Input and OutputOrganization:

Accessing I/O devices, Direct Memory Access (DMA), Buses: SynchronousBusand 6 Asynchronous Bus, Interface Circuits, Standard IO Interface.

Introduction:

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Accessing I/O Devices

Accessing I/O Devices.: In computing, input/output, or I/O, refers to the communication between an information processing system (computer), and the outside world. Inputs are the signals or data received by the system, and outputs are the signals or data sent from it. I/O devices are used by a person (or other system) to communicate with a computer.

Some of the input devices are keyboard, mouse, track ball, joy stick, touch screen, digital camera, webcam, image scanner, fingerprint scanner, barcode reader, microphone and so on. Some of the output devices are speakers, headphones, monitors and printers. Devices for communication between computers, such as modems and network cards, typically serve for both input and output. I/O devices can be connected to a computer through a single bus which enables the exchange of information. The bus 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. Figure 5.1 shows the simple arrangement of I/O devices to processor and memory with single bus.

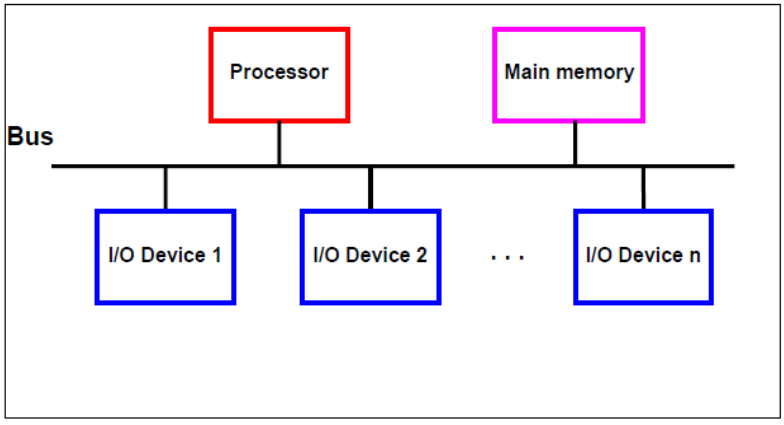


Figure 5.1 A Single bus structure

Memory-mapped I/O: The arrangement of I/O devices and the memory share the same address space 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. For example, if DATAIN is the address of the input buffer associated with the keyboard, the instruction

Move DATAIN,R0

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

Move R0,DATAOUT

sends the contents of register R0 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.

Figure 5.2 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.

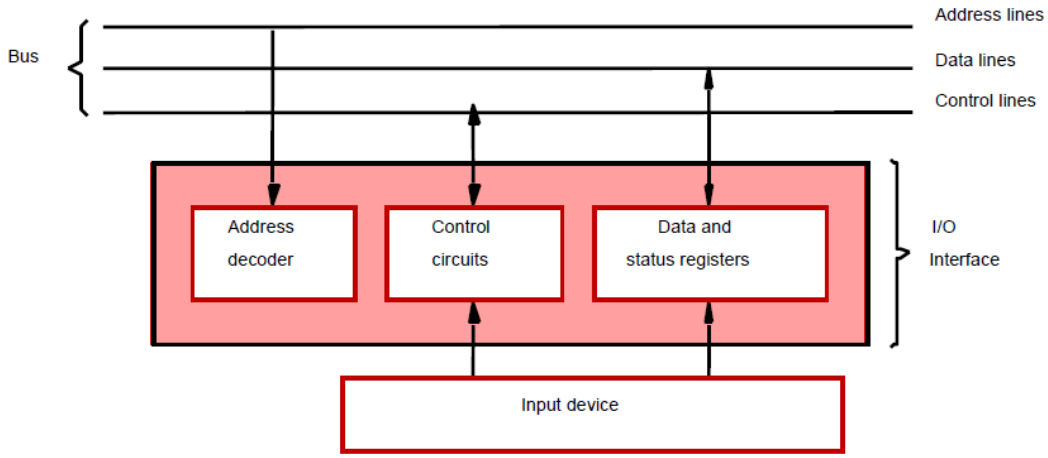


Figure 5.2 I/O interface for an input device

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. 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. This is often accomplished in a program loop that repeatedly reads the status register and checks the state of SIN. When SIN becomes equal to 1, the program reads the input data register. A similar procedure can be used to control output operations using an output status flag, SOUT

Direct Memory Access (DMA) :

DMA Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. DMA controller needs the same old circuits of an interface to communicate with the CPU and Input/Output devices.

Fig-1 below shows the block diagram of the DMA controller. The unit communicates with the CPU through data bus and control lines. Through the use of the address bus and allowing the DMA and RS register to select inputs, the register within the DMA is chosen by the CPU. RD and WR are two-way inputs. When BG (bus grant) input is 0, the CPU can communicate with DMA registers.

When BG (bus grant) input is 1, the CPU has relinquished the buses and DMA can communicate directly with the memory.

DMA controller registers :

The DMA controller has three registers as follows.

- **Address register –** It contains the address to specify the desired location in memory.
- **Word count register –** It contains the number of words to be transferred.
- **Control register –** It specifies the transfer mode.

Note –

All registers in the DMA appear to the CPU as I/O interface registers. Therefore, the CPU can both read and write into the DMA registers under program control via the data bus.

Fig 1- Block Diagram

Explanation :

The CPU initializes the DMA by sending the given information through the data bus.

- The starting address of the memory block where the data is available (to read) or where data are to be stored (to write).
- It also sends word count which is the number of words in the memory block to be read or write.
- Control to define the mode of transfer such as read or write.
- A control to begin the DMA transfer.

Interface Circuit

The **I/O interface circuit** is a mediator between the I/O device and the system to which this I/O has to be connected. In our earlier content bus structure, we had discussed a little about the I/O interface where we have seen that one end of the I/O interface is connected to the system bus and the other is connected to the input device.

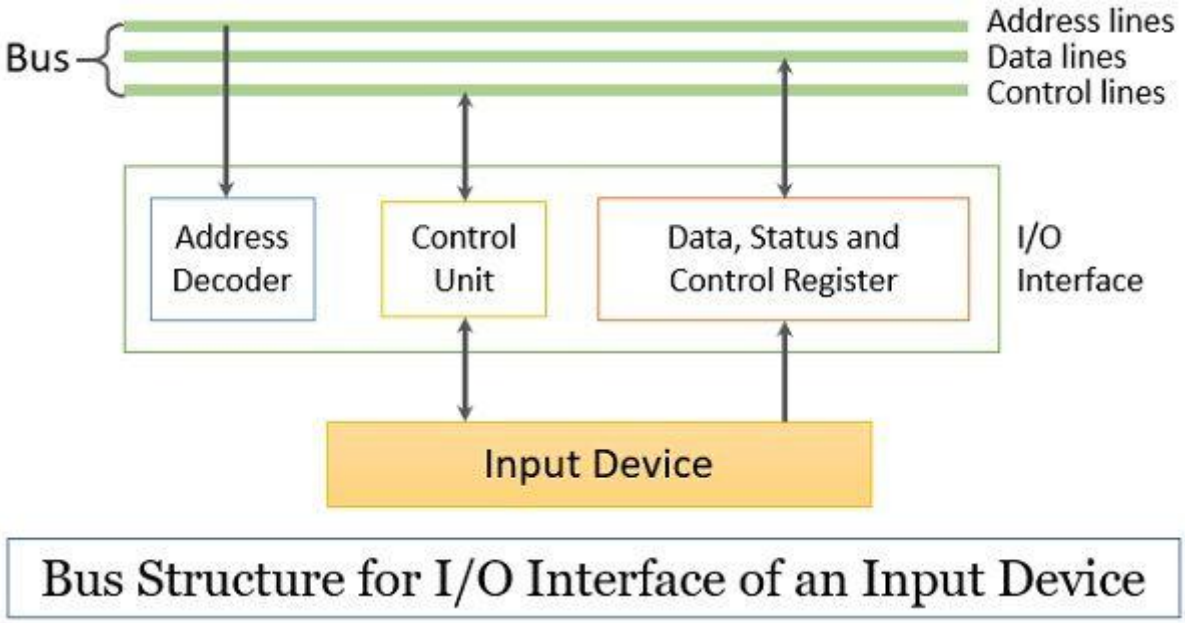
In this context, we will discuss the I/O interface circuit in more detail along with its entire functioning. We will also discuss two variants of interface circuit design i.e., parallel interface and the serial interface.

I/O Interface Circuit

The I/O interface circuit is circuitry that is designed to link the I/O devices to the processor. Now the question is why do we require an interface circuit?

We know that every component or module of the computer has its distinct capabilities and processing speed. For example, the processing speed of the CPU is much higher than the other components of the computer such as keyboard, display, etc.

So, we need a mediator to make the computer communicate with the I/O modules. This mediator is referred to as an interface circuit. Observe the figure below, we can easily see that the one end of the interface circuit is connected to the system bus line i.e., address line, data line, and control line.

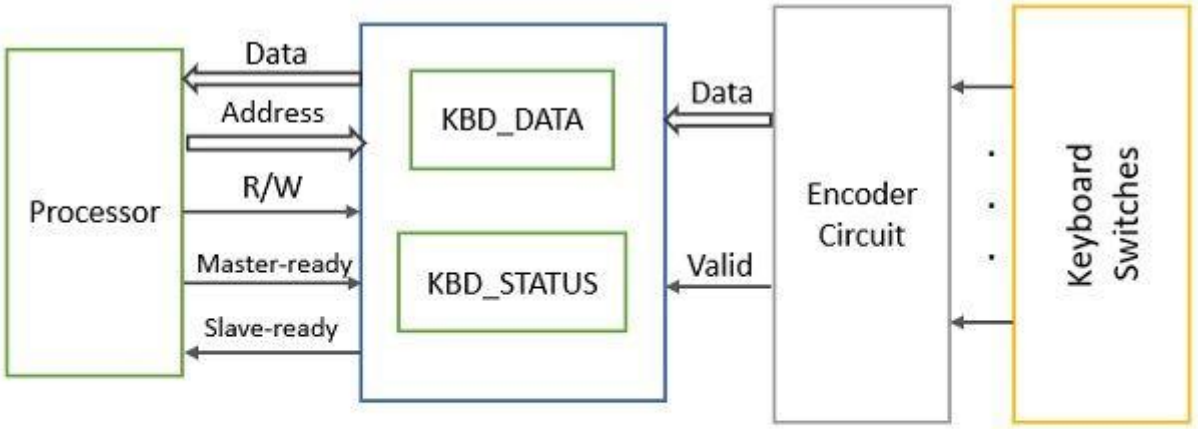


Bus Structure for I/O Interface of an Input Device

The address line is decoded by the interface circuit to determine if the processor has addressed this particular I/O device or not. The control line is decoded to identify which kind of operation is requested by the processor. The data line is used to transfer the data between I/O and the processor.
The other side of the interface circuit has the connections that are essential to transfer data between the I/O interface circuit and the I/O device. And this side of the I/O interface is referred to as the *port*. The port of the I/O interface can be a parallel port or a serial port. We will discuss these ports in the section ahead.
But before discussing these ports let us take a brief outlook of what are the features of the I/O interface circuit.

1. The interface circuit has a **data register** that stores the data temporarily while the data is being exhanged between I/O and processor.
2. The interface circuit also has a **status register**, the bits in the status register indicate the processor whether the I/O device is set for the transmission or not.
3. The interface circuit also has the **control register**, the bits in the control register indicate the type of operation (read or write) requested by the processor to the I/O interface.
4. The interface circuit also has **address decoding circuitry** which decodes the address over the address line to determine whether it is being addressed by the processor.
5. The interface circuitry also generates the **timing signals** that synchronize the operation between the processor and the I/O device.
6. The interface circuit is also responsible for the **format conversion** that is essential for exchanging data between the processor and the I/O interface.

Now, let us learn about the parallel port and the serial port of the I/O interface circuit.
Parallel Port
To understand the interface circuit with a parallel port we will take the example of two I/O devices. First, we will study an input device i.e., a keyboard that has an 8-bit input port, and then an output device i.e., a display that has an 8-bit output port. Here multiple bits are transferred at once.
Input Port
Observe the parallel input port that connects the keyboard to the processor. Now, whenever the key is tapped on the keyboard an electrical connection is established that generates an electrical signal. This signal is encoded by the encoder to convert it into ASCII code for the corresponding character pressed at the keyboard.



Parallel Input Port that connect Keyboard and Processor

The encoder then outputs one byte of data that presents the character encoded by the encoder along with one valid bit. This valid bit changes its status from 0 to 1 when the key is pressed. So, when the valid bit is 1 the ASCII code of the corresponding character is loaded to the KBD_DATA register of the input interface circuit.

Now, when the data is loaded into the KBD_DATA register the KIN status flag present in the KBD_STATUS register is set to1. Which causes the processor to read the data from KBD_DATA.

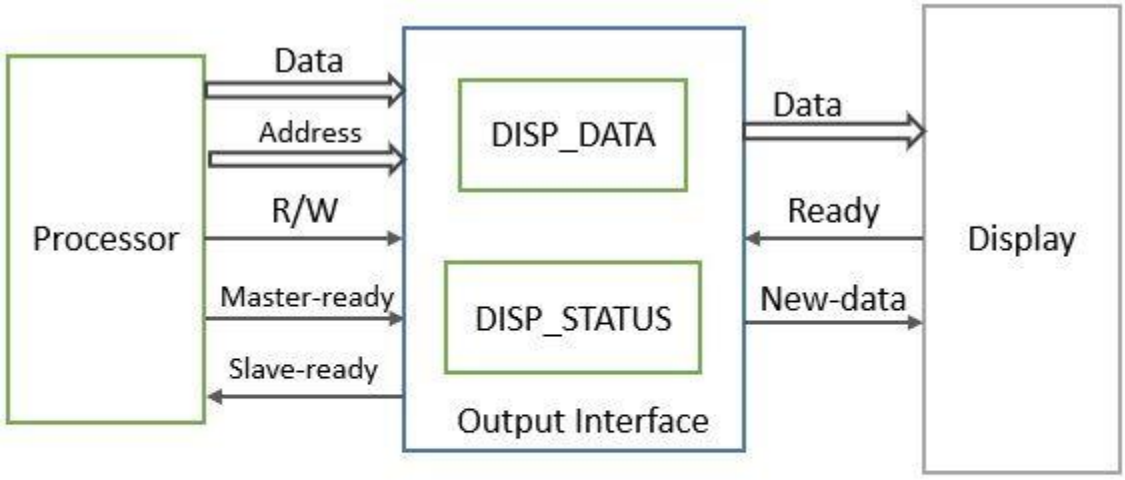
Once the processor reads the data from KBD_DATA register the KIN flag is again set to 0. Here the input interface is connected to the processor using an asynchronous bus.

So, the way they alert each other is using the master ready line and the slave ready line. Whenever the processor is ready to accept the data, it activates its master-ready line and whenever the interface is ready with the data to transmit it to the processor it activates its slave-ready line.

The bus connecting processor and interface has one more control line i.e., R/W which is set to one for reading operation.

Output Port

Observe the output interface shown in the figure below that connects the display and processor. The display device uses two handshake signals that are ready and new data and the other master and slave-ready.



Parallel Output Port that connect Display and Processor

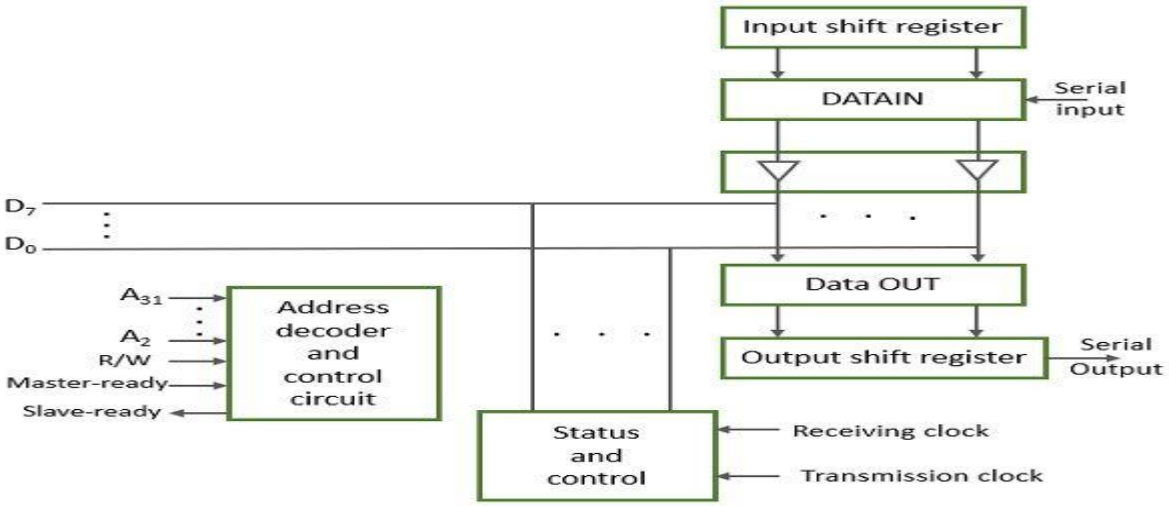
When the display unit is ready to display a character, it activates its ready line to 1 which setups the DOUT flag in the DISP_STATUS register to 1. This indicates the processor and the processor places the character to the DISP_DATA register.

As soon as the processor loads the character in the DISP_DATA the DOUT flag setbacks to 0 and the New-data line to 1. Now as the display senses that the new-data line is activated it turns the ready line to 0 and accepts the character in the DISP_DATA register to display it.

Serial Port

Opposite to the parallel port, the serial port connects the processor to devices that transmit only one bit at a time. Here on the device side, the data is transferred in the bit-serial pattern, and on the processor side, the data is transferred in the bit-parallel pattern.

The transformation of the format from serial to parallel i.e., from device to processor, and from parallel to serial i.e., from processor to device is made possible with the help of shift registers (input shift register & output shift register).



Serial Interface Circuit

Observe the figure above to understand the functioning of the serial interface at the device side. The input shift register accepts the one bit at a time in a bit-serial fashion till it receives all 8 bits. When all the 8 bits are received by the input shift register it loads its content into the DATA IN register parallelly. In a similar fashion, the content of the DATA OUT register is transferred in

parallel to the output shift register.
The serial interface port connected to the processor via system bus functions similarly to the parallel port. The status and control block has two status flags SIN and SOUT. The SIN flag is set to 1 when the I/O device inputs the data into the DATA IN register through the input shift register and the SIN flag is cleared to 0 when the processor reads the data from the DATA IN register.
When the value of the SOUT register is 1 it indicates to the processor that the DATA OUT register is available to receive new data from the processor. The processor writes the data into the DATA OUT register and sets the SOUT flag to 0 and when the output shift register reads the data from the DATA OUT register sets back SOUT to 1.
This makes the transmission convenient between the device that transmits and receives one bit at a time and the processor that transmits and receives multiple bits at a time.
The serial interface does not have any clock line to carry timing information. So, the timing information must be embedded with the transmitted data using the encoding scheme. There are two techniques to do this.

Asynchronous Serial Transmission

In the asynchronous transmission, the clock used by the transmitter and receiver is not synchronized. So, the bits to be transmitted are grouped into a group of 6 to 8 bits which has a defined starting bit and ending bit. The start bit has a logic value 0 and the stop bit has a logic value 1.
The data received at the receiver end is recognized by this start and stop bit. This approach is useful where is the transmission is slow.

Synchronous Serial Transmission

The start and stop bit we used in the asynchronous transmission provides the correct timing information but this approach is not useful where the transmission speed is high.
So, in the synchronous transmission, the receiver generates the clock that is synchronized with the clock of the transmitter. This lets the transmitting large blocks of data at a high speed.
This is all about the interface circuit that is an intermediatory circuit between the I/O device and processor. The parallel interface is faster, costly, and efficient for the devices that are at a closer distance to the processor. Whereas the serial interface is is slow, less costly, and efficient for long-distance connection.

STANDARD I/O INTERFACES

The processor bus is the bus defied 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.

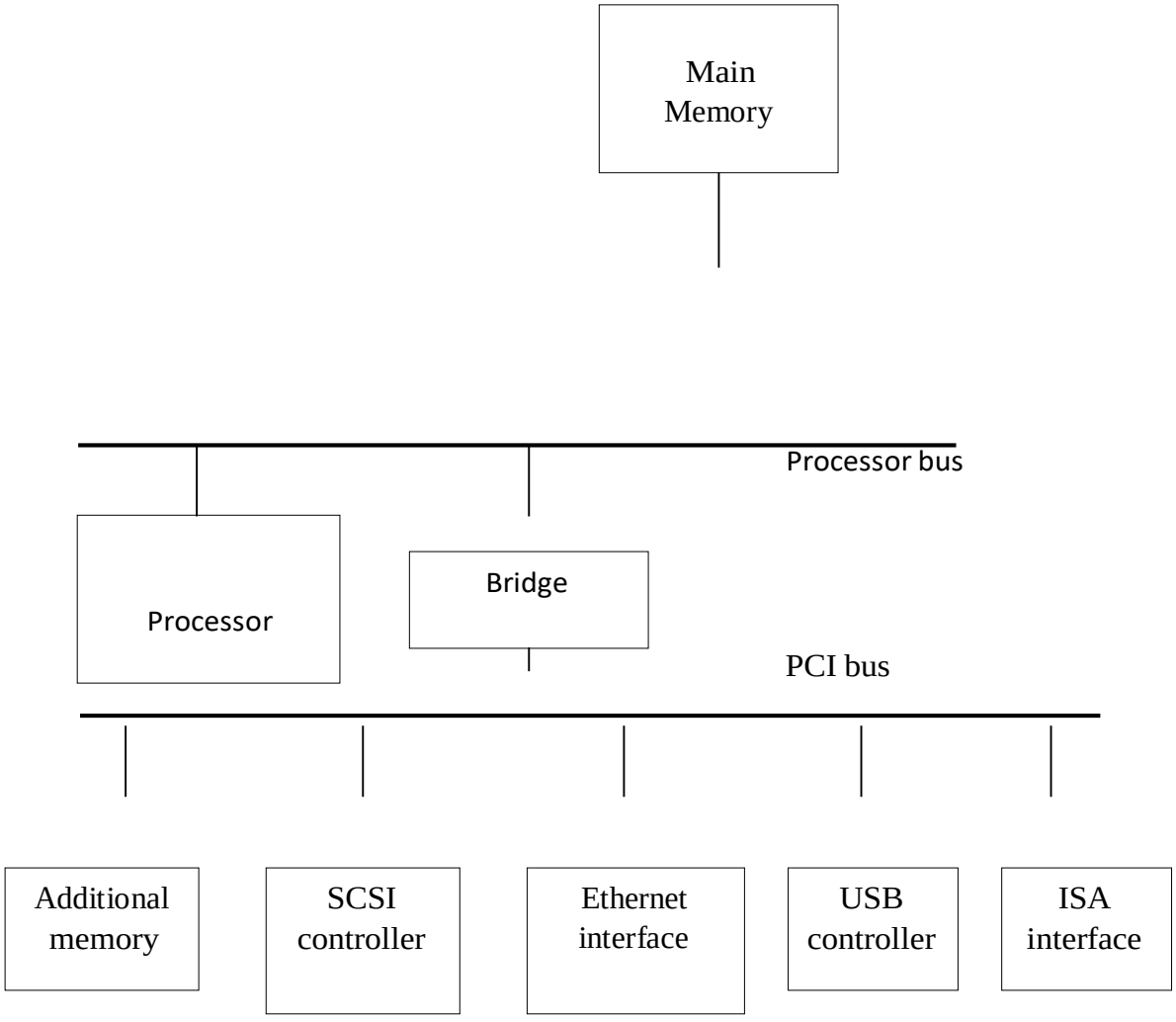
It is not possible to define a uniform standard for the processor bus. The structureof this bus is closely tied to the architecture of the processor. It is also dependent on the electrical characteristics of the processor chip, such as its clock speed. The expansion bus is not subject to these limitations, 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 bus they called ISA (Industry Standard Architecture) for their personal computer known at the time as PC AT.

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 National Standards Institute), or international bodies such as ISO (International Standards Organization) have blessed these standardsand given them an official status.

A given computer may use more than one bus standards. 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.

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



Peripheral Component Interconnect (PCI) Bus:-

The PCI bus is a good example of a system bus that grew out of the need for standardization. It supports the functions found on a processor bus bit 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
0x86 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 EISA bus. Other buses developed in the eighties with similar capabilities are the Microchannel 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. The software takes care of the rest.

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 burst 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 the address. The PCI is designed primarily 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 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, as noted , 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 its plug-and-play capability. A 4-bit command that accompanies the address identifies whichof the three spaces is being used in a given data transfer operation.

The signaling convention on the PCI bus is similar to the one used, we assumedthat the master maintains the address information on the bus until data transfer is completed. But, this is not necessary. The address is needed only long enough for the slave to be selected. The slave can store the address in its internal buffer. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for sending data in subsequent clock cycles. The result is a significant cost reduction because the number of wires on a bus is an important cost factor. This approach in used in the PCI bus.

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.

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.

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 is 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 bout various device options and characteristics.

Devices are assigned addresses during the initialization process. This means that during the bus configuration operation, devices cannot be accessed based on their address, as they have not yet been assigned one. Hence, the configuration address space uses a different mechanism. Each device has an input signal called Initialization Device Select, IDSEL#.

The PCI bus has gained great popularity in the PC word. It is also used in many other computers, such as SUNs, to benefit from the wide range of I/O devices for which a PCI interface is available. In the case of some processors, such as the Compaq Alpha, the PCI-processor bridge circuit is built on the processor chip itself, further simplifying system design and packaging.

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) under the designation X3.131 . In the original specifications 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 up to 5 megabytes/s.

The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years. SCSI-2 and SCSI-3 have been defined, and each has several options. 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.

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. The SCSI bus is connected to the processor bus through a SCSI controller. 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 status informationabout 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 with the main memory.

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. Clearly, the controller on the processor side, such as the SCSI controller, must be able to operate as an initiator. 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. While a particular connection is suspended, other device can use the bus to transfer information. 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.

The processor sends a command to the SCSI controller, which causes the following sequence of event 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 andhands 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 transferring 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 as before. At the end of this transfers, 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 as interrupt to the processor to inform it that the requested operation has been completed

This scenario show that the messages exchanged over the SCSI bus are at a higher level than those exchanged over the processor bus. In this context, a “higher level” means that the messages refer to operations that may require several steps to complete, depending on the device. Neither the processor nor the SCSI controller need be aware of the details of operation of the particular device involved in a data transfer. In the preceding example, the processor need not be involved in the disk seek operation.