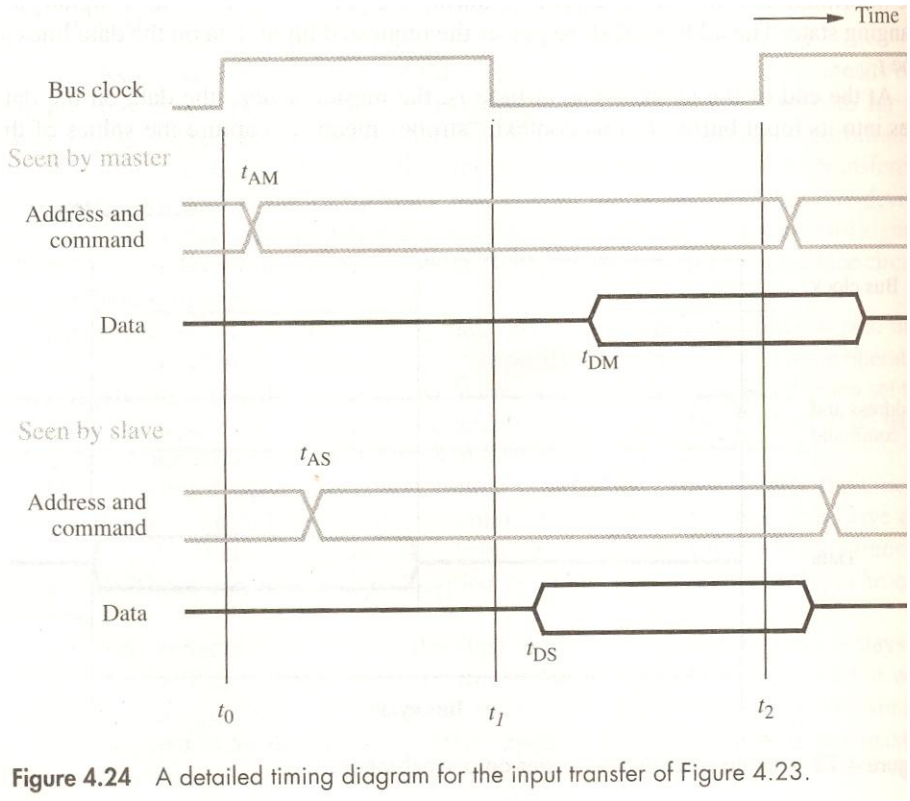

Figure 4.24 A detailed timing diagram for the input transfer of Figure 4.23.

- The master sends the address and command signals on the rising edge at the beginning of clock period 1 *(to).* However, these signals do not actually appear on the bus until *tAM,* largely due to the delay in the bus driver circuit. A while later, at *tAS,* the signals reach the slave.
- The slave decodes the address and at *t1* sends the requested data. Here again, the data signals do not appear on the bus until *tos.* They travel toward the master and arrive at *tOM.* At *t2,* the master loads the data into its input buffer. Hence the period *t2 - tOM* is the setup time for the master's input buffer. The data must continue to be valid after *t2* for a period equal to the hold time of that buffer.
- Timing diagrams in the literature often give only the simplified picture in Figure 4.23, particularly when the intent is to give a conceptual overview of how data are transferred. But, actual signals will always involve delays as shown in Figure 4.24.

Multiple-Cycle Transfers

- The scheme described above results in a simple design for the device interface.
- However, it has some limitations. Because a transfer has to be completed within

one clock cycle, the clock period, *t2 - to,* must be chosen to accommodate the longest delays on the bus and the slowest device interface. This forces all devices to operate at the speed of the slowest device.

- Also, the processor has no way of determining whether the addressed device has actually responded. It simply assumes that, at *t2,* the output data have been received by the I/O device or the input data are available on the data lines. If, because of a malfunction, the device does not respond, the error will not be detected.
- To overcome these limitations, most buses incorporate control signals that represent a response from the device. These signals inform the master that the slave has recognized its address and that it is ready to participate in a data-transfer operation. They also make it possible to adjust the duration of the data-transfer period to suit the needs of the participating devices.
- To simplify this process, a high-frequency clock signal is used such that a complete data transfer cycle would span several clock cycles. Then, the number of clock cycles involved can vary from one device to another.

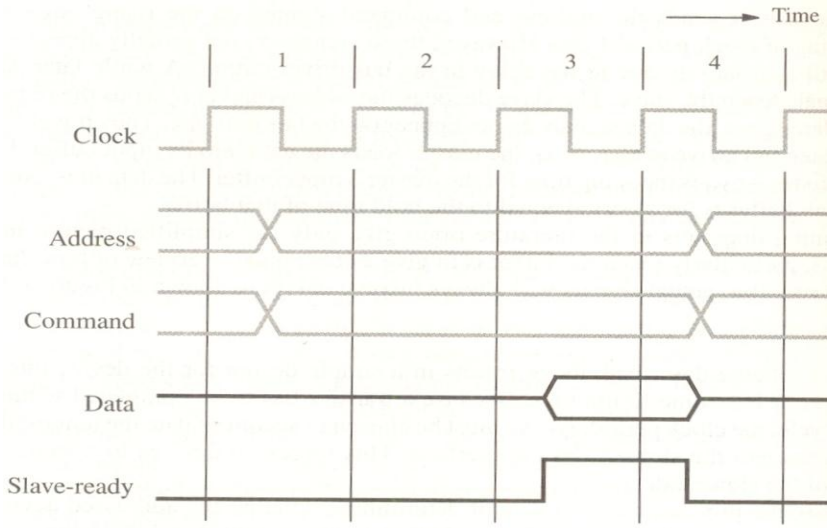

Figure 4.25 An input transfer using multiple clock cycles.

- An example of this approach is shown in Figure 4.25. During clock cycle 1, the master sends address and command information on the bus, requesting a read operation.
- The slave receives this information and decodes it. On the following active edge

of the clock, that is, at the beginning of clock cycle 2, it makes a decision to respond and begins to access the requested data.

- We have assumed that some delay is involved in getting the data, and hence the slave cannot respond immediately. The data become ready and are placed on the bus in clock cycle 3. At the same time, the slave asserts a control signal called Slave-ready.
- The master, which has been waiting for this signal, strobes the data into its input buffer at the end of clock cycle 3. The bus transfer operation is now complete, and the master may send a new address to start a new transfer in clock cycle 4.
- The Slave-ready signal is an acknowledgment from the slave to the master, confirming that valid data have been sent. In the example in Figure 4.25, the slave responds in cycle 3. Another device may respond sooner or later. The Slave-ready signal allows the duration of a bus transfer to change from one device to another. If the addressed device does not respond at all, the master waits for some predefined maximum number of clock cycles, then aborts the operation. This could be the result of an incorrect address or a device malfunction
- The delays encountered by signals internal to a chip are much less than on a bus that interconnects chips on a printed circuit board, for example. Clock frequencies are highly technology dependent. In modern processor chips, clock frequencies above 500 MHz are typical. On memory and I/O buses, the clock frequency may be in the range 50 to 150 MHz.
- Many computer buses, such as the processor buses of Pentium and ARM, use a scheme similar to that illustrated in Figure 4.25 to control the transfer of data.

5.4.2 ASYNCHRONOUS BUS

An alternative scheme for controlling data transfer on the bus is based on the use of a handshake between the master and the slave.

The concept of a handshake is a generalization of the idea of the Slave-ready signal in the fig 4.25.

The common clock is replaced by two timing control lines, Master-ready and slave-ready. The first is asserted by the master to indicate that it is ready for a transaction, and the second is a response from the slave.

In principle, a data transfer controlled by a handshake protocol proceeds as follows..
The master places the address and command information on the bus. Then it indicates to all devices that it has done so by activating the Master-ready line. This cause all devices on the bus to decode the address.
The selected slave performs the required operation and informs the processor it has done so by activating the Slave-ready line. The master waits for Slave-ready to become asserted before it removes its signals from the bus. In the case of a read operation, it also strobes the data into its input buffer.

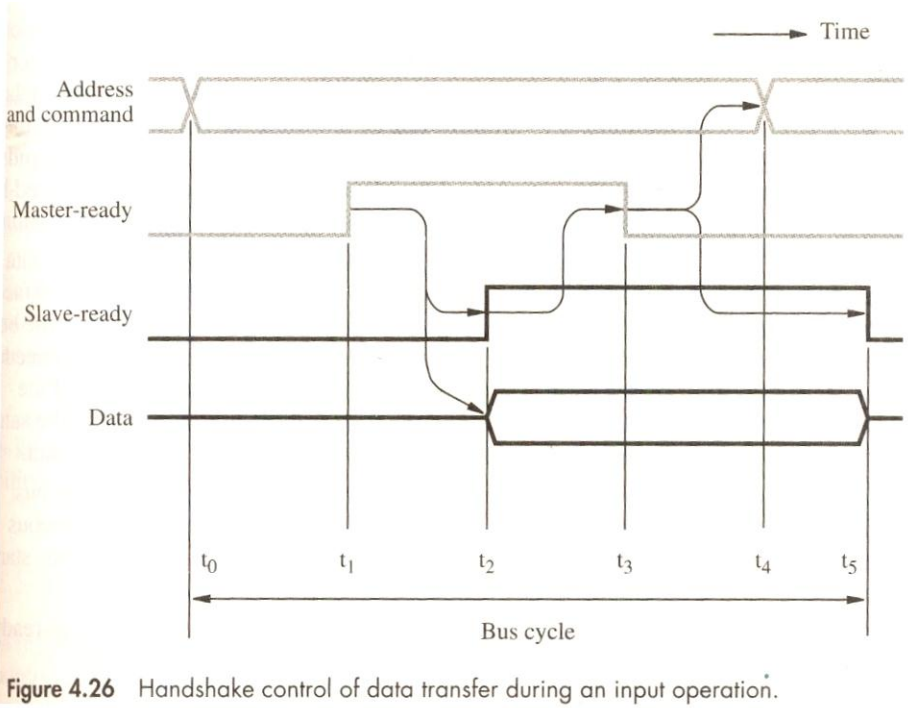

Figure 4.26 Handshake control of data transfer during an input operation.

An example of the timing of an input data transfer using the handshake scheme is give in Fig 4.26 which depicts the following sequence of events:

T0- The master places the address and command information on the bus and all devices on the bus begin to decode this information.

T1- The master sets the master-ready line to 1 to inform the I/O devices that the address and command information is ready. The delay t1-t0 is intended to allow for any skew that may occur on the bus. Skew occurs when two signals simultaneously transmitted from one source arrive at the destination at different times. This happends because

different lines of the bus may have different propagation speeds. Thus, to guarantee that the Master-ready signal does not arrive at any device ahead of the address and command information, the delay t1-t0 should be larger than the maximum possible bus skew. When the address information arrives at any device, it is decoded by the interface circuitry. Sufficient time should be allowed for the interface circuitry to decode the address. The delay needed can be included in the period t1-t0.

T2- The selected slave, having decoded the address and command information, performs the required input operation by placing the data from its data register on the data lines. The period t2-t1 depends on the distance between the master and the slave and on the delays introduced by the slave's circuitry.

T3- The slave-ready signal arrives at the master, indicating that the input data are available on the bus. After a delay equivalent to the maximum bus skew and the minimum setup time, the master strobes the data into its input buffer. At the same time, it drops the Master-ready signal., indicating that it has received the data.

T4- the Master removes the address and command information form the bus. The delay between t3 and t4 is again intended to allow for bus skew.

T5- when the device interface receives the 1 to 0 transition of the Master-ready signal, it removes the data and slave-ready signal form the bus. This completes the input transfer.

- The timing for an output operation illustrated in fig 4. 27 is essentially the same as for an input operation.
- The master places the output data on the data lines at the same time that it transmits the address and command information.
- The selected strobes the data into its output buffer when it receives the Master-ready signal and indicates that it has done so by setting the Slave-ready signal to 1.

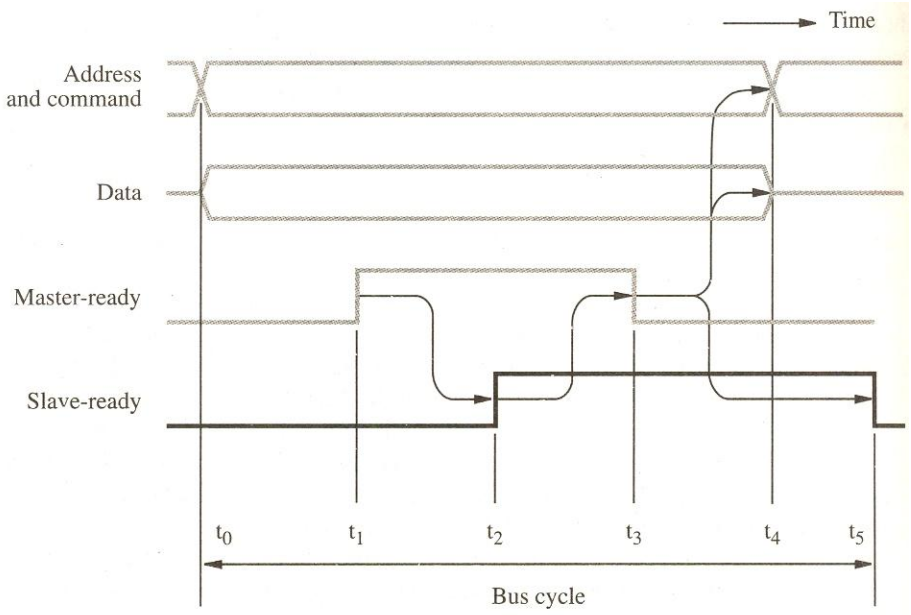

Figure 4.27 Handshake control of data transfer during an output operation.

- In the timing diagram in fig 4.26 and fig 4.27, it is assumed that the master compensates for bus skew and address decoding delay. It introduces the delays from t0-t1 and from t3-t4 for this purpose. If this delay provides sufficient time for the I/O device interface to decode the address, the interface circuit can use the Master-ready signal directly to gate other signals to or from the bus.
- The handshake signal in fig.. and fig.. are fully interlocked. A change of state in one signal is followed by a change in the other signal. Hence, this scheme is known as a full handshake. It provides the higher degree of flexibility and reliability.

5.5 INTERFACE CIRCUITS

- An I/O interface consists of the circuitry required to connect an I/O devices to a computer bus.
- On one side of the interface we have the signals for address, data and control. On the other side we have a data path with its associated control to transfer data between the interface and I/O devices. This side is called a ***port.***

Port can be classified as either a ***parallel*** or a ***serial port***.

- A ***parallel port*** transfers data in the form of a number of bits, typically 8 or 16, simultaneously to or from the device.

- A ***serial port*** transmits and receives data one bit at a time.
- Communication with the bus is the same for both formats: the conversion from the parallel to the serial format, and vice versa, takes place inside the interface circuit.
- In the case of a parallel port, the connection between the device and the computer uses a multiple-pin connector and a cable with as many wires, typically arranged in a flat configuration. The circuits at either end are relatively simple, as there is no need to convert between parallel and serial formats. This arrangement is suitable for devices that are typically close to the computer.
- For longer distances, the problem of timing skew mentioned earlier limits the data rates that can be used.
- The serial format is much more convenient and cost-effective where longer cables are needed.

Functions of an I/O interface:

1. Provides a storage buffer for at least one word of data.
2. Contains status flags that can be accessed by the processor to determine whether the buffer is full or empty.
3. Contains address-decoding circuitry to determine when it is being addressed by the processor.
4. Generates the appropriate timing signals required by the bus control scheme.
5. Performs any format conversion that may be necessary to transfer data between the us and the I/O device, such as parallel – Serial conversion in the case of a serial port.

5.5.1 PARALLEL PORT

The key aspects of interface design with a practical example.

- First, we describe circuits for an 8-bit input port and an 8-bit output port. Then, we combine two circuits to show how the interface for a general purpose 8-bit parallel port a be designed.
- We assume that the interface circuit is connected to 32-bit processor that uses memory-mapped I/O and the asynchronous bus protocol and also how the design can be modified to suit the bus protocol.

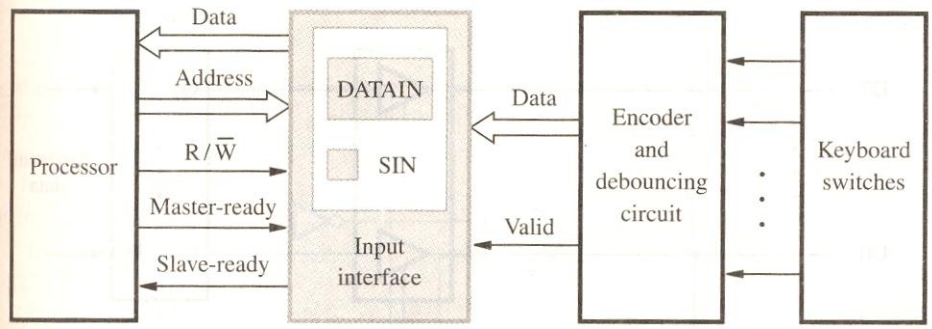

Figure 4.28 Keyboard to processor connection.

- The fig 4.28 shows the hardware components needed for connecting a keyboard to a processor. A typical keyboard consists of mechanical switches that are normally open. When a key is pressed, its switch closes and establishes a path for an electrical signal. This signal is detected by an encoder circuit that generates the ASCII code for the corresponding character. A difficultly with such push-button switches it that the contacts bounce when a key is pressed.

The effect of bouncing must be eliminated by two ways..

- A simple debouncing circuit can be included or a software approach can be used.
- When debouncing is implemented in software, the I/O routine that reads a character from the keyboard waits long enough to ensure that bouncing has subsided.

Fig 4. 28 Illustrates the hardware approach; debouncing circuits are included as a part of the encoder block.

- The output of the encoder consists of the bits that represent the encoded character and one control signal called valid, which indicates that a key is being pressed. This information is sent to the interface circuit, which contains a data register, DATAIN, and a status flag, SIN.
- When a key is pressed, the valid signal changes from 0 to 1, causing the ASCII code to be loaded into DATAIN and SIN to be set to 1. The status flag SIN is cleared to 0 when the processor reads the contents of the DATAIN register.

- The interface circuit is connected to an asynchronous bus on which transfers are controlled using handshake signals Master-ready and Salve-ready as in the fig 4.26
- The third control line, R/W distinguishes read and write transfers.

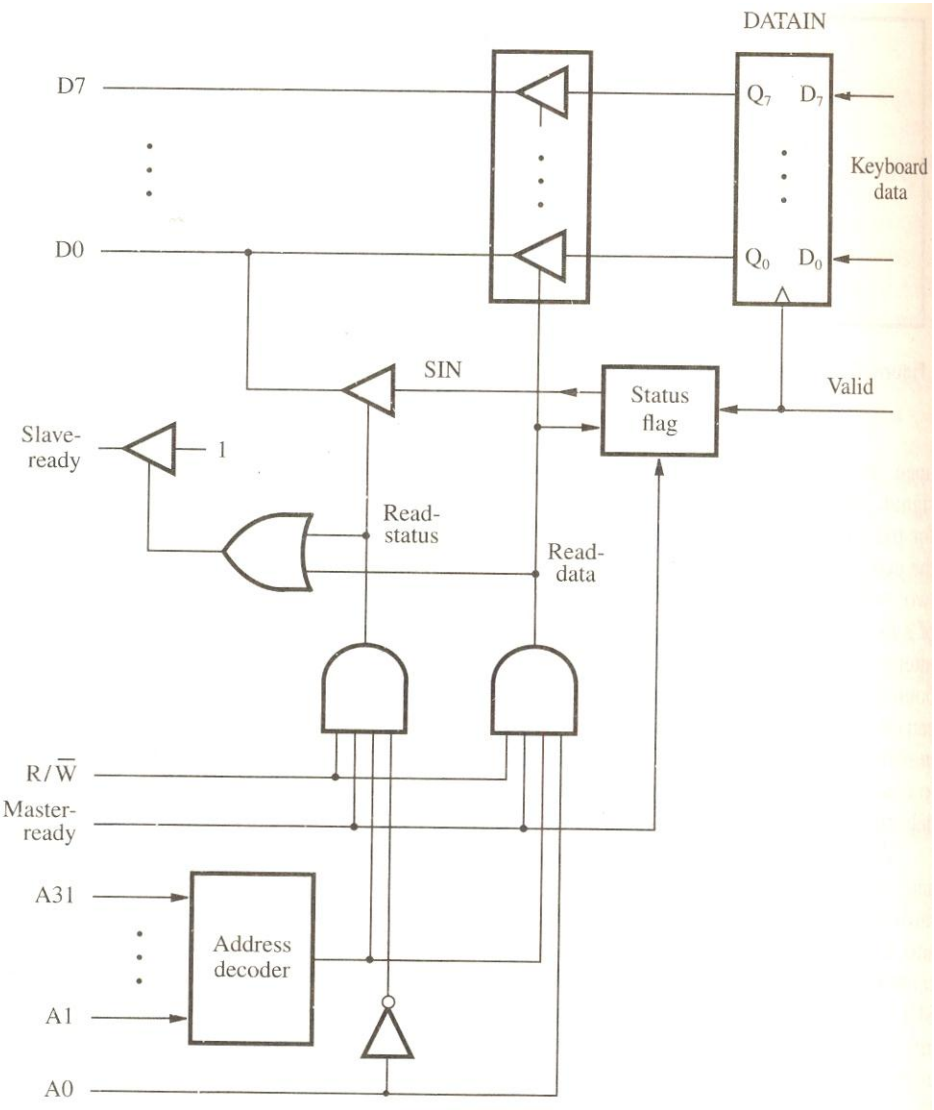

Figure 4.29 Input interface circuit.

Fig 4.29 shows a suitable circuit for an input interface.

- The output lines of the DATAIN register are connected to the data lines of the bus by means of three-state drives, which are tuned on when the processor issues a read instruction with the address that selects this register.
- The SIN signal is generated by a status flag circuit. This signal is also sent to the bus through a three-state driver. It is connected to bit D0, which means it

will appear as bit 0 of the status register. Other bits of this register do not contain valid information.

- An address decoder is used to select the input interface when the high-order 31 bits of an address correspond to any of the addresses assigned to this interface.
- Address bit A0 determines whether the status or the data registers is to read when the Master-ready signal is active. The control handshake is accomplished by activating the slave-ready signal when either Read-status or Read-data is equal to 1.

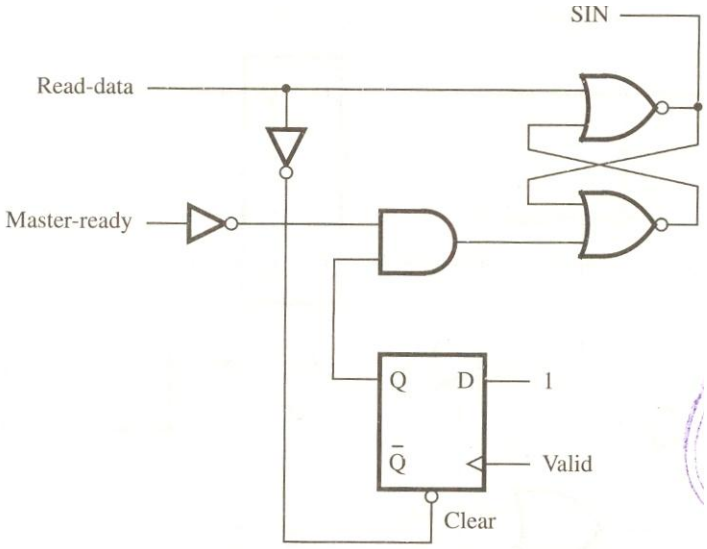

Figure 4.30 Circuit for the status flag block in Figure 4.29.

A possible implementation of the status flag circuit is shown in fig.4.30

- An edge-triggered D flip-flop is set to 1 by a rising edge on the valid signal line.
- This event changes the state of the NOR latch such that SIN is set to 1. The state of this latch must not change while SIN is being read by the processor. Hence, the circuit ensures that SIN can be set only while Master-ready is equal to 0. Both the flip-flop and the latch are reset to 0 when Read-data is set to 1 to read the DATAIN register.

Let us now consider an output interface that can be used to connect an output device, such as a printer, to a processor, as shown in Fig 4.31

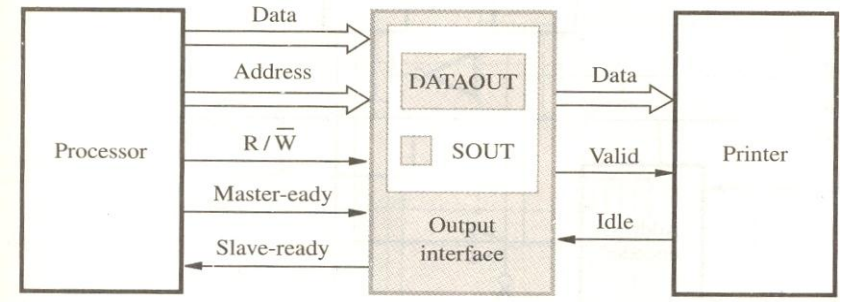

Figure 4.31 Printer to processor connection.

- The printer operates under control of the handshake signals valid and Idle in a manner similar to the handshake used on the bus with the Master-ready and Slave-ready signals. When it is ready to accept a character, the printer asserts its Idle signal. The interface circuit can then place a new character on the data lines and activate the valid signal.
- In response, the printer starts printing the new character and negates the Idle signal, which in turn causes the interface to deactivate the valid signal.
- The interface contains a data register, DATAOUT, and a status flag, SOUT. The SOUT flag is set to 1 when the printer is ready to accept another character and it is cleared to 0 when a new character is loaded into DATAOUT by the processor.

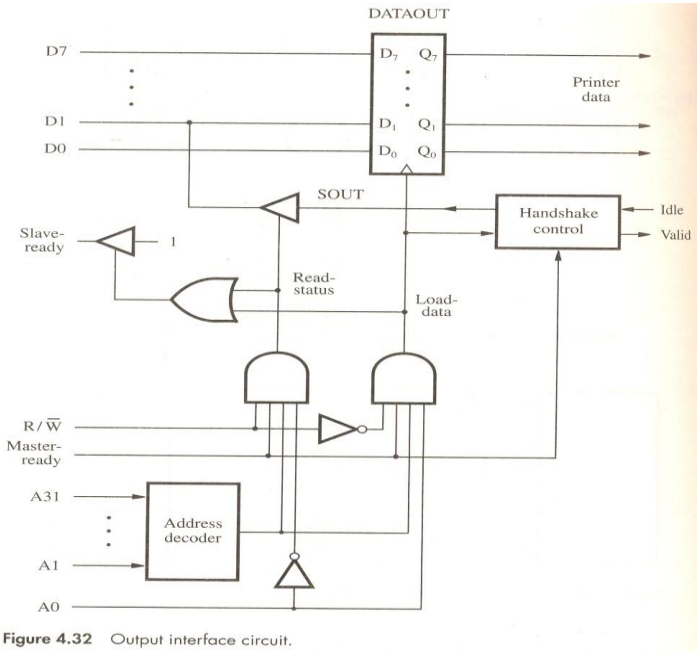

Figure 4.32 Output interface circuit.

Fig4.32 shows an implementation of this interface. Its operation is similar to the input interface of fig.4.29. the only significant difference is the handshake control circuit, the detailed design of which we leave as an exercise for the reader.

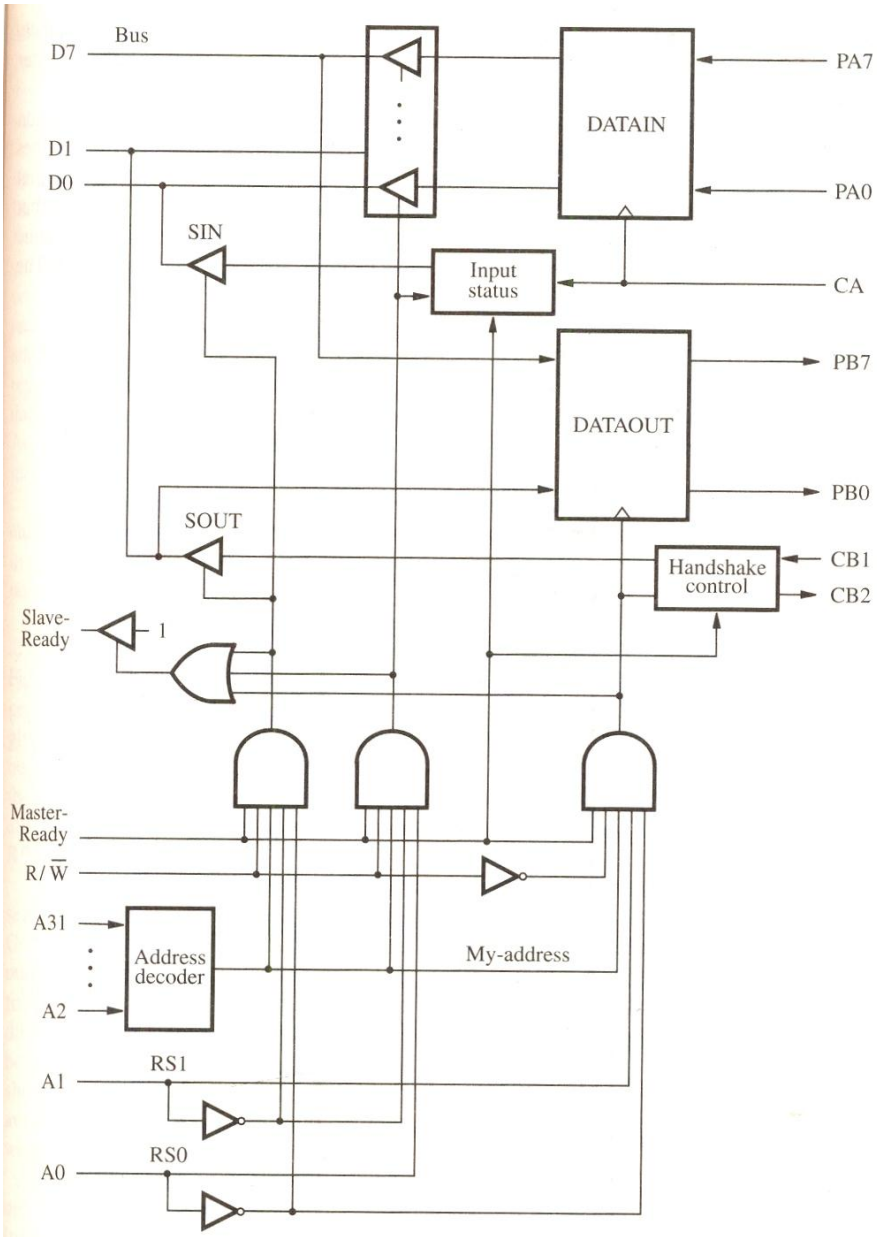

Figure 4.33 Combined input/output interface circuit.

The input and output interfaces just described can be combined into a single interface as shown in fig4.33.

Address bits A1 and A0 select one of the three addressable locations in the interface, namely, the two data registers and the status register.

The status register contains the flags SIN and SOUT in bits 0 and 1. respectively. Since such locations in I/O interfaces are often referred to as registers, we have used the labels RS1 and RS0 to denote the two inputs that determine the register being selected.

The circuit in fig 4.33 has separate input and output data lines for connection to an I/O device.

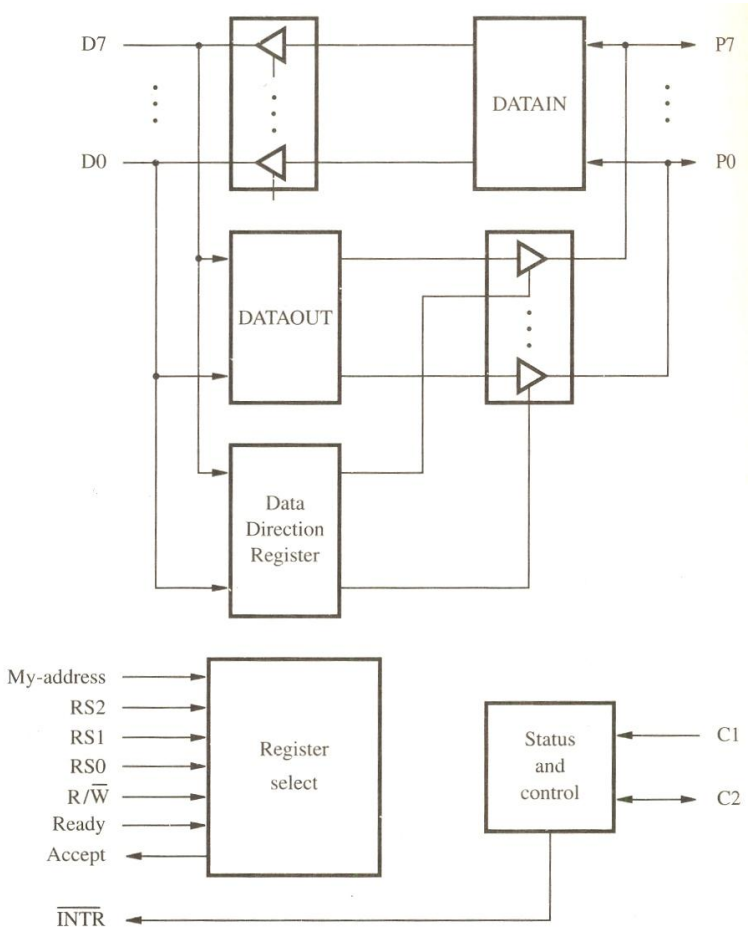

Figure 4.34 A general 8-bit parallel interface.

Fig.4.34 shows a general-purpose parallel interface circuit that cab be configured in a variety of ways. Data lines P7 through P0 can be used for either input or output purposes.

- For, increased flexibility, the circuit makes it possible for some lines to serve as inputs and some lines to serve as outputs, under program control.
- The DATAOUT register is connected to these lines via three-state drivers that are controlled by a data direction, DDR. The processor can write any 8-bit pattern into DDR.

- Two lines, C1 and C2 are provided to control the interaction between the interface circuit and the I/O device it serves. These lines are also programmable. Line C2 is bidirectional to provide several different modes of signaling, including the handshake.
- The ready and Accept lines are the handshake control lines on the processor bus side, and hence would be connected to Master-ready and Slave-ready.
- The input signal My-address should be connected to the output of an address decoder that recognizes the address assigned to the interface.
- There are three register select lines, allowing up to eight registers in the interface, input and output data, data direction and control and status registers for various modes of operation.
- An interrupt request output, INTR, is also provided. It should be connected to the interrupt-request line on the computer bus.
- Parallel interface circuits that have the features as in fig 4.34 are often encountered in practice.

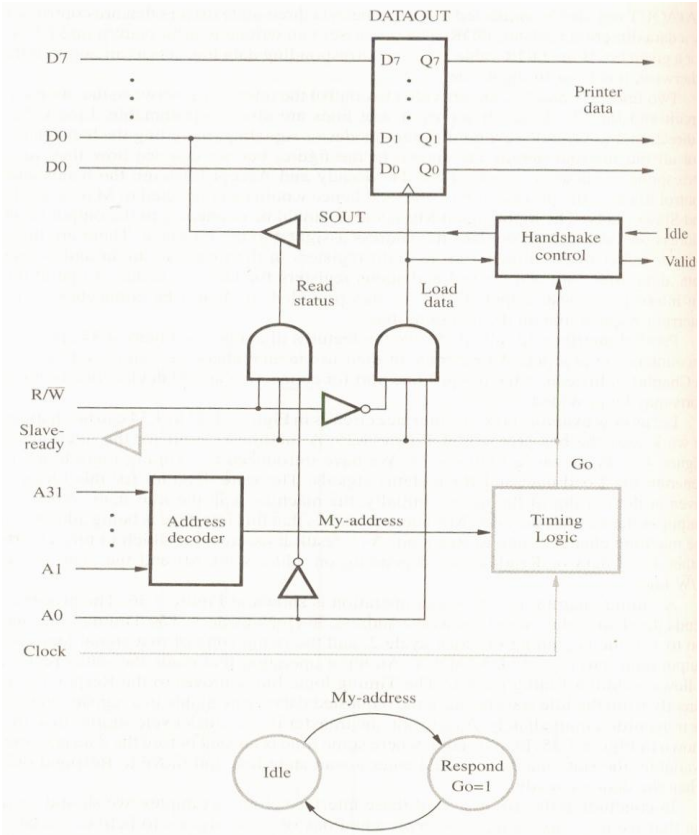

Figure 4.35 A parallel port interface for the bus of Figure 4.25, with a state-diagram for the timing logic.

- A modified circuit for the interface in fig.4.32 is shown fig.4.35. We have introduced the Timing logic block to generate the Load-data and Read-status signals.
- Initially, the machine is in the Idle state. When the output of the address decoder, My-address, shows that this interface is being addressed, the machine changes state to Respond.

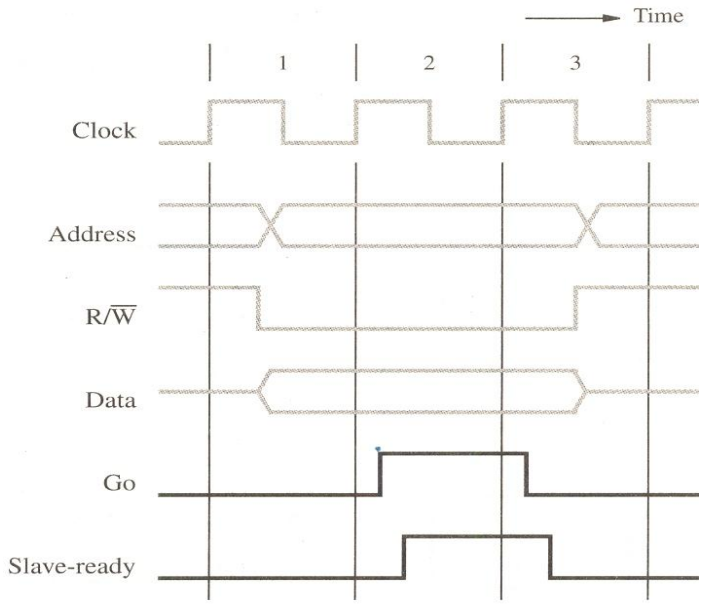

Figure 4.36 Timing for the output interface in Figure 4.35.

- A timing diagram for an output operation is shown in fig.4.36. The processor sends the data at the same time as the address, in clock cycle 1.
- The timing logic sets Go to 1 at the beginning of clock cycle 2, and the rising edge of that signal loads the output data into register DATAOUT.
- An input operation that reads the status register follows a similar timing pattern. The timing logic block moves to the respond state directly from the Idle state because the requested data are available in a register and can be transmitted immediately.
- In practice, the Slave-ready signal is likely to be an open-drain signal and would be called Slave-ready, for the same reasons as for INTR. This line must have a pull-up resistor connected to ensure that it is always in the negated state except when it is asserted by some device.

5.5.2 SERIAL PORT

- A serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.
- The key feature of an interface circuit for a serial port is that it is capable of communicating in a bit-serial fashion on the device side and in a bit-parallel fashion on the bus side.
- The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.

A block diagram of a typical serial interface is shown in fig. 4.37

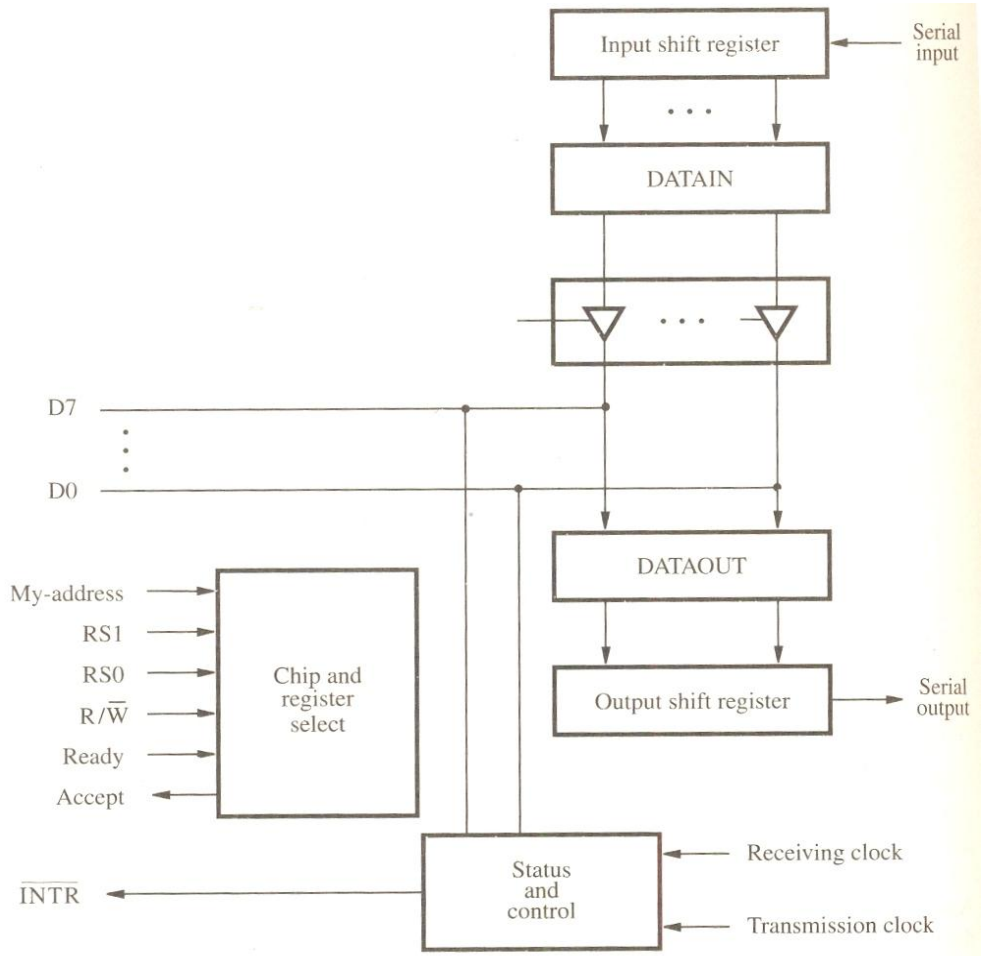

Figure 4.37 A serial interface.

- It includes the familiar DATAIN and DATAOUT registers.
- The input shift register accepts bit-serial input from the I/O device.

- When all 8 bits of data have been received, the contents of this shift register are loaded in parallel into the DATAIN register.
- Similarly, output data in the DATAOUT register are loaded into the output shift register, from which the bits are shifted out and sent to the I/O device.
- The status flags SIN and SOUT serve similar functions.
- The SIN flag is set to 1 when new data are loaded in DATAIN; it is cleared to 0 when the processor reads the contents of DATATIN.
- As soon as the data are transferred from the input shift register into the DATAIN register, the shift register can start accepting the next 8-bit character from the I/O device.
- The SOUT flag indicates whether the output buffer is available. It is cleared to 0 when the processor writes new data into the DATAOUT register and set to 1 when data are transferred from DATAOUT into the output shift register.
- A simpler interface could be implemented by turning DATAIN and DATAOUT into shift registers and eliminating the shift registers in fig.4.37.
- After receiving one character from the serial line, the device cannot start receiving the next character until the processor reads the contents of DATAIN.
- Thus, a pause would be needed between two characters to allow the processor to read the input data.
- With the double buffer, the transfer of the second character can begin as soon as the first character is loaded from the shift register into the DATAIN register.
- Thus, provided the processor reads the contents of DATAIN before the serial transfer of the second character is completed, the interface can receive a continous stream of stream data.
- An analogous situation occurs in the output path of the interface.
- Because it requires fewer wires, serial transmission is convenient for connecting devices that are physically far away from the computer.
- The speed of transmission, often given as a bit rate, depends on the nature of the devices connected.
- To accommodate a range of devices, a serial interface must be able to use a range of clock speeds.

- A standard circuit that includes the features in fig.4.37 is known as a Universal Asynchronous Receive Transmitter(UART).

5.6 STANDARD I/O INTERFACES

A different interface may have to be designed for every combination of I/O device and computer, resulting in many different interfaces.

Under this we have three widely used bus standards, PCI(Peripheral Component Interconnect), SCSI(Small Computer System Interface) and USB(Universal Serial Bus). The way these standards are used in a typical computer system is given below…

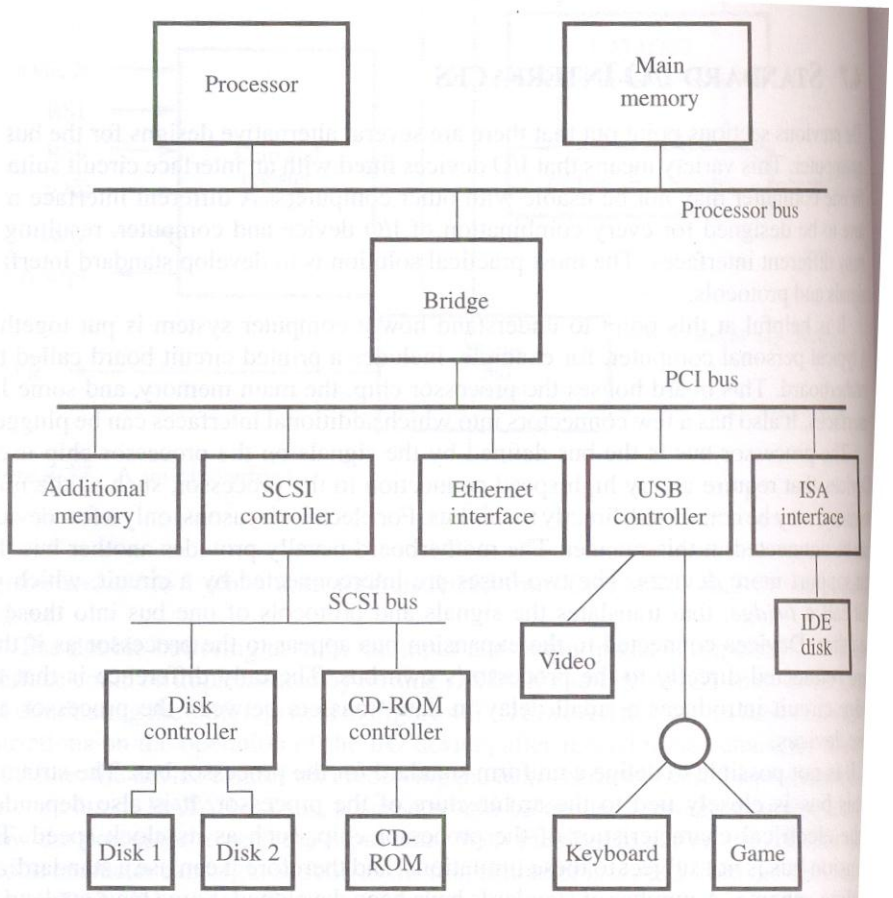

Figure 4.38 An example of a computer system using different interface standards.

PCI → Defines an expansion bus on the motherboard.

SCSI → Used for connecting additional devices, both inside and outside the computer box.

→ It is a high speed parallel bus intended for devices such as disk and video Displays.

USB → Used for connecting additional devices, both inside and outside the computer box.

→ Uses serial transmission to suit the needs of equipment ranging from keyboard to game controls to internet connections.

A Computer may use more than one bus standard. A typical Pentium computer has both a PCI bus and as ISA bus, thus providing the user with a wide range of devices to choose from.

5.6.1 Peripheral Component Interconnect

- It supports the functions found on a processor bus but in a standardized format that is independent of any particular processor.
- Devices connected to the PCI bus appear to the processor as if they were connected directly to the processor bus. They are assigned addresses in the memory address space of the processor.
- The PCI follows a sequence of bus standards that were used primarily in IBM PCs. Early PCs used the 8-bit XT bus, whose signals closely mimicked those of Intel's *80x* 86 processors. Later, the 16-bit bus used on the PC AT computers became known as the ISA bus.
- The PCI was developed as a low-cost bus that is truly processor independent. Its design anticipated a rapidly growing demand for bus bandwidth to support high-speed disks and graphic and video devices, as well as the specialized needs of multiprocessor systems.
- As a result, the PCI is still popular as an industry standard almost a decade after it was first introduced in 1992.
- An important feature that the PCI pioneered is a plug-and-play capability for con-

necting *I/O* devices. To connect a new device, the user simply connects the device interface board to the bus. The software takes care of the rest. We will discuss this feature after we describe how the PCI bus operates.

Data Transfer

- In today's computers, most memory transfers involve a burst of data rather than
- just one word.
- Data are transferred between the cache and the main memory in bursts of several words each.
- The words involved in such a transfer are stored at successive memory locations. When the processor (actually the cache controller) specifies an address and requests a read operation from the main memory.
- The memory responds by sending a sequence of data words starting at that address. Similarly, during a write operation, the processor sends a memory address followed by a sequence of data words, to be written in successive memory locations starting at that address.
- The PCI is designed to support this mode of operation. A read or write operation involving a single word is simply treated as a burst of length one.
- The bus support three independent address spaces: memory, I/O and configuration. The first two are self-explanatory
- The I/O address space is intended for use with processors. Configuration space is intended to give the PCI its plug and play capability.
- Fig 4.38 Shows the main memory of the computer connected directly to the processor bus.
- An alternative arrangement that is used often with the PCI bus is shown in fig.4.39.
- The PCI bridge provides a separate physical connection for the main memory.
- For electrical reasons, the bus may be further divided into segments connected via bridges.
- However, regardless of which bus segment a device is connected to, it may still be mapped into the processor’s memory address space.

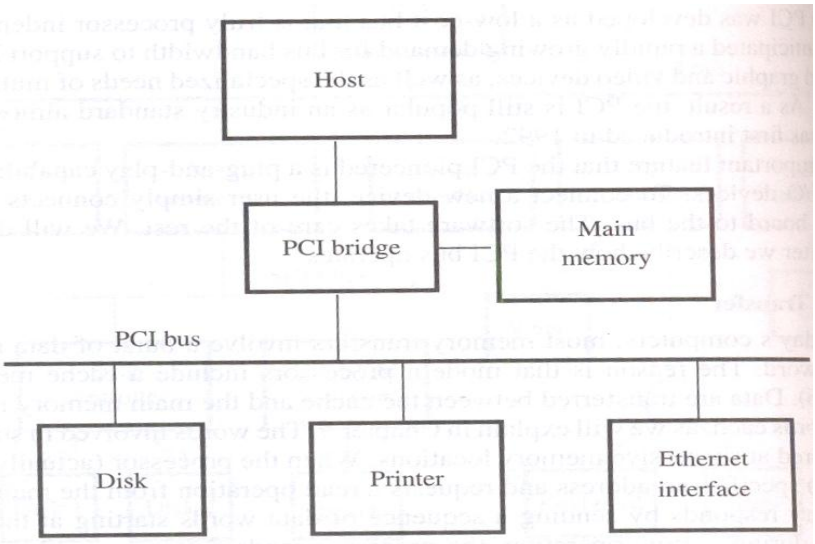

Figure 4.39 Use of a PCI bus in a computer system.

- At any given time, one device is the bus master. It has the right to initiate data transfers by issuing read and write commands.
- A master is called an *initiator* in PCI terminology. This is either a processor or a DMA controller. The addressed device that responds to read and write commands is called a *target.*

The operation of the bus and its various features

The main bus signals used for transferring data are listed in Table 4.3.

Table 4.3 Data transfer signals on the PCI bus.

Name	Function
CLK	A 33-MHz or 66-MHz clock.
FRAME#	Sent by the initiator to indicate the duration of a transaction.
AD	32 address/data lines, which may be optionally increased to 64.
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus).
IRDY#, TRDY#	Initiator-ready and Target-ready signals.
DEVSEL#	A response from the device indicating that it has recognized its address and is ready for a data transfer transaction.
IDSEL#	Initialization Device Select.

Signals whose name ends with the symbol # are asserted when in the low voltage state. PCI also uses an Initiator-ready signal, IRDY#.

Consider a bus transaction in which the processor reads four 32-bit words from the memory. In this case, *the initiator is the processor and the target is the memory.*

- A complete transfer operation on the bus, involving an address and a burst of data, is called a *transaction.*
- Individual word transfers within a transaction are called *phases.* The sequence of events on the bus is shown in Figure 4.40.
- A clock signal provides the timing reference used to coordinate different phases of a transaction. All signal transitions are triggered by the rising edge of the clock

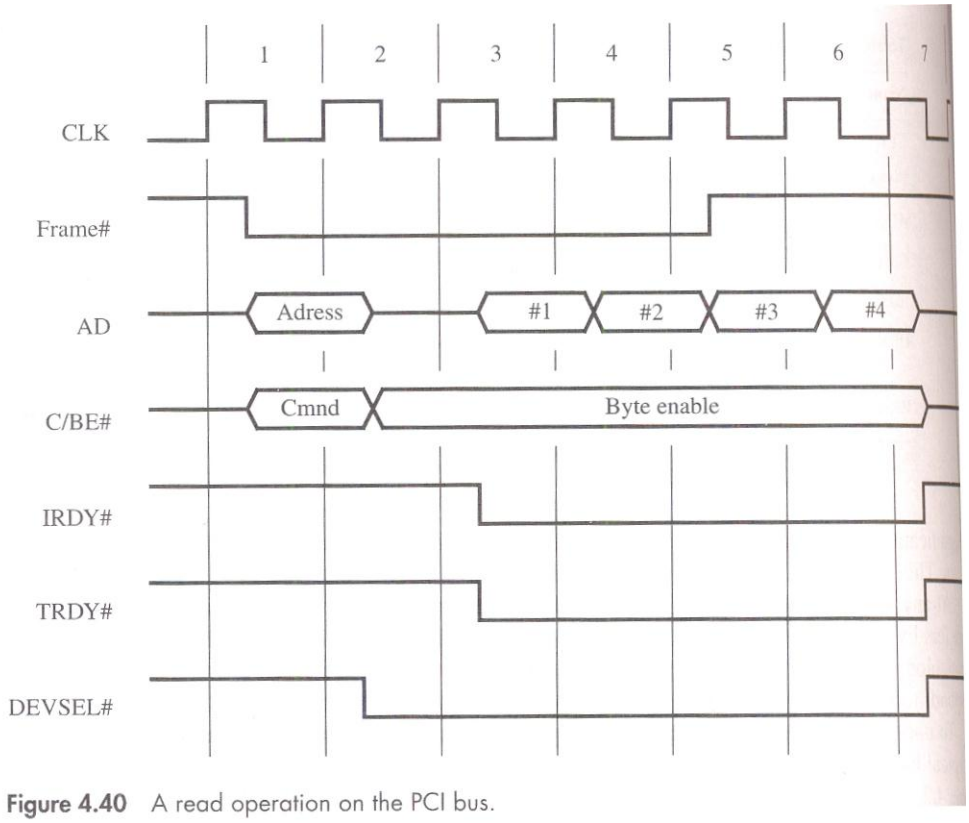

Figure 4.40 A read operation on the PCI bus.

In clock cycle 1, the processor asserts FRAME# to indicate the beginning of a transaction. At the same time, it sends the address on the AD lines and a command on the C/BE# lines. In this case, the command will indicate that a read operation is requested and that the memory address space is being used.

Clock cycle 2 is used to turn the AD bus lines around. The processor removes the address and disconnects its drivers from the AD lines. The selected target enables its drivers on

the AD lines, and fetches the requested data to be placed on the bus during clock cycle 3. It asserts DEVSEL# and maintains it in the asserted state until the end of the transaction.

The C/BE# lines, which were used to send a bus command in clock cycle1, are used for a different purpose during the rest of the transaction. Each of these four lines is associated with one byte on the AD lines. The initiator sets one or more of the C/BE# lines to indicate which byte lines are to be used for transferring data.

During clock cycle 3, he initiator asserts the initiator asserts the initiator ready signal, IRDY#, to it asserts target ready, TRDY#, and sends a word of data. The initiator loads the data into its input buffer at the end of the clock cycle. The target sends three more words of data in clock cycles 4 to 6.

The initiator uses the FRAME # signal to indicate the duration of the burst. It negates this signal during the second last word of the transfer.

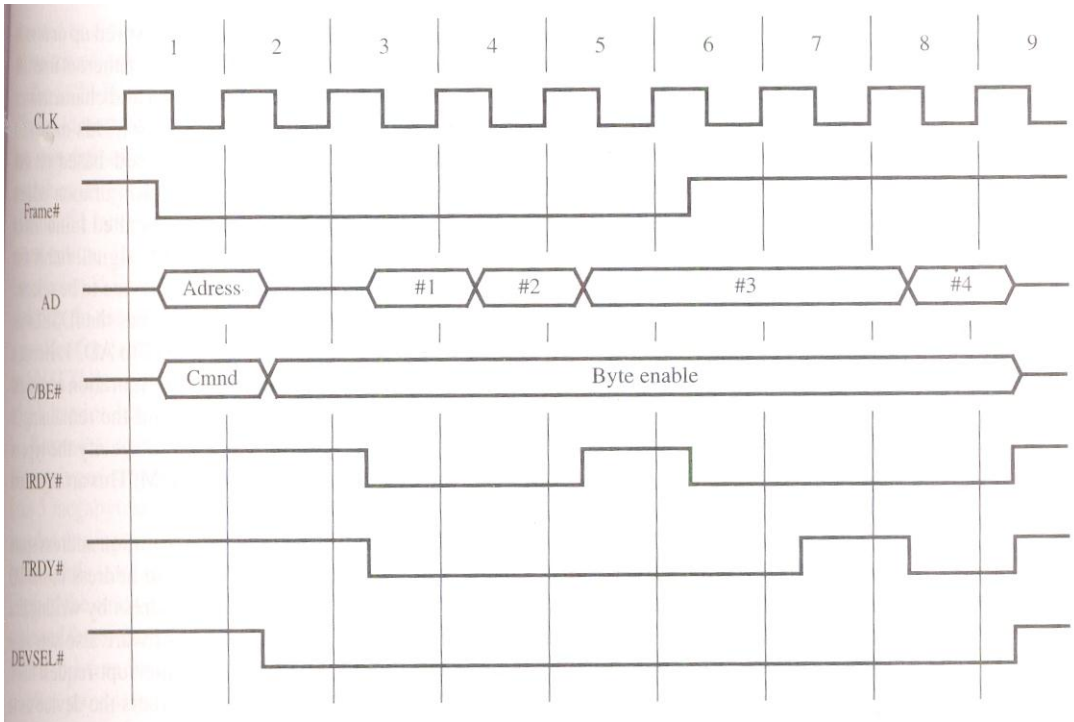

Figure 4.41 A read operation showing the role of IRDY#/TRDY#.

Fig.4.41 gives an example of a more general input transaction. It shows how the IRDY# and TRDY# signals can be used by the initiator and target, respectively, to indicate a pause in the middle of a transaction.

Device configuration

- When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it. A typical device interface card for an ISA bus, for example, has a number of jumpers or switches that have to be set by the user to select certain options.
- Once the device is connected, the software needs to know the address of the device. It may also need to know relevant device characteristics, such as the speed of the transmission link, whether parity bits are used and so on.
- The PCI initialization software reads these ROMs whenever the system is powered up or reset. In each case, it determines whether the device is a printer, a keyboard, an Ethernet interface, or a disk controller. It can further learn about various device options and characteristic.

Electrical Characteristics

The PCI bus has been defined for operation with either a 5- or 3.3 – v power supply. The motherboard may be designed to operate with either signaling system. Connectors on expansion boards are designed to ensure that they can be plugged only in a compatible motherboard.

5.6.2 SCSI BUS

- The acronym SCSI stands for Small Computer System Interface. It refers to a standard bus defined by the American National Standards Institute(ANSI).
- In the original specification of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be up to 25 meters in length and can transfer data at rates upto 5 megabytes.
- A SCSI bus may have eight data lines, in which case it is called a narrow bus and transfers data one byte at a time. Alternatively, a wide SCSI bus has 16 data lines and transfers data 16 bits at a time.

- There are also several options for the electrical signaling scheme used. The bus may use single-ended transmission (SE), where each signal uses one wire, with a common ground return for signals. In another option, differential signaling is used, where a separate return wire is provided for each signal.
- In this case, two voltage levels are possible. Earlier versions tile 5 V (TTL levels) and are known as High Voltage Differential (HVD). More recently, 3.3 V version has been introduced and is known as Low Voltage Differential (LVD).
- Because of these various options, the SCSI connector may have 50, 68, or 80 pins. The maximum transfer rate in commercial devices that are currently available varies from 5 megabytes/s to 160 megabytes.
- The maximum transfer rate on a given bus is often a function of the length of the cable and the number of devices connected, with higher rates for a shorter cable and fewer devices.
- To achieve the top data transfer rate, the bus length is typically limited to 1.6 m for SE signaling and 12 m for LVD signaling. However, manufacturers often provide special bus expanders to connect devices that are farther away.
- The maximum capacity of the bus is 8 devices for a narrow bus and 16 devices for a wide bus.
- Devices connected to the SCSI bus are not part of the address space of the processor in the same way as devices connected to the processor bus. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa.
- A packet may contain a block of data, commands from the processor to the device, or information about the device.
- To illustrate the operation of the SCSI bus, let us consider how it may be used with a disk drive. Communication with a disk drive differs substantially from communication wilh the main memory.
- Because of the constraints of the mechanical motion of the disk, there is a long delay, on the order of several milliseconds, before reaching the first sector to or from which data are to be transferred.
- Then, a burst of data is transferred at high speed. Another delay may ensue,

followed by a burst of data. A single read or write request may involve several such bursts. The SCSI protocol is designed to facilitate this mode of operation.

- A controller connected to a SCSI bus is one of two types - an *initiator* or a *target.* An initiator has the ability to select a particular target and to send commands specifying the operations to be performed.
- The disk controller operates as a target. It carries out the commands it receives from the initiator. The initiator establishes a *logical connection* with the intended target.
- Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data. This ability to overlap data transfer requests is one of the key features of the SCSI bus that leads to its high performance.
- Data transfers on the SCSI bus are always controlled by the target controller. To send a command to a target, an initiator requests control of the bus and.
- After winning arbitration, selects the controller it wants to communicate with and hands control of the bus over to it. Then the controller starts a data transfer operation to receive a command from the initiator.

Assume that the processor wishes to read a block of data from a disk drive and that these data are stored in two disk sectors that are not contiguous. The processor sends a command to the SCSI controller, which causes the following sequence of even~ to take place:

1. The SCSI controller, acting as an initiator, contends for control of the bus.
2. When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
3. The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
5. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored

in that sector and stores them in a data buffer. When it is ready to begin transferrin! data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.

6. The target transfers the contents of the data buffer to the initiator and then suspends the connection again. Data are transferred either 8 or 16 bits in parallel, depending on the width of the bus.

7. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator, a~ before. At the end of this transfer, the logical connection between the two controllers is terminated.

8. As the initiator controller receives the data, it stores them into the main *memory* using the DMA approach.

9. The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.

This scenario shows that the messages exchanged over the SCSI bus are at a higher level than those exchanged over the processor bus.

Bus signals

The bus signals are summarized in the table. All the signals are narrow bus and the names are preceded by a minus sign.

Category	Name	Function
Data	– DB(0) to – DB(7)	Data lines: Carry one byte of information during the information transfer phase and iden tify device during arbitration, selection and reselection phases
	– DB(P)	P arit y bit for the data bus
Phase	– BSY	Busy: Asserted when the bus is not free
	– SEL	Selection: Asserted during selection and reselection
Information t yp e	– C/D	Con trol/Data: Asserted during transfer of con trol information (command, status or message)
	– MSG	Message: indicates that the information being transferred is a message
Handshak e	– REQ	Request: Asserted by a target to request a data transfer cycle
	– A CK	Ac kno wledge: Asserted by the initiator when it has completed a data transfer op eration
Direction of transfer	– I/O	Input/Output: Asserted to indicate an input op eration (relativ e to the initiator)
Other	– A TN	A tten tion: Asserted by an initiator when it wishes to send a message to a target
	– RST	Reset: Causes all device con trols to disconnect from the bus and assume their start-up state

The bus has no address lines. Instead, the data lines are used to identify the bus controllers involved during the selection or reselection process and during bus arbitration.

Arbitration

In figure 4.42, we have assumed that controller 6 is an initiator that wishes to establish a connection to controller 5. After winning arbitration, controller 6 proceeds to the selection phase, in which it identifies the target.

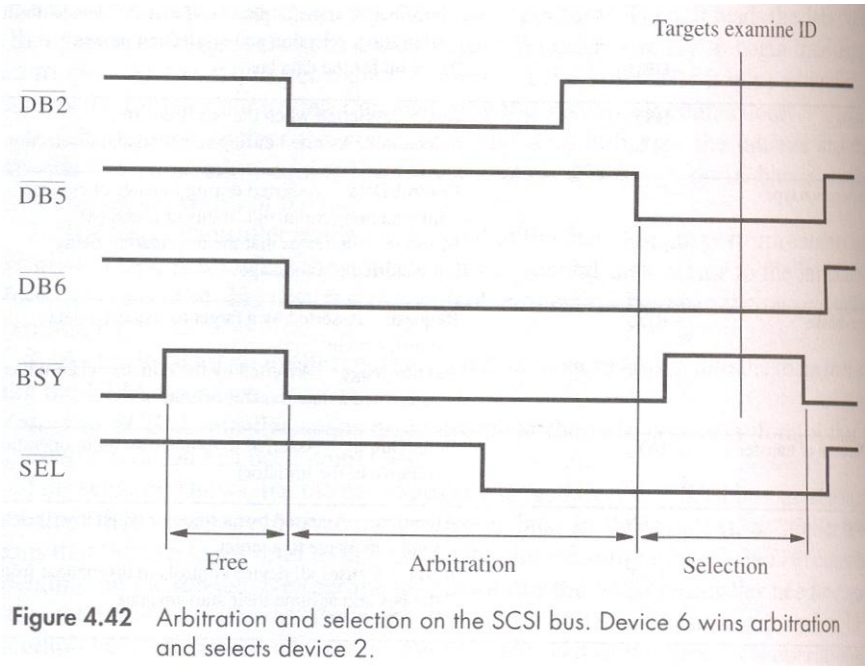

Figure 4.42 Arbitration and selection on the SCSI bus. Device 6 wins arbitration and selects device 2.

Selection

The selected target controller responds by asserting –BSY. This informs the initiator that the connection it is requesting has been established, so that it may remove the address information from the data lines. The selection process is now complete and the target controller is asserting –BSY.

Information Transfer

The information transferred between two controllers may consist of commands from the initiator to the target, status responses from the target to the initiator or being transferred to or from the I/O device. At the end of the transfer, the target controller releases the –BSY signal, thus freeing the bus for use by other devices.

Reselection

When a logical connection is suspended and the target is ready to restore it, the target must first gain control of the bus. Before data transfer can begin, the initiator must have control over to the target. This is achieved by having the target controller assert – BSY after selecting the initiator.

5.6.3 UNIVERSAL SERIAL BUS

Most computers have a wired or wireless connection to the internet. A key requirement in such an environment is the availability of a simple, low-cost mechanism to connect the devices such as keyboards, microphones, cameras, speakers & display devices to the computer. An important recent development in this regard is the introduction of the Universal Serial Bus (USB).

The USB has been designed to meet several key objectives:

- It provide a simple, low-cost, and easy to use interconnection system
- A wide range of data transfer characteristics for I/O devices.
- Enhance user conveniences through a "plug and play" mode of operation.

An objective of the USB is to make it possible to add many devices to a computer system at any time, without opening the computer box. The main characteristics that a storage device should have is data transfer rate should be high.

Plug and Play

The plug and play feature means that a new device, such as an additional speaker, can be connected at any time while the system is operating.

USB Architecture

To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure shown in fig.4.43

Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices.

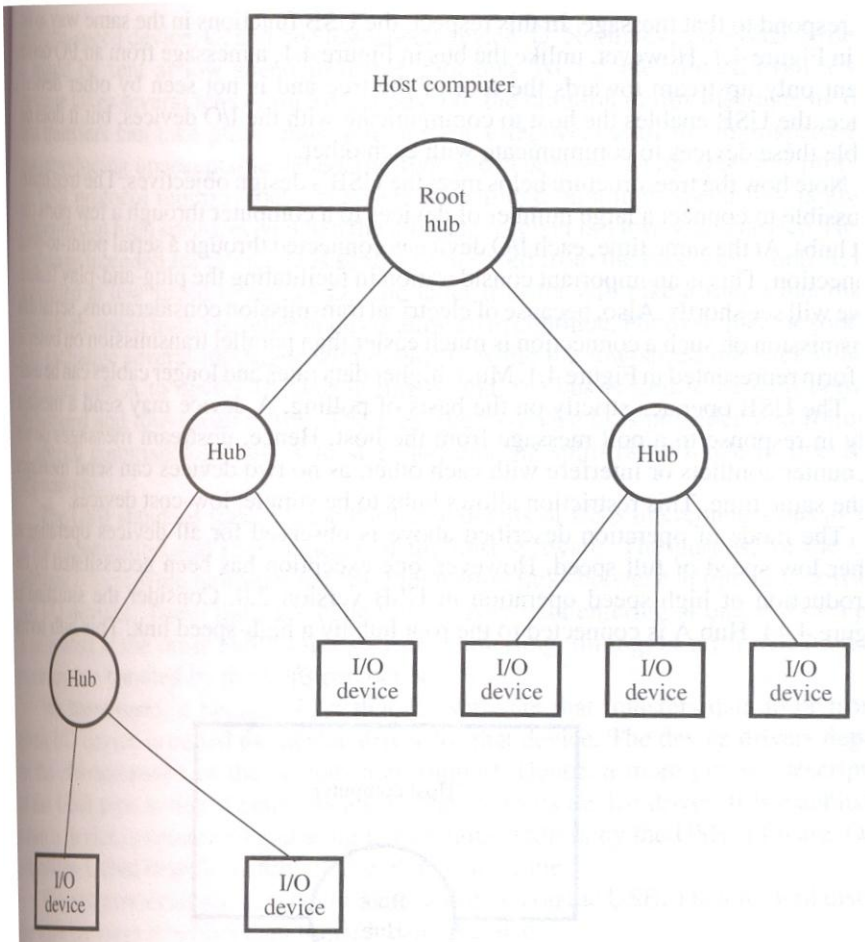

Figure 4.43 Universal Serial Bus tree structure.

- Support wide variety of devices
- Low cost
- Meet key objectives
    - Provide easy to use interconnection system with limited I/O Ports
    - Support wide range of data transfer characteristics such as for internet connections
    - Support plug-and-play mode
        - Hot swappable
- Three speeds of operation
    - Low speed – 1.5 megabits/s
    - Full-speed – 12 megabits/s

    - High-speed – 480 megabits/s
- Architecture
    - Clock + data encoded into single signal
    - USB Tree – see figure
        - Root hub
        - Hubs and I/O Devices (functions)
- Point-to-point serial links from port to I/O device

USB Addressing

Each device has a 7 bit address local to USB Tree

- On connection or power-up has address 0
- Hub detects new device and records it
- Host polls each hub and gets updated
- Host performs tasks to make device ready

Endpoints

- Locations on device to or from which data transfer can take place
- A pipe is between an endpoint and the host software
- A device can have 16 ingoing + 16 outgoing endpoints