A disk drive that incorporates the required SCSIcircuit is referred as SCSI drive.

The SCSI can transfer data at higher rate than the disk tracks.An efficient method to deal with the possible difference in transfer rate between disk and SCSI bus is accomplished by including a data buffer.This buffer is a semiconductor memory.

The data buffer can also provide cache mechanism for the disk (ie) when a read request arrives at the disk, then controller first check if the data is available in the cache(buffer).

If the data is available in the cache, it can be accessed and placed on SCSI bus . If it is not available then the data will be retrieved from the disk.

Disk Controller

The disk controller acts as interface between disk drive and system bus.

The disk controller uses DMA scheme to transfer data between disk and main memory.

When the OS initiates the transfer by issuing Read/Write request, the controllers register will load the following information. They are,

Main memory address(address of first main memory location of the block of words involved in the transfer)Disk address(The location of the sector containing the beginning of the desired block of words)(number of words in the block to be transferred).Sector header -> contains identification information.It helps to find the desired sector on the selected track.ECC (Error checking code)- used to detect and correct errors.An unformatted disk has no information on its tracks.

The formatting process divides the disk physically into tracks and sectors and this process may discover some defective sectors on all tracks.

The disk controller keeps a record of such defects.

The disk is divided into logical partitions. They are,

Primary partition

Secondary partition

In the diag, Each track has same number of sectors.

So all tracks have same storage capacity.

Thus the stored information is packed more densely on inner track than on outer track. **Access time**

There are 2 components involved in the time delay between receiving an address and the beginning of the actual data transfer. They are,

Seek time

Rotational delay / Latency

Seek time – Time required to move the read/write head to the proper track.

Latency – The amount of time that elapses after the head is positioned over the correct track until the starting position of the addressed sector passes under the read/write head.

Seek time + Latency = Disk access time

UNIT-4
INPUT/OUTPUT ORGANIZATION

4.1 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 Figure 4.1. The bus enables all the devices connected to it to exchange information. Typically, it consists of three sets of lines used to carry address, data, and control signals. Each I/O device is assigned a unique set of addresses. When the processor places a particular address on the address lines, the device that recognizes this address responds to the commands issued on the control lines. The processor requests either a read or a write operation, and the requested data are transferred over the data lines. When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.

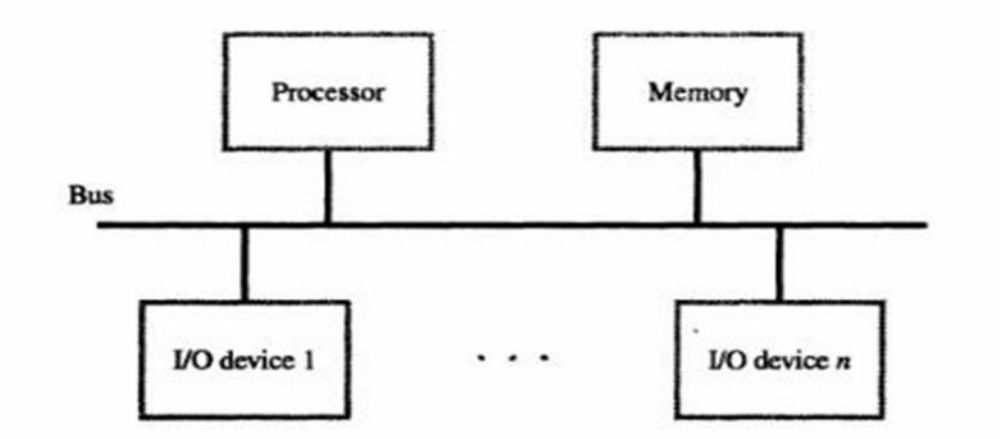


Figure 4.1: A single-bus structure

With memory-mapped I/O, any machine instruction that accesses memory can be used to transfer data to or from an I/O device. For example, if DATAIN is the address of the input buffer associated with the keyboard, the instruction

Move DATAIN, RO

reads the data from DATAIN and stores them into processor register RO. Similarly, the instruction

Move RO, DATAOUT

Sends the contents of register RO to location DATAOUT, which may be the output data buffer of a display unit or a printer.

Most computer systems use memory-mapped I/O. Some processors have special in and out instructions to perform I/O transfers. For example, processors in the Intel have special I/O instructions and a separate 16-bit address space for I/O devices. When building a computer system based on these 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.

The latter approach is by far the most common as it leads to simpler software. One advantage of a separate I/O address space is that I/O devices deal with fewer address lines. Note that 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 special 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. The I/O devices examine the low-order bits of the address bus to determine whether they should respond.

Figure 4.2 illustrates the hardware required to connect an I/O device to the bus. The address decoder enables the device to recognize its address when this address appears on the address lines. The data register holds the data being transferred to or from the processor. The 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 registers, and the control circuitry required to coordinate I/O transfers constitute the device's interface circuit.

I/O devices operate at speeds that are vastly different from that of the processor. When a human operator is entering characters at a keyboard, the processor is capable of executing millions of instructions between successive character entries. An instruction that 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 that an input character is read only once.

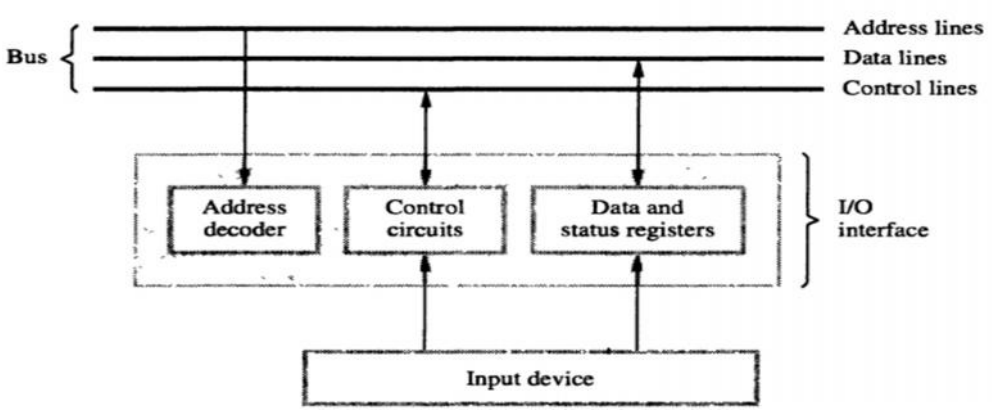


Figure 4.2: I/O interface for an input device

This example illustrates program-controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor polls the device. There are two other commonly used mechanisms for implementing I/O operations: interrupts and direct memory access. In the 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 having the device interface transfer data directly to or from the memory, without continuous involvement by the processor.

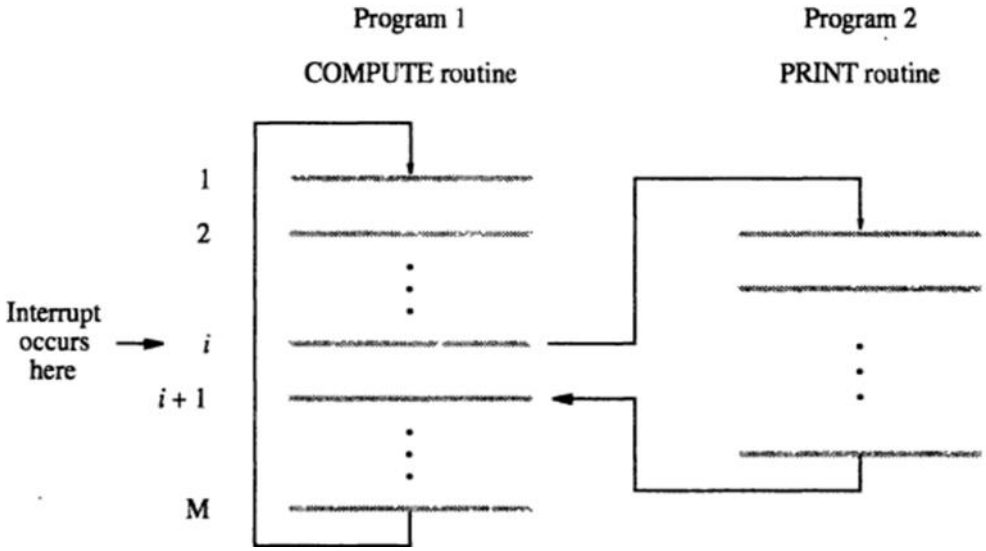
INTERRUPTS

When a program enters a wait loop, it will repeatedly check the device status. During this period, the processor will not perform any function.

The Interrupt request line will send a hardware signal called the interrupt signal to the processor.

On receiving this signal, the processor will perform the useful function during the waiting period.

The routine executed in response to an interrupt request is called the interrupt- service routine, which is the PRINT routine in our example. Interrupts bear considerable resemblance to subroutine calls. Assume that an interrupt request arrives during execution of instruction I in figure 1



Program 1
COMPUTER Routine

Program 2
PRINT routine

Figure 1. Transfer of control through the use of interrupts

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. For the time being, let us assume that this address is hardwired in the processor. After execution of the interrupt-service routine, the processor has to come back to instruction
i +1. Therefore, when an interrupt occurs, the current contents of the PC, which point to instruction i+1, must be put in temporary storage in a known location. A Return-from- interrupt instruction at the end of the interrupt-service routine reloads the PC from the temporary storage location, causing execution to resume at instruction i +1. In many processors, the return address is saved on the processor stack.

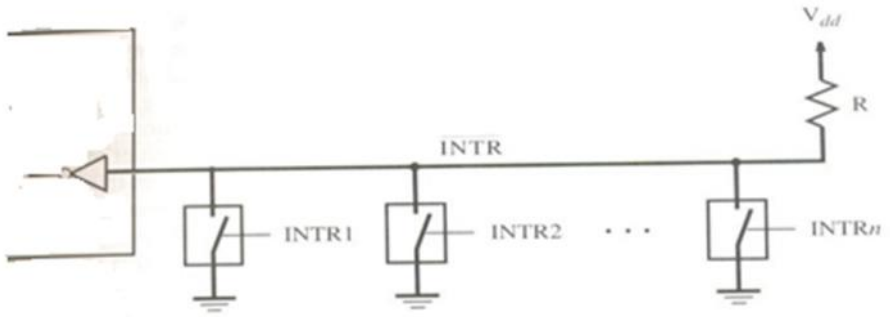
We should note that as part of handling interrupts, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal. This may be accomplished by means of a special control signal on the bus. An interrupt-acknowledge signal. The execution of an instruction in the interrupt-service routine that accesses a status or data register in the device interface implicitly informs that device that its interrupt request has been recognized.

So far, treatment of an interrupt-service routine is very similar to that of a subroutine. An important departure from this similarity should be noted. A subroutine performs a function required by the program from which it is called. However, the interrupt-service routine may not have anything in common with the program being executed at the time the interrupt request is received. In fact, the two programs often belong to different users. Therefore, before starting execution of the interrupt-service routine, any information that may be altered during the execution of that routine must be saved. This information must be restored before execution of the interrupt program is resumed. In this way, the original program can continue execution without being affected in any way by the interruption, except for the time delay. The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine.

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 registers involves memory transfers that increase the total execution time, and hence represent execution overhead. Saving registers also increase 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.

4.2 INTERRUPT HARDWARE:

We pointed out that an I/O device requests an interrupt by activating a bus line called interrupt-request. Most computers are likely to have several I/O devices that can request an interrupt. A single interrupt-request line may be used to serve n devices as



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 $INTR_1$ to $INTR_n$ 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. 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,

$$INTR = INTR_1 + \ldots\ldots\ldots + INTR_n$$

It is customary to use the complemented form **INTR,** to name the interrupt-request signal on the common line, because this signal is active when in the low-voltage state.

4.3 ENABLING AND DISABLING INTERRUPTS:

The facilities provided in a computer must give the programmer complete control over the events that take place during program execution. The arrival of an interrupt request from an external device causes the processor to suspend the execution of one program and start the execution of another. Because interrupts can arrive at any time, they may alter the sequence of events from the envisaged by the programmer. Hence, the interruption of program execution must be carefully controlled.

Let us consider in detail the specific case of a single interrupt request from one device. When a device activates the interrupt-request signal, it keeps this signal activated until it learns that the processor has accepted its request. This means that the interrupt- request signal will be active during execution of the interrupt-service routine, perhaps until an instruction is reached that accesses the device in question.

The first possibility is to have the processor hardware ignore the interrupt-request line until the execution of the first instruction of the interrupt-service routine has been completed. Then, 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, which is suitable for a simple processor with only one interrupt-request line, 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.

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.

Before proceeding to study more complex aspects of interrupts, 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.

4.4 HANDLING MULTIPLE DEVICES:

Let us now consider the situation where a number of devices capable of initiating interrupts are connected to the processor. Because these devices are operationally independent, there is no definite order in which they will generate interrupts. For example, device X may request in interrupt while an interrupt caused by device Y is being serviced, or several devices may request interrupts at exactly the same time. This gives rise to a number of questions

1. How can the processor recognize the device requesting an interrupts?
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?

The means by which these problems are resolved vary from one computer to another, And the approach taken is an important consideration in determining the computer's suitability for a given application.

When a request is received over the common interrupt-request line, additional information is needed to identify the particular device that activated the line.

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. For example, bits KIRQ and DIRQ are the interrupt request bits for the keyboard and the display, respectively. The simplest way to identify the interrupting device is to have the interrupt-service routine poll all the I/O devices connected to the bus. The first device encountered with its IRQ bit set is the device that should be serviced. An appropriate subroutine is called to provide the requested service.

The polling scheme is easy to implement. 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, which we describe next.

Vectored Interrupts:-

To reduce the time involved in the polling process, a device requesting an interrupt may identify itself directly to the processor. Then, the processor can immediately start executing the corresponding interrupt-service routine. The term vectored interrupts refers to all interrupt-handling schemes based on this approach.

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. The remainder of the address is supplied by the processor based on the area in its memory where the addresses for interrupt-service routines are located.

This arrangement implies that the interrupt-service routine for a given device must always start at the same location. The programmer can gain some flexibility by storing in this location an instruction that causes a branch to the appropriate routine.

Interrupt Nesting: -

Interrupts should be disabled during the execution of an interrupt-service routine, to ensure that a request from one device will not cause more than one interruption. The same arrangement is often used when several devices are involved, in which case execution of a given interrupt-service routine, once started, always continues to completion before the processor accepts an interrupt request from a second device. Interrupt-service routines are typically short, and the delay they may cause is acceptable for most simple devices.

For some devices, however, a long delay in responding to an interrupt request may lead to erroneous operation. Consider, for example, a computer that keeps track of the time of day using a real-time clock. This is a device that sends interrupt requests to the processor at regular intervals. For each of these requests, the processor executes a short interrupt-service routine to increment a set of counters in the memory that keep track of time in seconds, minutes, and so on.

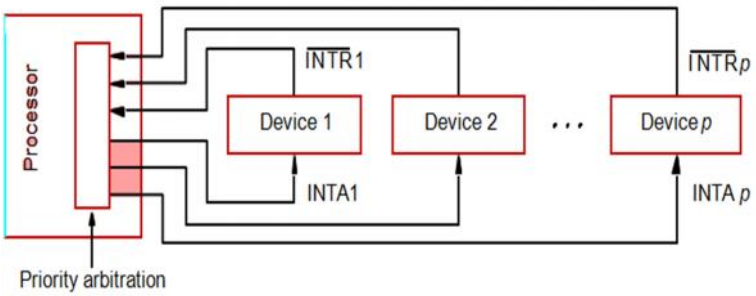
Proper operation requires that the delay in responding to an interrupt request from the real-time clock be small in comparison with the interval between two successive requests. To ensure that this requirement is satisfied in the presence of other interrupting devices, it may be necessary to accept an interrupt request from the clock during the execution of an interrupt-service routine for another device.

This example suggests that I/O devices should be organized in a priority structure. An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device.

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 accepts interrupts only from devices that have priorities higher than its own.

The processor's 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 instructions, which can be executed only while the processor is running in the supervisor mode. The processor is in the supervisor mode only when executing operating system routines. It switches to the user mode before beginning to execute application programs. Thus, a user program cannot accidentally, or intentionally, change the priority of the processor and disrupt the system's operation. An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privileged instruction.

A multiple-priority scheme can be implemented easily by using separate interrupt-request and interrupt-acknowledge lines for each device, as shown in figure. Each of the interrupt-request lines is assigned a different priority level. Interrupt requests received 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.



Priority arbitration Circuit

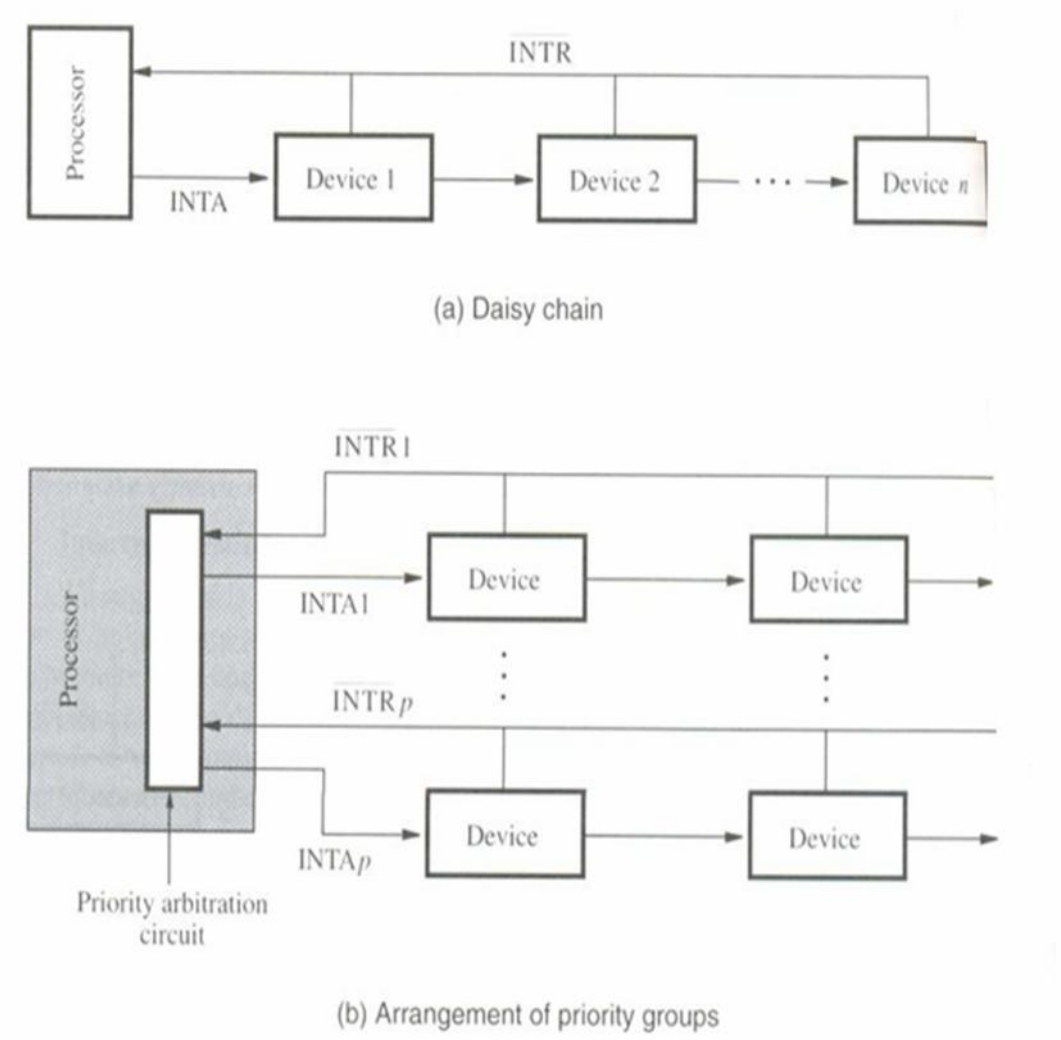
Figure2: Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

Simultaneous Requests:-

Let us now consider the problem of simultaneous arrivals of interrupt requests from two or more devices. The processor must have some means of deciding which requests to service first. Using a priority scheme such as that of figure, the solution is straightforward. The processor simply accepts the requests having the highest priority.

Polling the status registers of the I/O devices is the simplest such mechanism. In this case, priority is determined by the order in which the devices are polled. When vectored interrupts are used, we must ensure that only one device is selected to send its
interrupt vector code. A widely used scheme is to connect the devices to form a daisy chain, as shown in figure 3a. The interrupt-request line ***INTR*** is common to all devices. The interrupt-acknowledge line, INTA, is connected in a daisy-chain fashion, such that the INTA signal propagates serially through the devices.

When several d evices raise an interrupt request and the ***INTR*** line is activated the processor responds by setting the INTA line to 1. This signal is received by device 1



Device 1 passes the signal on to device 2 only if 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 the daisy-chain arrangement, the device that is electrically closest to the processor has the highest priority. The second device along the chain has second highest priority, and so on.

The scheme in figure 3.a requires considerably fewer wires than the individual connections in figure 2. The main advantage of the scheme in figure 2 is that it allows the processor to accept interrupt requests from some devices but not from others, depending upon their priorities. The two schemes may be combined to produce the more general structure in figure 3b. 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. This organization is used in many computer systems.

4.5 CONTROLLING DEVICE REQUESTS:

Until now, we have assumed that an I/O device interface generates an interrupt request whenever it is ready for an I/O transfer, for example whenever the SIN flag is 1. It is important to ensure that interrupt requests are generated only by those I/O devices that are being used by a given program. Idle devices must not be allowed to generate interrupt requests, even though they may be ready to participate in I/O transfer operations. Hence, we need a mechanism in the interface circuits of individual devices to control whether a device is allowed to generate an interrupt request.

The control needed is usually provided in the form of an interrupt-enable bit in the device's interface circuit. The keyboard interrupt-enable, KEN, and display interrupt- enable, DEN, flags in register CONTROL perform this function. If either of these flags is set, the interface circuit generates an interrupt request whenever the corresponding status flag in register STATUS is set. At the same time, the interface circuit sets bit KIRQ or DIRQ to indicate that the keyboard or display unit, respectively, is requesting an interrupt. If an interrupt-enable bit is equal to 0, the interface circuit will not generate an interrupt request, regardless of the state of the status flag.

To summarize, there are two independent mechanisms for controlling interrupt requests. At the device end, an interrupt-enable bit in a control register determines whether the device is allowed to generate an interrupt request. At the processor end, either an interrupt enable bit in the PS register or a priority structure determines whether a given interrupt request will be accepted.

4.6 EXCEPTIONS:

An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin. So far, we have dealt only with interrupts caused by requests received during I/O data transfers. However, the interrupt mechanism is used in a number of other situations.

The term exception is often used to refer to any event that causes an interruption. Hence, I/O interrupts are one example of an exception. We now describe a few other kinds of exceptions.

Recovery from Errors:

Computers use a variety of techniques to ensure that all hardware components are operating properly. For example, many computers include an error-checking code in the main memory, which allows detection of errors in the stored data. If errors occur, the control hardware detects it and informs the processor by raising an interrupt.

The processor may also interrupt a program if it detects an error or an unusual condition while executing the instructions of this program. For example, the OP-code field of an instruction may not correspond to any legal instruction, or an arithmetic instruction may attempt a division by zero.

When exception processing is initiated as a result of such errors, the processor proceeds in exactly the same manner as in the case of an I/O interrupt request. It suspends the program being executed and starts an exception-service routine. This routine takes appropriate action to recover from the error, if possible, or to inform the user about it. Recall that in the case of an I/O interrupt, the processor completes execution of the instruction in progress before accepting the interrupt. However, when an interrupt is caused by an error, execution of the interrupted instruction cannot usually be completed, and the processor begins exception processing immediately.

Debugging:

Another important type of exception is used as an aid in debugging programs. System software usually includes a program called a debugger, which helps the programmer find errors in a program. The debugger uses exceptions to provide two important facilities called trace and breakpoints.

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

Breakpoint provides a similar facility, except that 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. Execution of this instruction results in exactly the same actions as when a hardware interrupt request is received. While debugging a program, the user may wish to interrupt program execution after instruction i. The debugging routine saves instruction i+1 and replaces it with a software interrupt instruction. When the program is executed and reaches that point, it is interrupted and the debugging routine is activated. This gives the user a chance to examine memory and register contents. When the user is ready to continue executing the program being debugged, the debugging routine restores the saved instruction that was a location i+1 and executes a Return-from-interrupt instruction.

Privilege Exception:

To protect the operating system of a computer from being corrupted by user programs, certain instructions can be executed only while the processor is in 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 exceptions, causing the processor to switch to the supervisor mode and begin executing an appropriate routine in the operating system.

4.7 DIRECT MEMORY ACCESS:

A special control unit may be provided to allow the transfer of large block of data at high speed directly between the external device and main memory , without continous intervention by the processor. This approach is called **DMA.**

DMA transfers are performed by a control circuit called the **DMA Controller**. To initiate the transfer of a block of words , the processor sends,

Starting address
Number of words in the block
Direction of transfer.

When a block of data is transferred , the DMA controller increment the memory address for successive words and keep track of number of words and it also informs the processor by raising an interrupt signal.

While DMA control is taking place, the program requested the transfer cannot continue and the processor can be used to execute another program.

After DMA transfer is completed, the processor returns to the program that requested the transfer.

R/W Determines the direction of transfer . When

R/W =1, DMA controller read data from memory to I/O device.

R/W =0, DMA controller perform write operation.

Done Flag=1, the controller has completed transferring a block of data and is ready to receive another command.

IE=1, it causes the controller to raise an interrupt (interrupt Enabled) after it has completed transferring the block of data.

IRQ=1, it indicates that the controller has requested an interrupt.

Fig:Registes in a DMA Interface

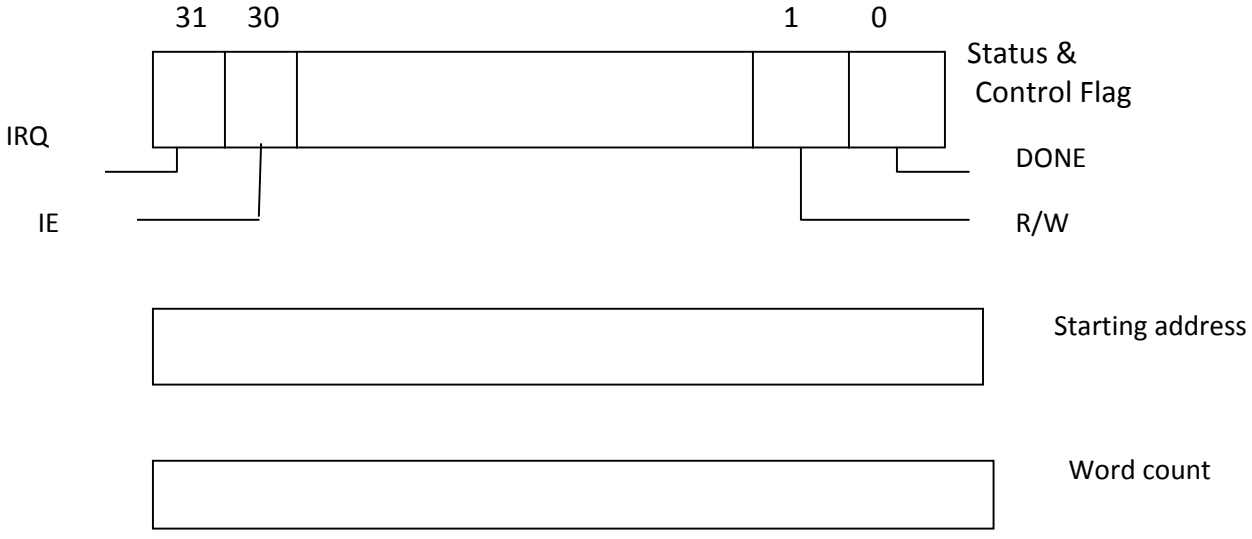
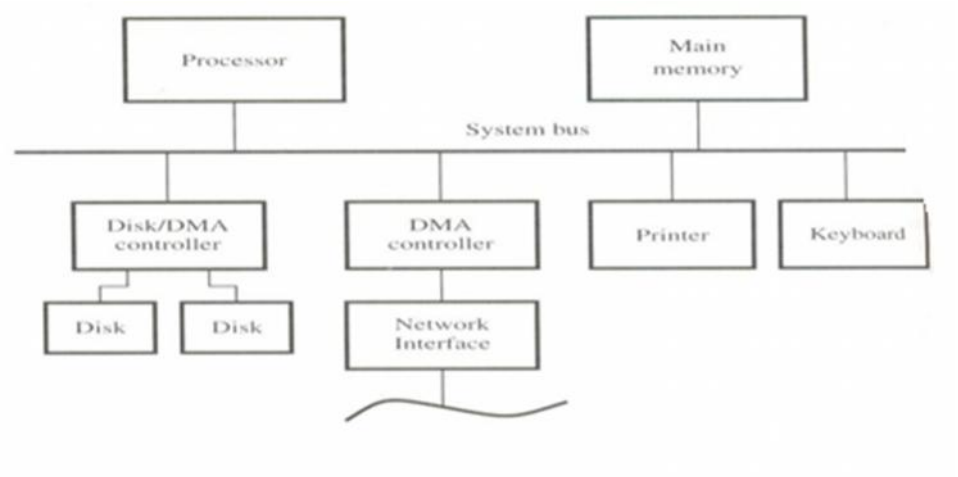


Fig: Use of DMA controllers in a computer system



A DMA controller connects a high speed network to the computer bus . The disk controller two disks, also has DMA capability and it provides two DMA channels.

To start a DMA transfer of a block of data from main memory to one of the disks, the program write s the address and the word count inf. Into the registers of the corresponding channel of the disk controller.

When DMA transfer is completed, it will be recorded in status and control registers of the DMA channel (ie) **Done bit=IRQ=IE=1**.

Cycle Stealing:

Requests by DMA devices for using the bus are having higher priority than processor requests .

Top priority is given to high speed peripherals such as , D i s k H i g h speed Network Interface and Graphics display device.

Since the processor originates most memory access cycles, the DMA controller can be said to steal the memory cycles from the processor.

This interviewing technique is called **Cycle stealing.**

burst Mode:

The DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as **Burst/Block Mode**

Bus Master:

The device that is allowed to initiate data transfers on the bus at any given time is called the bus master.

Bus Arbitration:

It is the process by which the next device to become the bus master is selected and the bus mastership is transferred to it.

Types:

There are 2 approaches to bus arbitration. They are,

Centralized arbitration (A single bus arbiter performs arbitration)

Distributed arbitration (all devices participate in the selection of next bus master).

Centralized Arbitration:

Here the processor is the bus master and it may grants bus mastership to one of its DMA controller.

A DMA controller indicates that it needs to become the bus master by activating the Bus Request line (BR) which is an open drain line.

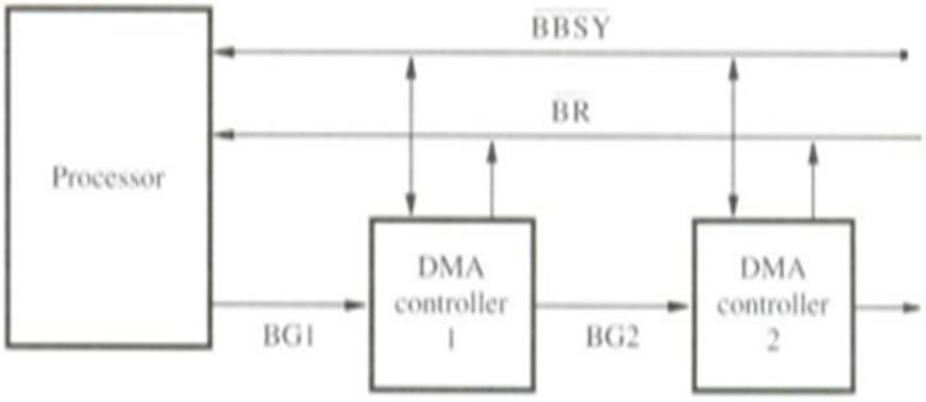
The signal on BR is the logical OR of the bus request from all devices connected to it.

When BR is activated the processor activates the Bus Grant Signal (BGI) and indicated the DMA controller that they may use the bus when it becomes free.

This signal is connected to all devices using a daisy chain arrangement.

If DMA requests the bus, it blocks the propagation of Grant Signal to other devices and it indicates to all devices that it is using the bus by activating open collector line, Bus Busy (BBSY).

Fig:A simple arrangement for bus arbitration using a daisy chain



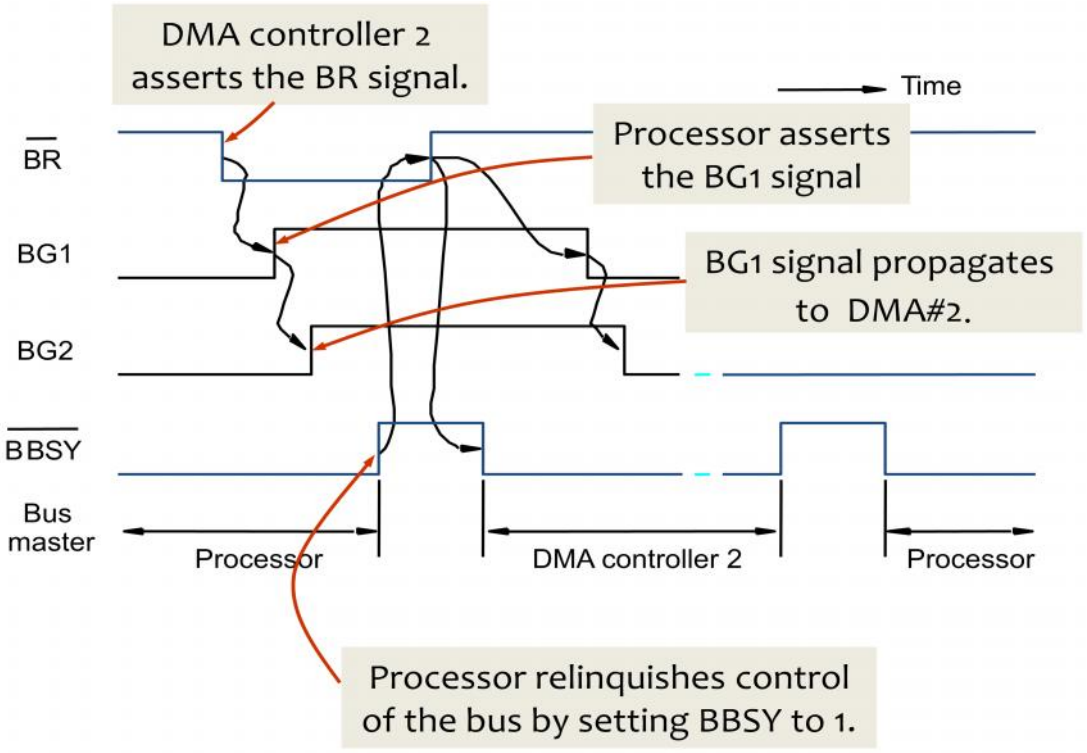


Fig: Sequence of signals during transfer of bus mastership for the devices

The timing diagram shows the sequence of events for the devices connected to the processor is shown.DMA controller 2 requests and acquires bus mastership and later releases the bus. During its tenture as bus master, it may perform one or more data transfer.

After it releases the bus, the processor resources bus mastership

Distributed Arbitration:

It means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process.

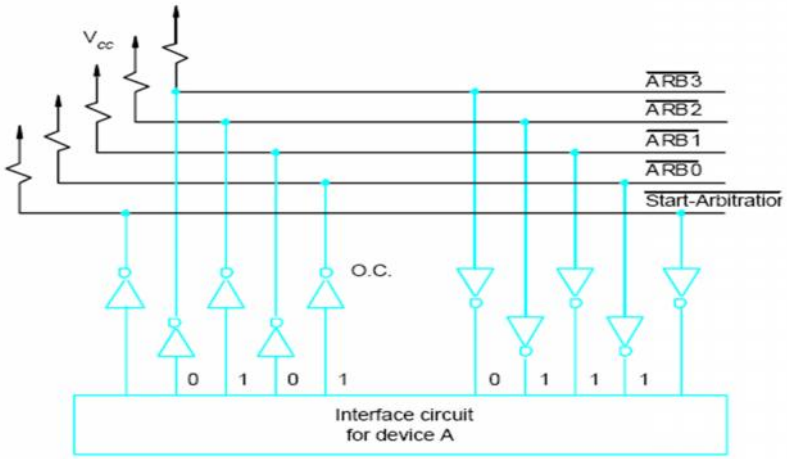


Fig:A distributed arbitration scheme

Each device on the bus is assigned a 4 bit id.
When one or more devices request the bus, they assert the Start-Arbitration signal & place their 4 bit ID number on four open collector lines, ARB0 to ARB3.
A winner is selected as a result of the interaction among the signals transmitted over these lines.
The net outcome is that the code on the four lines represents the request that has the highest ID number.
The drivers are of open collector type. Hence, if the i/p to one driver is equal to 1, the i/p to another driver connected to the same bus line is equal to „0"(ie. bus the is in low-voltage state).
Example: Assume two devices A & B have their ID 5 (0101), 6(0110) and their code is0111.
Each devices compares the pattern on the arbitration line to its own ID starting from MSB.
If it detects a difference at any bit position, it disables the drivers at that bit position. It does this by placing „0" at the i/p of these drivers.
In our eg. „A" detects a difference in line ARB1, hence it disables the drivers onlines ARB1 & ARB0.This causes the pattern on the arbitration line to change to 0110 which means that„B" has won the contention.

4.8 Buses

A bus protocol is the set of rules that govern the behavior of various devices connected to the bus ie, when to place information in the bus, assert control signals etc.

The bus lines used for transferring data is grouped into 3 types. They are,

- Address line
- Data line
- Control line.

Control signals Specifies that whether read / write operation has to performed.
It also carries timing infn/. (ie) they specify the time at which the processor & I/O devices place the data on the bus & receive the data from the bus.

During data transfer operation, one device plays the role of a „**Master**".
Master device initiates the data transfer by issuing read / write command on the bus. Hence it is also called as „**Initiator**".

The device addressed by the master is called as **Slave / Target**.

Types of Buses:
There are 2 types of buses. They are,

- Synchronous Bus
- Asynchronous Bus.

4.8.1 Synchronous Bus:-

In synchronous bus, all devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time.

During a „**bus cycle**", one data transfer on take place.

The „**crossing points**" indicate the tone at which the patterns change.

A „**signal line'** in an indeterminate / high impedance state is represented by an intermediate half way between the low to high signal levels.

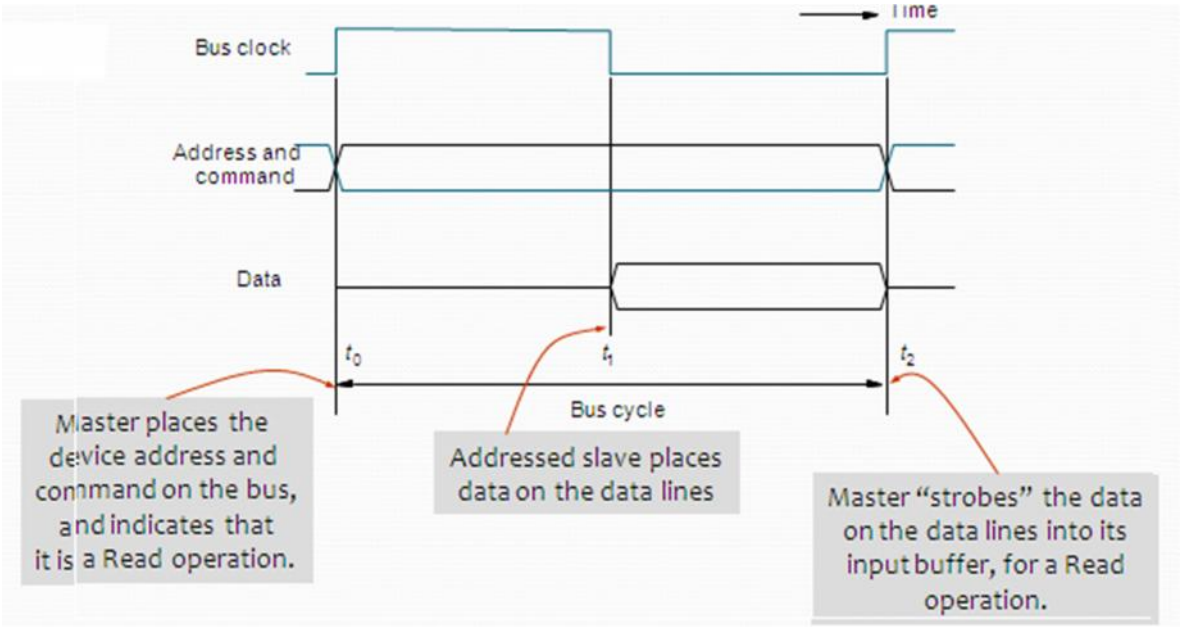


Fig:Timing of an input transfer of a Synchronous bus

At time t0 master places address on address lines & sends an appropriate command on the control lines.

In this case, the command will indicate an input operation & specify the length of the operand to be read.

The clock pulse width t1 – t0 must be longer than the maximum delay between devices connected to the bus.

The clock pulse width should be long to allow the devices to decode the address & control signals so that the addressed device can respond at time t_1. The slaves take no action or place any data on the bus before t_1.

- At the end of the clock cycle, at time t_2, the master strobes the data on the data lines into its input buffer if it's a Read operation.
- "Strobe" means to capture the values of the data and store them into a buffer.
- When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.
- Width of the pulse $t_2 - t_1$ should be longer than:

- Maximum propagation time of the bus plus
- Set up time of the input buffer register of the master.

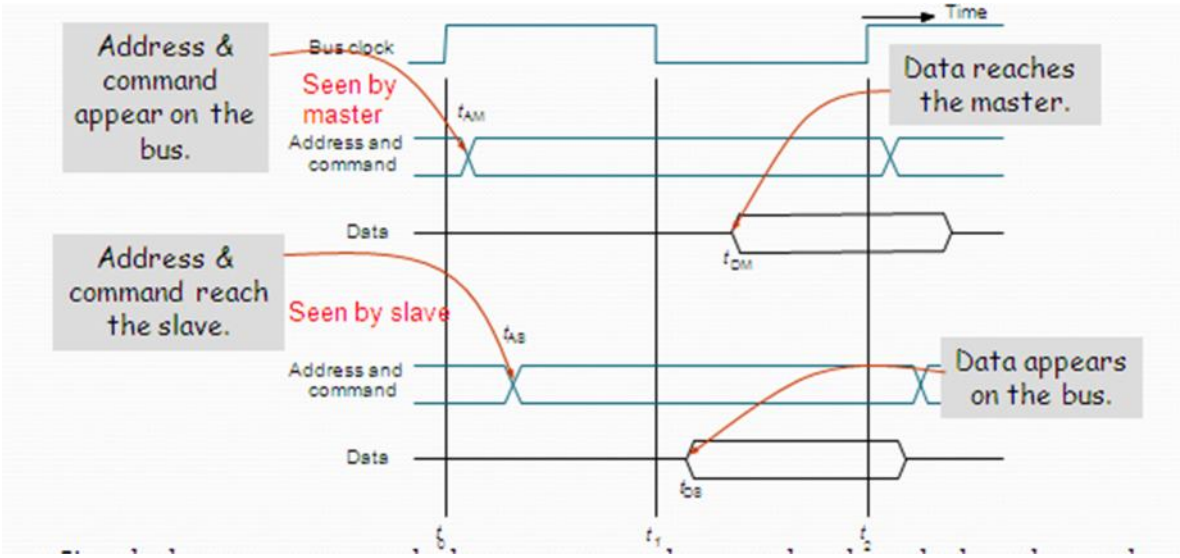


Fig:A detailed timing diagram for the input transfer

The picture shows two views of the signal except the clock.
One view shows the signal seen by the master & the other is seen by the salve. The master sends the address & command signals on the rising edge at the beginning of clock period (t_0). These signals do not actually appear on the bus until t_{am}.

Signals do not appear on the bus as soon as they are placed on the bus, due to the propagation delay in the interface circuits.
Signals reach the devices after a propagation delay which depends on the characteristics of the bus.
Data must remain on the bus for some time after t_2 equal to the hold time of the buffer.

Data transfer has to be completed within one clock cycle.
Clock period t2 - t0 must be such that the longest propagation delay on the bus and the slowest device interface must be accommodated.
Forces all the devices to operate at the speed of the slowest device.
Processor just assumes that the data are available at t2 in case of a Read operation, or are read by the device in case of a Write operation.
What if the device is actually failed, and never really responded?
Most buses have control signals to represent a response from the slave.

- Control signals serve two purposes:
  - Inform the master that the slave has recognized the address, and is ready to participate in a data transfer operation.
  - Enable to adjust the duration of the data transfer operation based on the speed of the participating slaves.
- High-frequency bus clock is used:

- Data transfer spans several clock cycles instead of just one clock cycle as in the earlier case.

Multiple Cycle Transfer:-

During, clock cycle1, the master sends address & cmd infn/. On the bus‟ requesting a „read‟ operation.

The slave receives this information & decodes it.

At the active edge of the clock (ie) the beginning of clock cycel2, it makes accession to respond immediately.

The data become ready & are placed in the bus at clock cycle3.

At the same times, the slave asserts a control signal called „slave-ready‟. The master which has been waiting for this signal, strobes, the data to its i/p buffer at the end of clock cycle3.

The bus transfer operation is now complete & the master sends a new address to start a new transfer in clock cycle4.

The „slave-ready‟ signal is an acknowledgement form the slave to the master confirming that valid data has been sent.

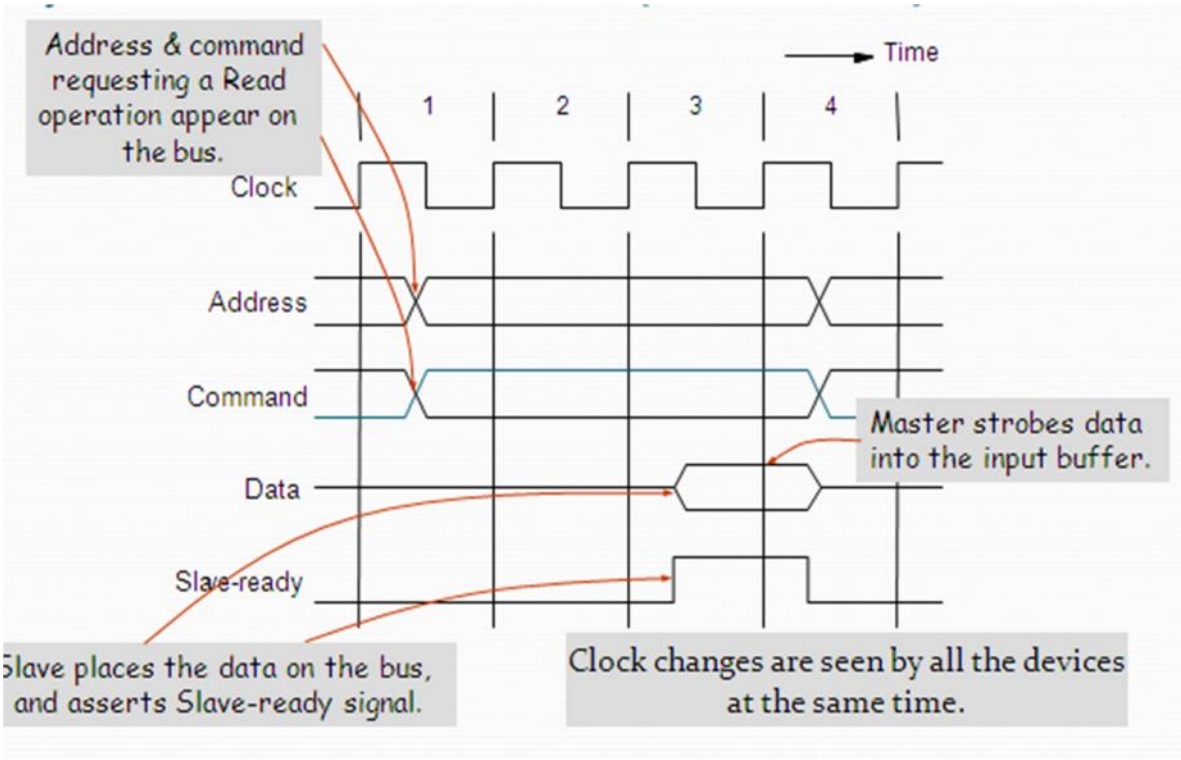


Fig:An input transfer using multiple clock cycles

4.8.2 Asynchronous Bus:-

An alternate scheme for controlling data transfer on the bus is based on the use of „handshake" between Master & the Slave. The common clock is replaced by two timing control lines.

They are

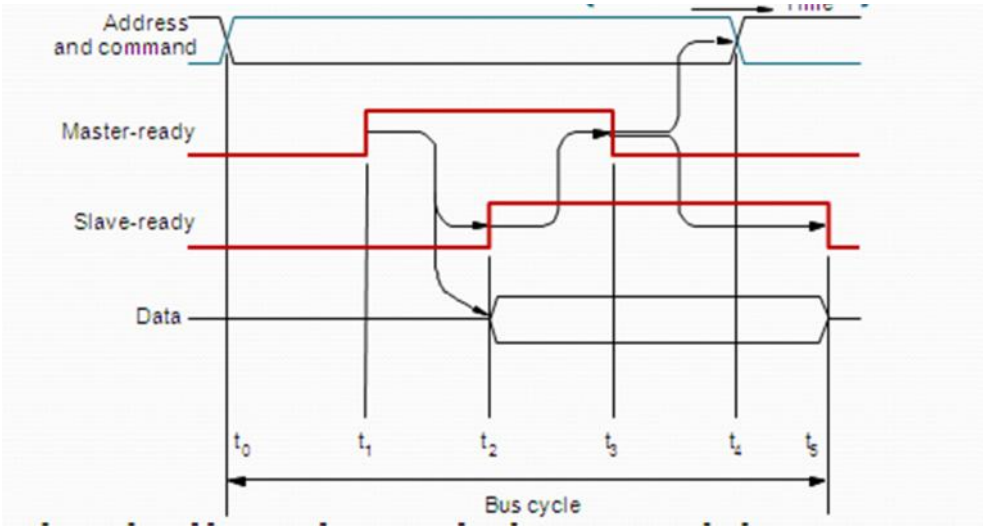
- Master–ready
- Slave ready.

- Master-ready signal is asserted by the master to indicate to the slave that it is ready to participate in a data transfer.
- Slave-ready signal is asserted by the slave in response to the master-ready from the master, and it indicates to the master that the slave is ready to participate in a data transfer.

Data transfer using the handshake protocol:

- Master places the address and command information on the bus.
- Asserts the Master-ready signal to indicate to the slaves that the address and command information has been placed on the bus.
- All devices on the bus decode the address.
- Address slave performs the required operation, and informs the processor it has done so by asserting the Slave-ready signal.
- Master removes all the signals from the bus, once Slave-ready is asserted.
- If the operation is a Read operation, Master also strobes the data into its input buffer.

Fig:Handshake control of data transfer during an input operation



The handshake protocol proceed as follows :

- At t_0 The master places the address and command information on the bus and all devices on the bus begin to decode the information
- At t_1 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 $t_1 - t_0$ is intended to allow for any skew that may occurs on the bus.
The skew occurs when two signals simultaneously transmitted from one source arrive at the destination at different time.

Thus to guarantee that the Master ready signal does not arrive at any device a head of the address and command information the delay $t_1 - t_0$ should be larger than the maximum possible bus skew.

- At t_2 The selected slave having decoded the address and command information performs the required i/p operation by placing the data from its data register on the data lines. At the same time, it sets the "slave – Ready" signal to 1.
- At t_3 The slave ready signal arrives at the master indicating that the i/p data are available on the bus.
- At t_4 The master removes the address and command information on the bus.

The delay between t_3 and t_4 is again intended to allow for bus skew.
Errorneous addressing may take place if the address, as seen by some device on the bus, starts to change while the master – ready signal is still equal to 1.

- At t_5 When the device interface receives the 1 to 0 tranitions of the Master – ready signal. It removes the data and the slave – ready signal from the bus. This completes the i/p transfer.

In this diagram, the master place the output data on the data lines and at the same time it transmits the address and command information.

The selected slave strobes the data to its o/p buffer when it receives the Master-ready signal and it indicates this by setting the slave – ready signal to 1.
At time t_0 to t_1 and from t_3 to t_4, the Master compensates for bus.
A change of state is one signal is followed by a change is the other signal. Hence this scheme is called as Full Handshake.
It provides the higher degree of flexibility and reliability.

4.9 INTERFACE CIRCUITS:

- I/O interface consists of the circuitry required to connect an I/O device to a computer bus.
- Side of the interface which connects to the computer has bus signals for:
  - Address,
  - Data
  - Control
- Side of the interface which connects to the I/O device has:
  - Datapath and associated controls to transfer data between the interface and the I/O device.
  - This side is called as a "port".
- Ports can be classified into two:
  - Parallel port,
  - Serial port.

4.9.1 Parallel Port:

- Parallel port transfers data in the form of a number of bits, normally 8 or 16 to or from the device.
- Serial port transfers and receives data one bit at a time.
- Processor communicates with the bus in the same way, whether it is a parallel port or a serial port.
  - Conversion from the parallel to serial and vice versa takes place inside the interface circuit.

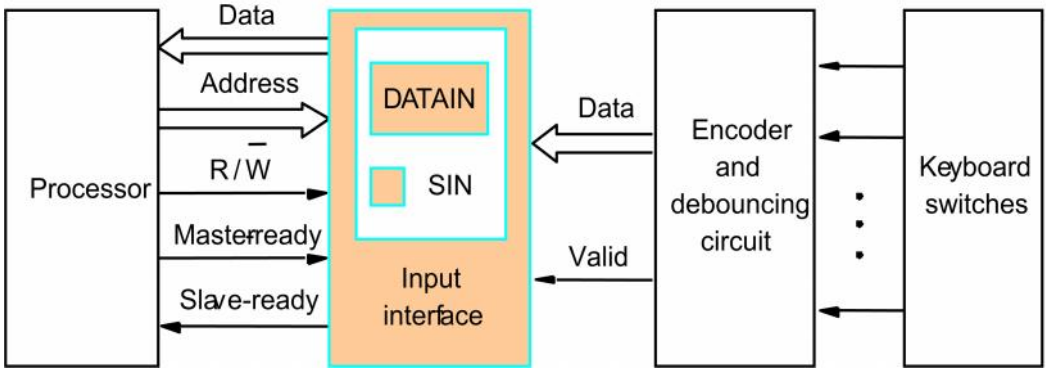


Fig:Keyboard is connected to a processor using a parallel port.

Processor is 32-bits and uses memory-mapped I/O and the asynchronous bus protocol.
On the processor side of the interface we have:

- Data lines.
- Address lines
- Control or R/W line.
- Master-ready signal and
- Slave-ready signal.

The output of the encoder consists of the bits that represent the encoded character and one signal called **vali**d,whi**c**h indicates the key is pressed.

The information is sent to the interface circuits,which contains a data register,DATAIN and a status flag SIN.

When a key is pressed, the Valid signal changes from 0 to1,causing the ASCII code to be loaded into DATAIN and SIN set to 1.

The status flag SIN set to 0 when the processor reads the contents of the DATAIN register.

The interface circuit is connected to the asynchronous bus on which transfers are controlled using the Handshake signals Master ready and Slave-ready.

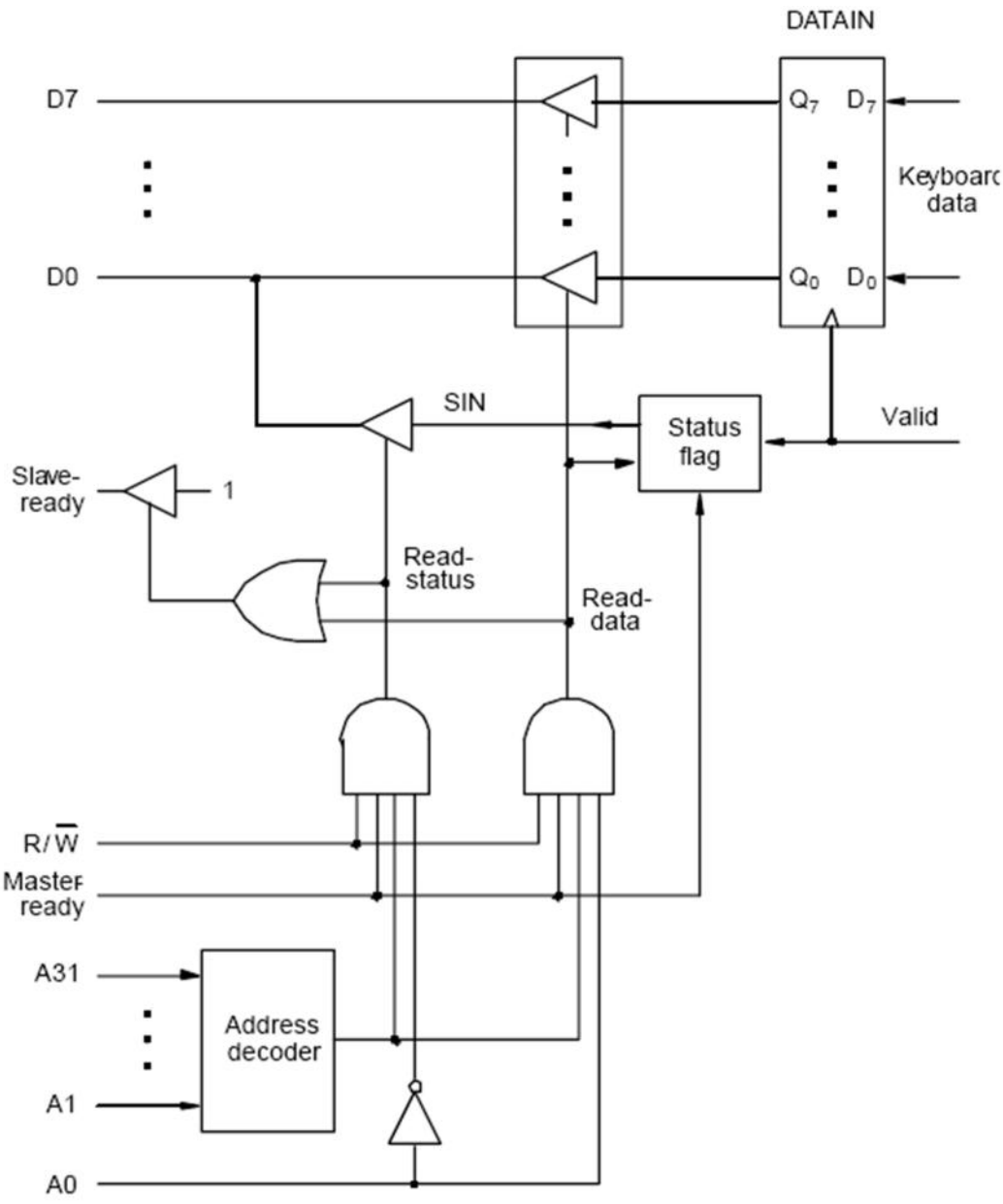


Fig: Input Interface Circuit

Output lines of DATAIN are connected to the data lines of the bus by means of 3 state drivers are turned on when the processor issues a read signal and the address selects this register

SIN signal is generated using a status flag circuit. It is connected to line D_0 of the processor bus using a three-state driver. Address decoder selects the input interface basedon bits A_1 through A_{31}. Bit A0 determines whether the status or data register is to be read, when Master-ready is active. In response, the processor activates the Slave-readysignal, when either the Read-status or Read-datais equal to 1, which depends on line A_0.

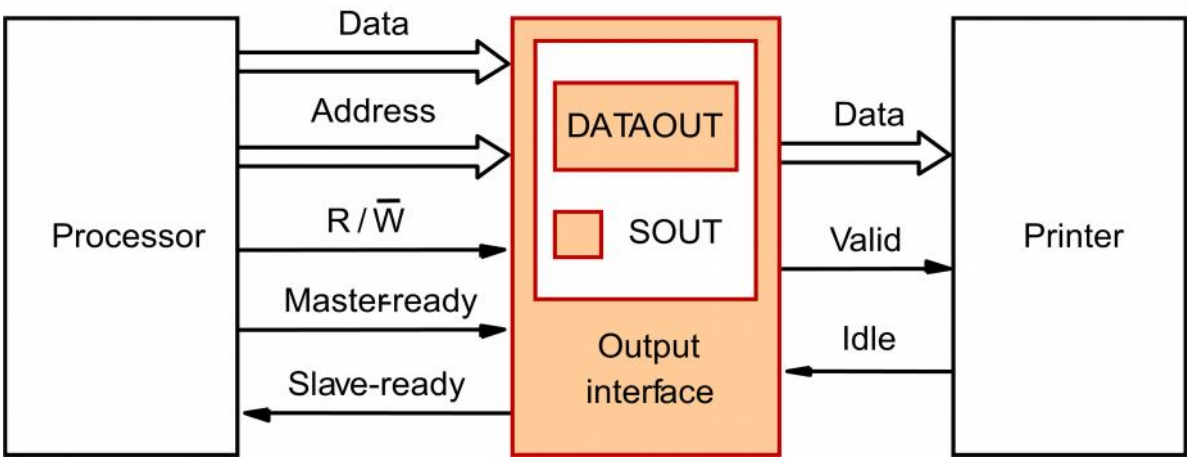


Fig:Printer is connected to a processor using a parallel port.

Processor is 32 bits, uses memory-mapped I/O and asynchronous bus protocol.
On the processor side:

- Data lines.
- Address lines
- Control or R/W line.
- Master-ready signal and
- Slave-ready signal.

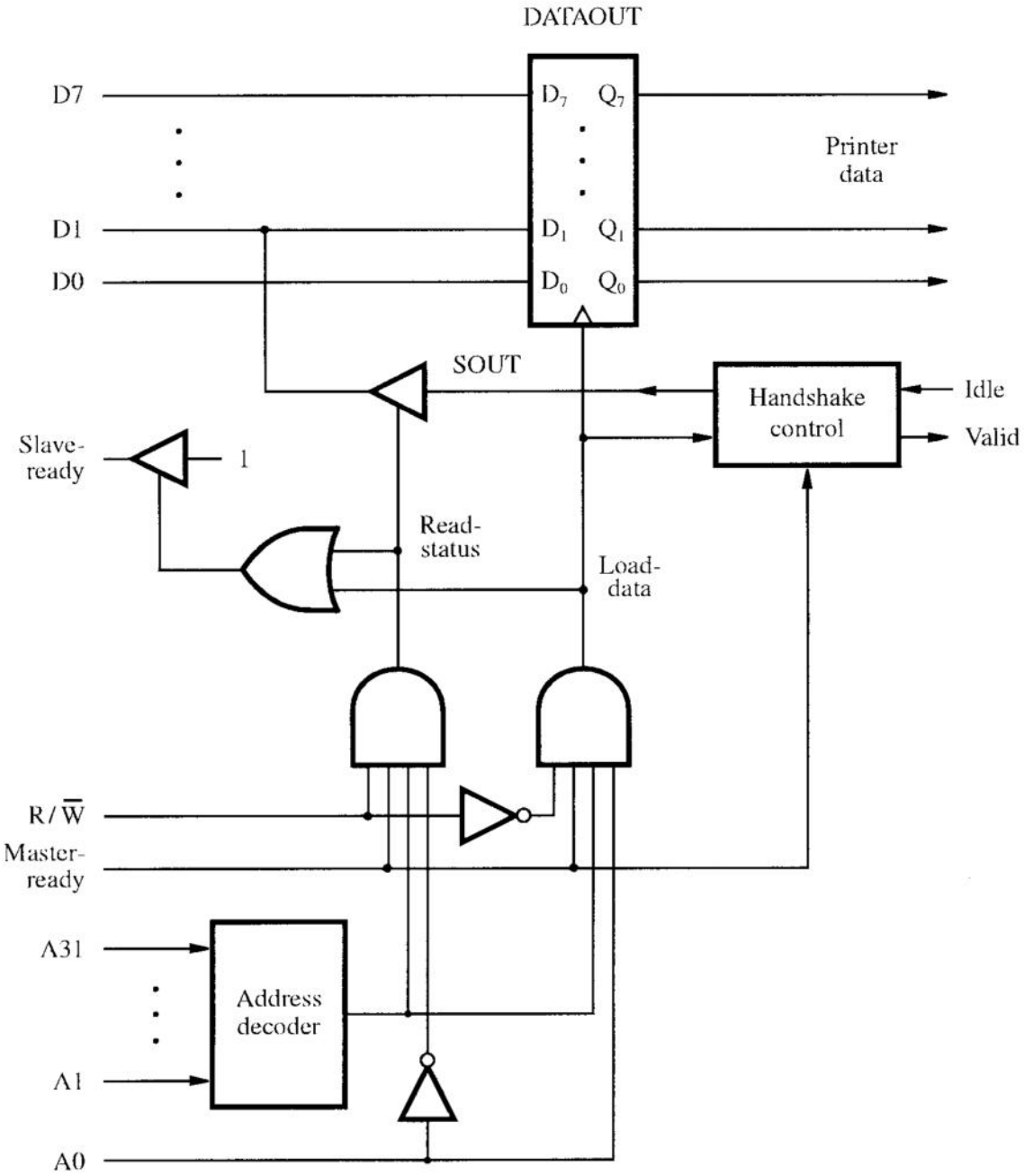


Fig: Output Interface Circuit

Data lines of the processor bus are connected to the DATAOUT register of the interface. The status flag SOUT is connected to the data line D1 using a three-state driver. The three-state driver is turned on, when the control Read-status line is1. Address decoder selects the

output interface using address lines A1 through A31.Address line A0 determines whether the data is to be loaded into the DATAOUT register or status flag isto be read. If the Load-data line is 1, then the

Valid line is set to 1.If the Idle line is 1, then the statusflag SOUT is set to 1.

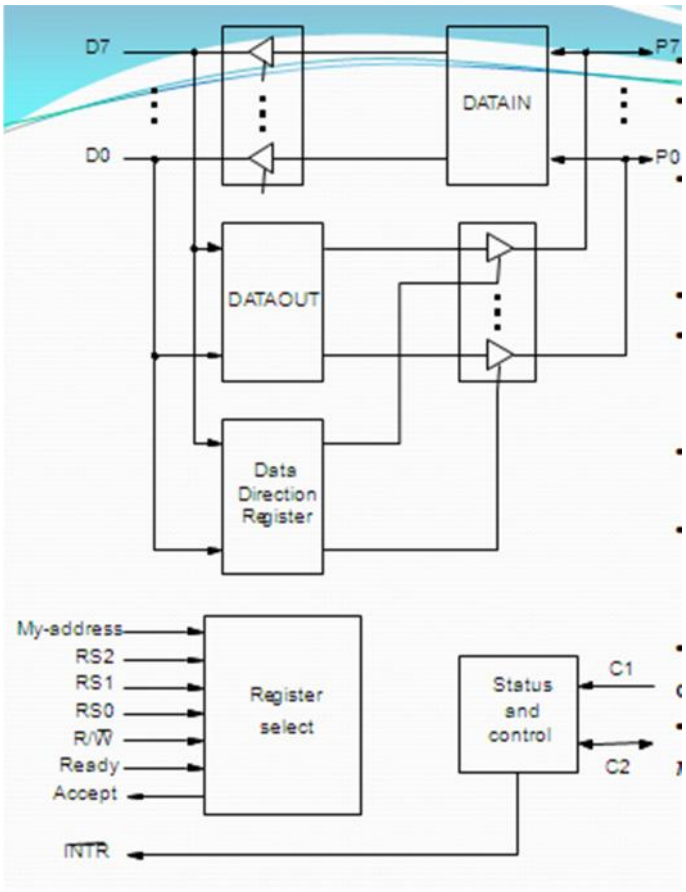


Fig :General 8 bit- parallel interface

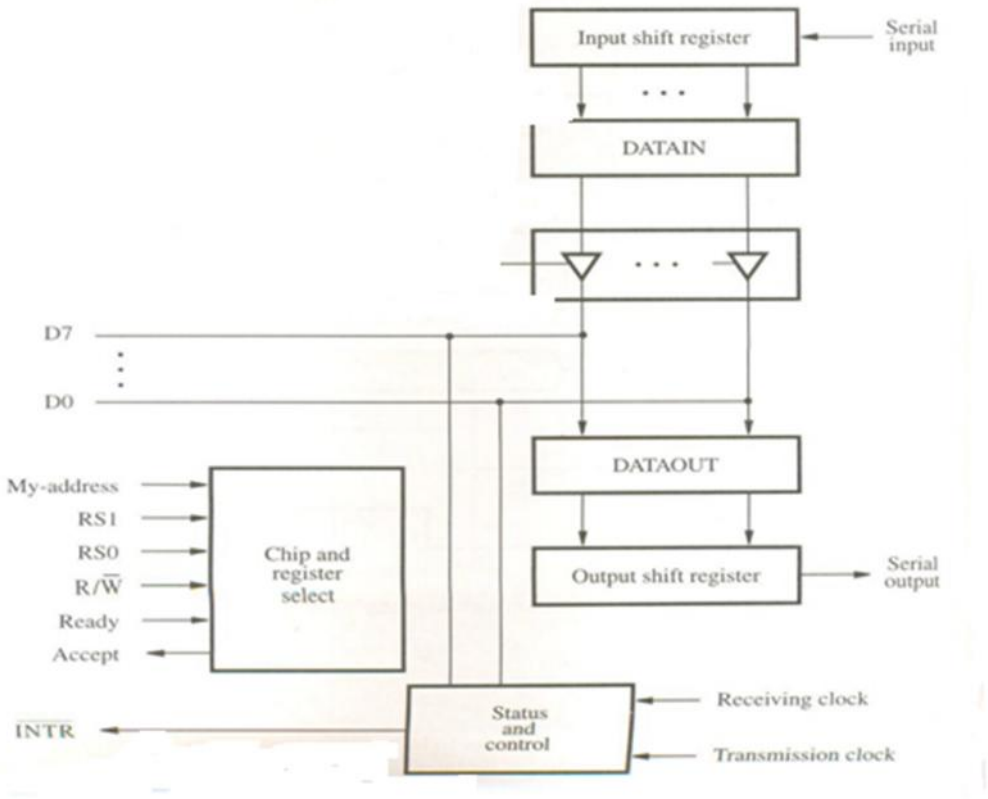
Data lines to I/O device are bidirectional. Data lines P7 through P0 can be used for both input, and output. In fact, some lines can be used for input & some for output depending on the pattern in the Data Direction Register (DDR). Processor places an 8-bit pattern into a DDR.If a given bit position in the DDR is 1, the corresponding data line acts as an output line, otherwise it acts as an input line. C1 and C2 control the interaction between the interface circuit and the I/O devices. Ready and Accept lines are the handshake control lines on the processor bus side, and are connected to Master-ready & Slave-ready.Input signal My-address is connected to the output of an address decoder. Three register select lines that allow up to 8registers to be selected.

4.9.2 Serial Port:

Serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.

Serial port communicates in a bit-serial fashion on the device side and bit parallel fashion on the bus side.

Transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.



Input shift register accepts input one bit at a time from the I/O device. Once all the 8 bits are received, the contents of the input shift register are loaded in parallel into DATAIN register.Output data in the DATAOUT register are loaded into the output shift register. Bits are shifted out of the output shift register and sent out to the I/O device onebit at a time. As soon as data from the input shift reg.are loaded into DATAIN, it can start accepting another 8 bits of data. Input shift register and DATAIN registersare both used at input so that the input shift register can start receiving anotherset of 8 bits from the input device after loading the contents to DATAIN, before the processor reads the contents of DATAIN. This is called as double-buffering.

Serial interfaces require fewer wires, and hence serial transmission is convenient for connecting devices that are physically distant from the computer.

Speed of transmission of the data over a serial interface is known as the “bit rate”.Bit rate depends on the nature of the devices connected.In order to accommodate devices with a range of speeds, a serial interface must be able to use a range of clock speeds.

Several standard serial interfaces have been developed:

Universal Asynchronous Receiver Transmitter (UART) for low-speed serial devices.

RS-232-C for connection to communication links.

4.10. Standard I/O INTERFACES:

The processor bus is the bus defined by the signals on the processor chip itself. Devices that require a very high speed connection to the processor, such as the main memory, may be connected directly to this bus. For electrical reasons, only a few devices can be connected in this manner. The motherboard usually provides another bus that can support more devices. The two buses are interconnected by a circuit, which we will call a bridge that translates the signals and protocols of one bus into those of the other. Devices connected to the expansion bus appear to the processor as if they were connected directly to the processor's own bus. The only difference is that the bridge circuit introduces a small delay in data transfers between the processor and those devices.

It is not possible to define a uniform standard for the processor bus. The structure of this bus is closely tied to the architecture of the processor. It is also dependent on the electrical characteristics of the processor chip, such as its clock speed. The expansion bus is not subject to these limitations, and therefore it can use a standardized signaling scheme. A number of standards have been developed. Some have evolved by default, when a particular design became commercially successful. For example, IBM developed a standard called ISA (Industry Standard Architecture) for their personal computer known at the time as PC AT. The popularity of that computer led to other manufacturers producing ISA -compatible interfaces for their 110 devices, thus making ISA into a de facto standard.

Some standards have been developed through industrial cooperative efforts, even among competing companies driven by their common self-interest in having compatible

products. In some cases, organizations such as the IEEE (Institute of Electrical and Electronics Engineers), ANSI (American Nationa1 Standards Institute), or international bodies such as ISO (International Standards Organization) have blessed these standards and given them an official status.

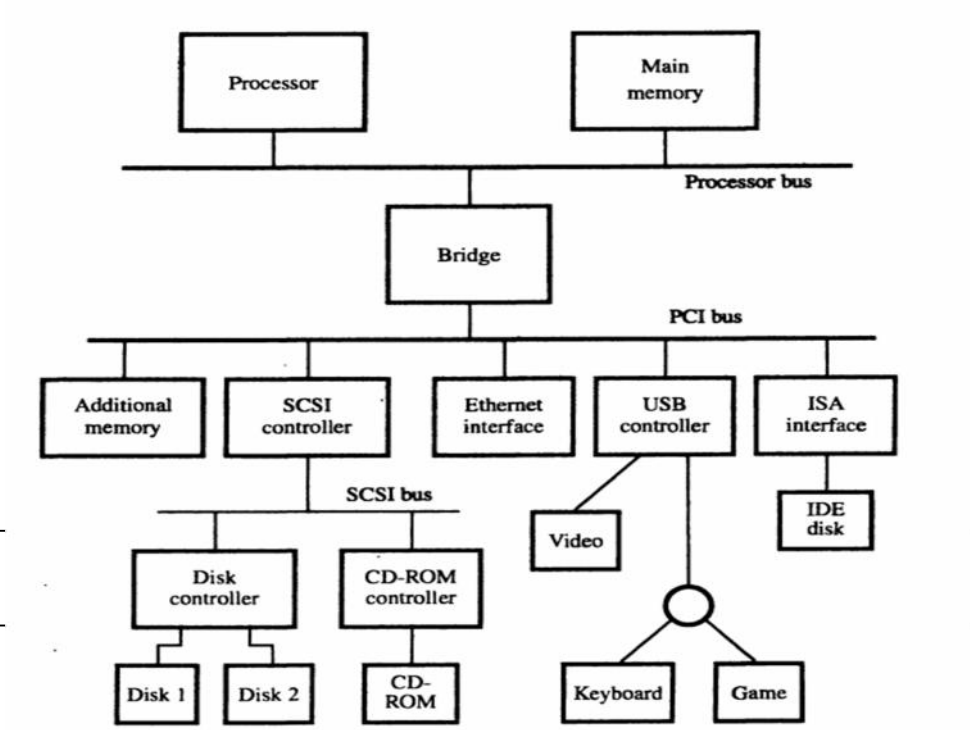


Figure 12.3: An example of a computer system using different interface standards

In this section, we present 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 illustrated in Figure 12.3. The PCI standard defines an expansion bus on the motherboard, SCSI and USB are used for connecting additional devices, both inside and outside the computer box. The SCSI bus is a high-speed parallel bus intended for devices such as disks and video displays. The USB bus uses serial transmission to suit the needs of equipment ranging from keyboards to game controls to internet connections. The figure shows an interface circuit that enables devices compatible with the earlier ISA standard, such as the popular IDE (Integrated Device Electronics) disk, to be connected. It also shows a connection to an Ethernet. The Ethernet is a widely used local area network, providing a high-speed

connection among computers in a building or a university campus.

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

4.10.1 PCI BUS

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 80x86 processors. Later, the 16-bit bus used on the PC AT computers became known as the ISA bus. Its extended 32-bit version is known as the ISA bus. Other buses developed in the eighties with similar capabilities are the Micro channel used in IBM PCs and the NuBus used in Macintosh computers.

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 connecting I/O devices. To connect a new device, the user simply connects the device interface board to the bus.

Data Transfer

In today's computers, most memory transfers involve a burst of data rather than just one word. The reason is that modern processors include a cache memory. 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 primarily to

support this mode of operation. A read or a write operation involving a single word is simply treated as a burst of length one.

The bus supports 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, such as Pentium, that have a separate I/O address space. However, the system designer may choose to use memory-mapped I/O even when a separate I/O address space is available. In fact, this is the approach recommended by the PCI standard for wider compatibility. The configuration space is intended to give the PCI its plug-and- play capability. A 4-bit command that accompanies the address identifies which of the three spaces is being used in a given data transfer operation.

Figure 12.3 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 Figure 12.4.

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.

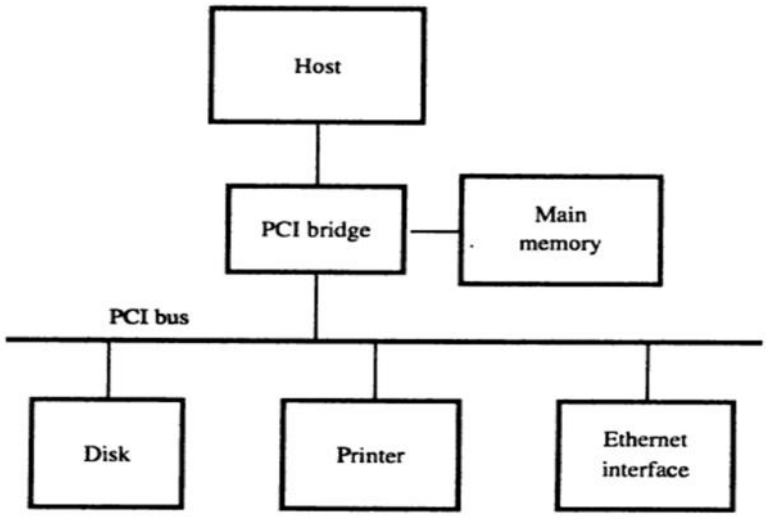


Figure 12.4: Use of a PCI bus in a computer system

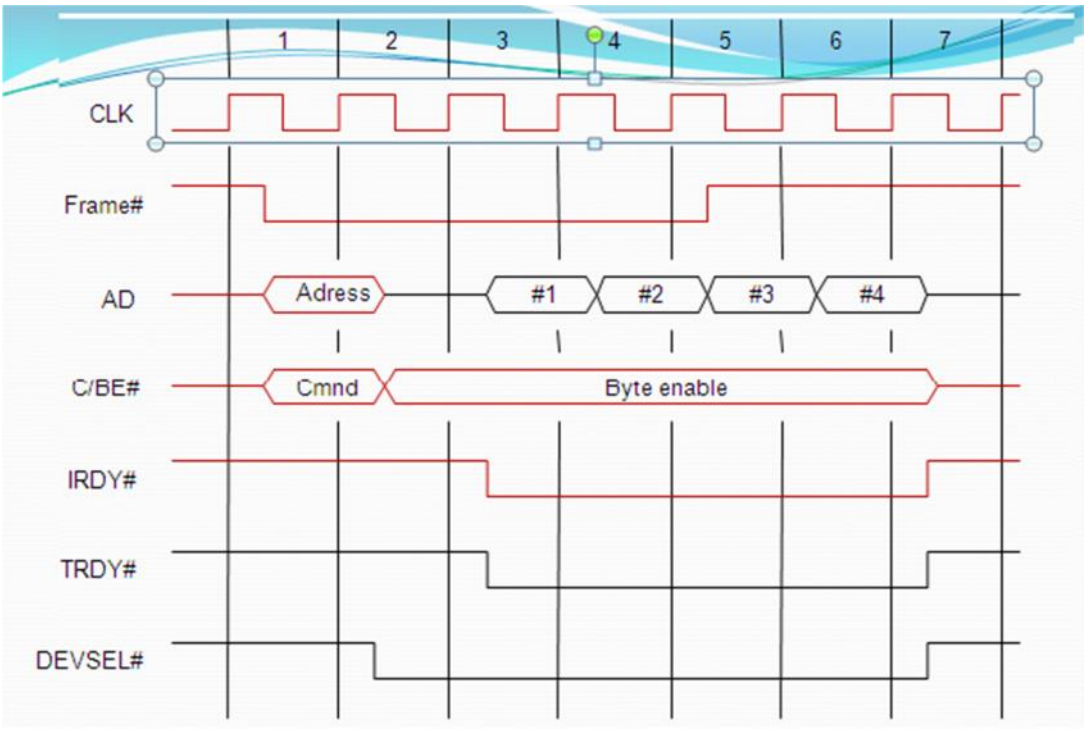
Data Transfer Signals on PCI Bus:

Name	**Function**
CLK	33 MHZ / 66 MHZ clock

FRAME #	Sent by the indicator to indicate the duration of
AD	32 address / data line
C/BE #	4 command / byte Enable Lines
IRDY, TRDYA	Initiator Ready, Target Ready Signals
DEVSEL #	A response from the device indicating that it has recognized its address and is ready for data transfer transaction.
IDSEL #	Initialization Device Select

Individual word transfers are called „**phases'**.

Fig :Read operation an PCI Bus



In Clock cycle1, the processor asserts FRAME # to indicate the beginning of a transaction ; it sends the address on AD lines and command on C/BE # Lines.

Clock cycle2 is used to turn the AD Bus lines around ; the processor ; The processor removes the address and disconnects its drives from AD lines.

The selected target enable its drivers on AD lines and fetches the requested data to be placed on the bus.

It asserts DEVSEL # and maintains it in asserted state until the end of the transaction.

C/BE # is used to send a bus command in clock cycle and it is used for different purpose during the rest of the transaction.

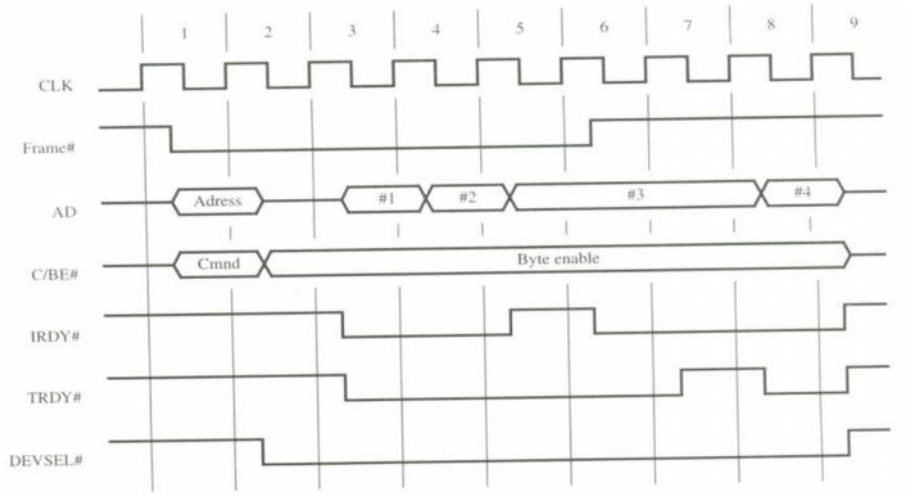
During clock cycle 3, the initiator asserts IRDY #, to indicate that it is ready to receive data.

If the target has data ready to send then it asserts TRDY #. In our eg, the target sends 3 more words of data in clock cycle 4 to 6.

The indicator uses FRAME # to indicate the duration of the burst, since it read 4 words, the initiator negates FRAME # during clock cycle 5.

After sending the 4^{th} word, the target disconnects its drivers and negates DEVSEL # during clockcycle 7.

Fig: A read operation showing the role of IRDY# / TRY#



It indicates the pause in the middle of the transaction.

The first and words are transferred and the target sends the 3^{rd} word in cycle 5. But the indicator is not able to receive it. Hence it negates IRDY#.

In response the target maintains 3^{rd} data on AD line until IRDY is asserted again. In cycle 6, the indicator asserts IRDY. But the target is not ready to transfer the fourth word immediately, hence it negates TRDY in cycle 7. Hence it sends the 4^{th} word and asserts TRDY# at cycle 8.

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 simplifies this process by incorporating in each I/O device interface a small configuration ROM memory that stores information about that device. The configuration ROMs of all devices are accessible in the configuration address space. 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 characteristics.

The PCI has a configuration ROM memory that stores information about that device.

The configuration ROM‟s of all devices are accessible in the configuration address space.

The initialization s/w read these ROM‟s whenever the S/M is powered up or reset In each case, it determines whether the device is a printer, keyboard, Ethernet interface or disk controller.

Devices are assigned address during initialization process and each device has an w/p signal called IDSEL # (Initialization device select) which has 21 address lines (AD) (AD to AD_{31}).

During configuration operation, the address is applied to AD i/p of the device and the corresponding AD line is set to and all other lines are set to 0.

AD11 - AD31 **Upper address line**

A00 - A10 **Lower address line** → Specify the type of the operation and to access the content of device configuration ROM.

The configuration software scans all 21 locations.

PCI bus has interrupt request lines.

Each device may requests an address in the I/O space or memory space

Electrical Characteristics:

The connectors can be plugged only in compatible motherboards PCI bus can operate with either 5 – 33V power supply.

The motherboard can operate with signaling system.

4.10.2 SCSI Bus:- (Small Computer System Interface)

SCSI refers to the standard bus which is defined by ANSI (American National Standard Institute).

SCSI bus the several options. It may be,

Narrow bus	It has 8 data lines & transfers 1 byte at a time.
Wide bus	It has 16 data lines & transfer 2 byte at a time.
Single-Ended Transmission	Each signal uses separate wire.
HVD (High Voltage Differential)	It was 5v (TTL cells)
LVD (Low Voltage Differential)	It uses 3.3v

Because of these various options, SCSI connector may have 50, 68 or 80 pins. The data transfer rate ranges from 5MB/s to 160MB/s 320Mb/s, 640MB/s.

The transfer rate depends on,

Length of the cable

Number of devices connected.

To achieve high transfer rat, the bus length should be 1.6m for SE signaling and 12m for LVD signaling.

The SCSI bus us connected to the processor bus through the SCSI controller. The data are stored on a disk in blocks called sectors.

Each sector contains several hundreds of bytes. These data will not be stored in contiguous memory location.

SCSI protocol is designed to retrieve the data in the first sector or any other selected sectors.

Using SCSI protocol, the burst of data are transferred at high speed. The controller connected to SCSI bus is of 2 types. They are,

- Initiator
- Target

Initiator:

It has the ability to select a particular target & to send commands specifying the operation to be performed.

They are the controllers on the processor side.

Target:

The disk controller operates as a target.

It carries out the commands it receive from the initiator. The initiator establishes a logical connection with the intended target.

Steps:

Consider the disk read operation, it has the following sequence of events.

The SCSI controller acting as initiator, contends process, it selects the target controller & hands over control of the bus to it.

The target starts an output operation, in response to this the initiator sends a command specifying the required read operation.

The target that it needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspends the connection between them. Then it releases the bus.

The target controller sends a command to disk drive to move the read head to the first sector involved in the requested read in a data buffer. When it is ready to begin transferring data to initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.

The target transfers the controls of the data buffer to the initiator & then suspends the connection again. Data are transferred either 8 (or) 16 bits in parallel depending on the width of the bus.

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

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

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

Bus Signals:-

The bus has no address lines.

Instead, it has data lines to identify the bus controllers involved in the selection / reselection / arbitration process.

For narrow bus, there are 8 possible controllers numbered from 0 to 7. For a wide bus, there are 16 controllers.

Once a connection is established b/w two controllers, these is no further need for addressing & the datalines are used to carry the data.

SCSI bus signals:

Category	**Name**	**Function**
Data	- DB (0) to DB (7) - DB(P)	Datalines Parity bit for data bus.
Phases	- BSY - SEL	Busy Selection
Information type	- C/D - MSG	Control / Data Message
Handshake	- REQ - ACK	Request Acknowledge
Direction of transfer	I/O	Input / Output
Other	- ATN - RST	Attention Reset.

All signal names are proceeded by minus sign.
This indicates that the signals are active or that the dataline is equal to 1, when they are in the low voltage state.

Phases in SCSI Bus:-

The phases in SCSI bus operation are,

- Arbitration
- Selection
- Information transfer
- Reselection

Arbitration:-

When the –BSY signal is in inactive state, the bus will he free & any controller can request the use of the bus.

Since each controller may generate requests at the same time, SCSI uses distributed arbitration scheme.

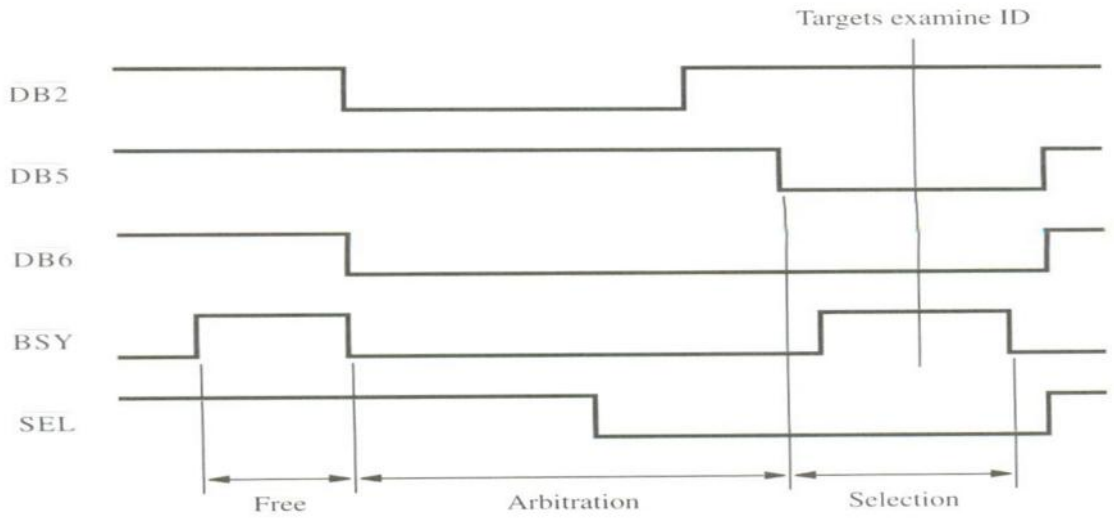
Each controller on the bus is assigned a fixed priority with controller 7 having the highest priority.

When –BSY becomes active, all controllers that are requesting the bus examines the data lines & determine whether the highest priority device is requesting the bus at the same time.

The controller using the highest numbered line realizes that it has won the arbitration process.

At that time, all other controllers disconnect from the bus & wait for –BSY to become inactive again.

Fig:Arbitration and selection on the SCSI bus.Device 6 wins arbitration and select device 2



Selection:

Here Device wons arbitration and it asserts –BSY and –DB6 signals. The Select Target Controller responds by asserting –BSY.
This informs that the connection that it requested is established.

Reselection:

The connection between the two controllers has been reestablished,with the target in control the bus as required for data transfer to proceed.

4.10.3 USB –Universal Serial Bus

USB supports 3 speed of operation. They are,

- Low speed (1.5Mb/s)
- Full speed (12mb/s)
- High speed (480mb/s)

The USB has been designed to meet the key objectives. They are,

- It provide a simple, low cost & easy to use interconnection s/m that overcomes the difficulties due to the limited number of I/O ports available on a computer.
- It accommodate a wide range of data transfer characteristics for I/O devices including telephone & Internet connections.
- Enhance user convenience through **'Plug & Play'** mode of operation.

Port Limitation:-
Normally the system has a few limited ports.
To add new ports, the user must open the computer box to gain access to the internal expansion bus & install a new interface card.

The user may also need to know to configure the device & the s/w.

Merits of USB:-

USB helps to add many devices to a computer system at any time without opening the computer box.

Device Characteristics:-

The kinds of devices that may be connected to a cptr cover a wide range of functionality.

The speed, volume & timing constrains associated with data transfer to & from devices varies significantly.

Eg:1 Keyboard Since the event of pressing a key is not synchronized to any other event in a computer system, the data generated by keyboard are called asynchronous. The data generated from keyboard depends upon the speed of the human operator which is about 100bytes/sec.

Eg:2 Microphone attached in a cptr s/m internally / externally

The sound picked up by the microphone produces an analog electric signal, which must be converted into digital form before it can be handled by the cptr.
This is accomplished by sampling the analog signal periodically.
The sampling process yields a continuous stream of digitized samples that arrive at regular intervals, synchronized with the sampling clock. Such a stream is called isochronous (ie) successive events are separated by equal period of time.

If the sampling rate in „S" samples/sec then the maximum frequency captured by sampling process is s/2.

A standard rate for digital sound is 44.1 KHz.

Requirements for sampled Voice:-

It is important to maintain precise time (delay) in the sampling & replay process. A high degree of jitter (Variability in sampling time) is unacceptable.

Eg-3:Data transfer for Image & Video:-

The transfer of images & video require higher bandwidth.
The bandwidth is the total data transfer capacity of a communication channel. To maintain high picture quality, The image should be represented by about
160kb, & it is transmitted 30 times per second for a total bandwidth if 44MB/s.

Plug & Play:-

The main objective of USB is that it provides a plug & play capability.
The plug & play feature enhances the connection of new device at any time, while the system is operation.

The system should,

- Detect the existence of the new device automatically.
- Identify the appropriate device driver s/w.
- Establish the appropriate addresses.
- Establish the logical connection for communication.

USB Architecture:-

USB has a serial bus format which satisfies the low-cost & flexibility requirements.

Clock & data information are encoded together & transmitted as a single signal. There are no limitations on clock frequency or distance arising form data skew, & hence it is possible to provide a high data transfer bandwidth by using a high
clock frequency.

To accommodate a large no/. of devices that can be added / removed at any time, the USB has the tree structure.

USB Tree Structure:

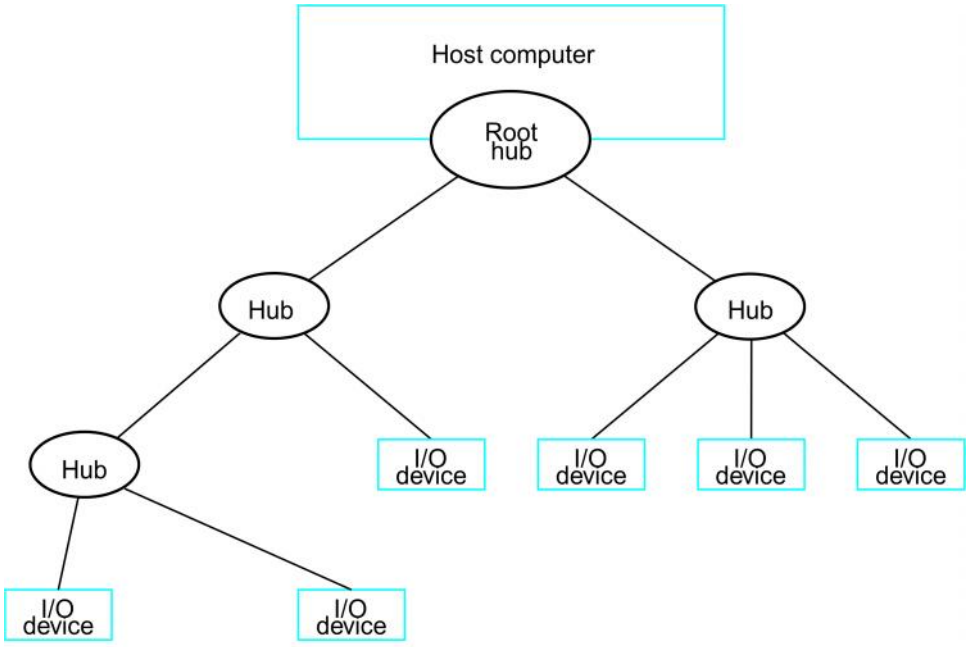


Fig:Universal Serial Bus tree structure

- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure as shown in the figure.
- 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. At the root of the tree, a root hub connects the entire tree to the host computer. The leaves of the tree are the I/O devices being served (for example, keyboard, Internet connection, speaker, or digital TV)
- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports. As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message. However, a message from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices. Hence, the USB

enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.

Addressing:

- When a USB is connected to a host computer, its root hub is attached to the processor bus, where it appears as a single device. The host software communicates with individual devices attached to the USB by sending packets of information, which the root hub forwards to the appropriate device in the USB tree.
- Each device on the USB, whether it is a hub or an I/O device, is assigned a 7-bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus.
- A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily. When a device is first connected to a hub, or when it is powered on, it has the address 0. The hardware of the hub to which this device is connected is capable of detecting that the device has been connected, and it records this fact as part of its own status information. Periodically, the host polls each hub to collect status information and learn about new devices that may have been added or disconnected.
- When the host is informed that a new device has been connected, it uses a sequence of commands to send a reset signal on the corresponding hub port, read information from the device about its capabilities, send configuration information to the device, and assign the device a unique USB address. Once this sequence is completed the device begins normal operation and responds only to the new address.

USB protocols:

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information. There are many types of packets that perform a variety of control functions.
- The information transferred on the USB can be divided into two broad categories: control and data.
    - Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error.
    - Data packets carry information that is delivered to a device.

- A packet consists of one or more fields containing different kinds of information. The first field of any packet is called the packet identifier, PID, which identifies the type of that packet.
- They are transmitted twice. The first time they are sent with their true values, and the second time with each bit complemented
- The four PID bits identify one of 16 different packet types. Some control packets, such as ACK (Acknowledge), consist only of the PID byte.

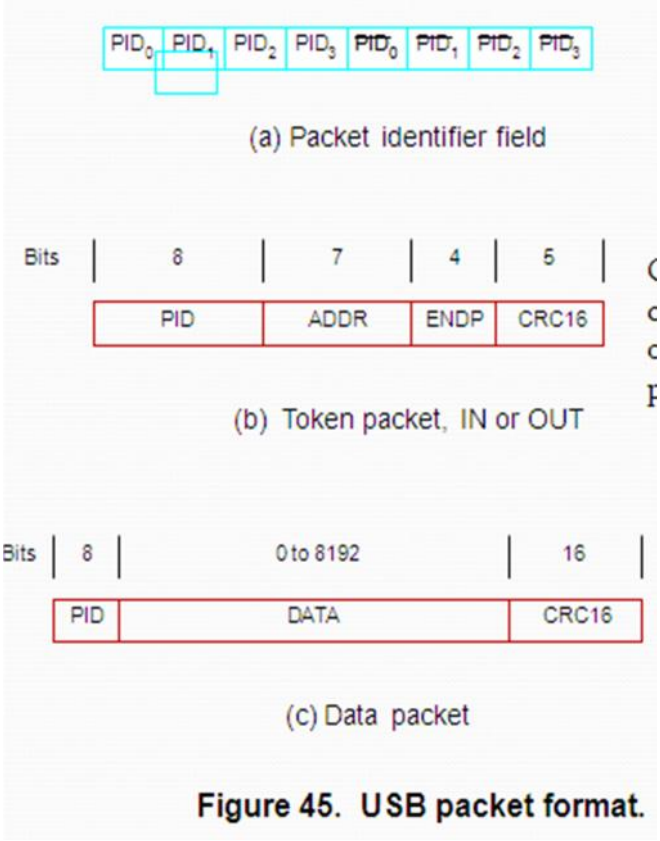


Figure 45. USB packet format.

Control packets used for controlling data transfer operations are called token packets.

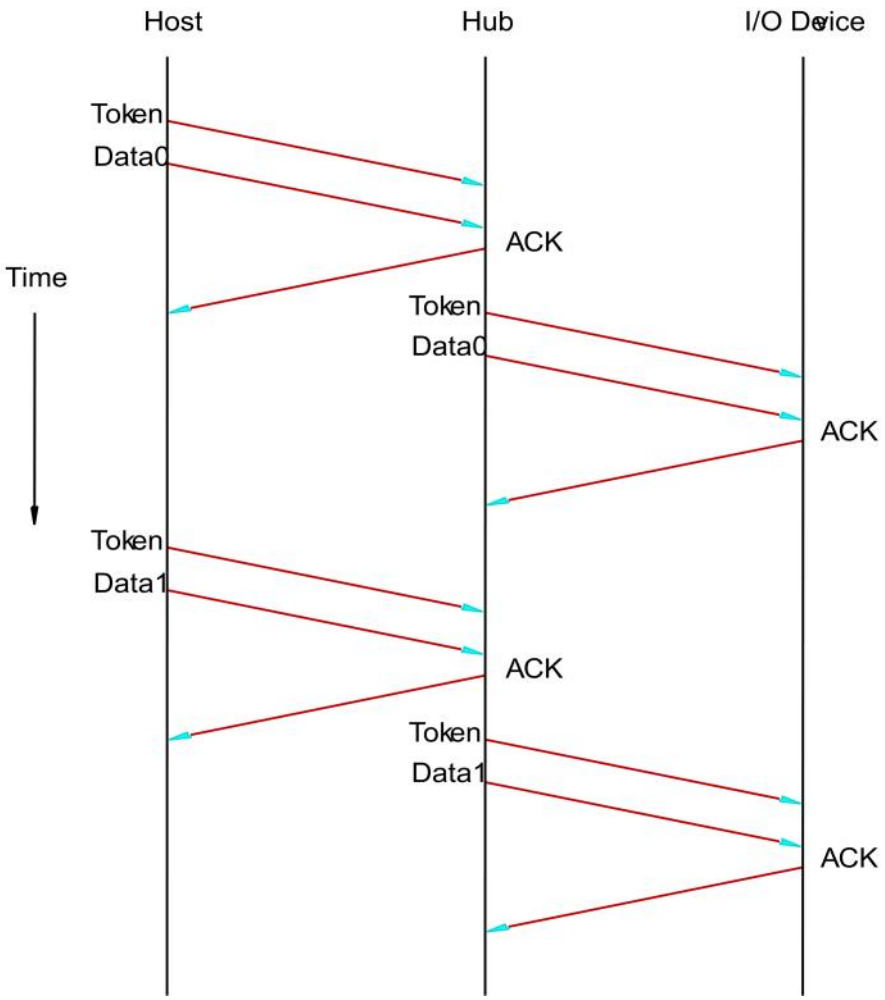


Figure: An output transfer

Isochronous Traffic on USB:

- One of the key objectives of the USB is to support the transfer of isochronous data.
- Devices that generates or receives isochronous data require a time reference to control the sampling process.
- To provide this reference. Transmission over the USB is divided into frames of equal length.
- A frame is 1ms long for low-and full-speed data.
- The root hub generates a Start of Frame control packet (SOF) precisely once every 1 ms to mark the beginning of a new frame.
- The arrival of an SOF packet at any device constitutes a regular clock signal that the device can use for its own purposes.