CHAPTER 8:
SERIAL COMMUNICATIONS

PC-MOS uses serial ports for communication with workstations, printers, modems, networks, mice, scanners and a variety of other peripherals. Certain considerations apply to ports which are placed under interrupt driven control so the information in this chapter must be studied in conjunction with that in Chapter 9.

Serial Communications Hardware

Most systems have at least one built-in serial port, typically referred to as COM1. Some systems also have a second built-in serial port -- COM2. These built-in ports are managed directly by the host CPU chip and are thus referred to as "dumb" ports.

Adapter cards may be installed which contain 4 or 8 additional serial ports. These cards come in two varieties -- dumb and smart. A dumb multiport card provides additional ports to which modems, printers and workstations can be connected but it also requires additional processing power of the host CPU. Figure 8-1 shows the basic layout of a 4-port dumb card. A serial port is basically a UART chip (Universal Asynchronous Receiver/Transmitter) along with some associated circuitry to buffer signals to and from the outside world.

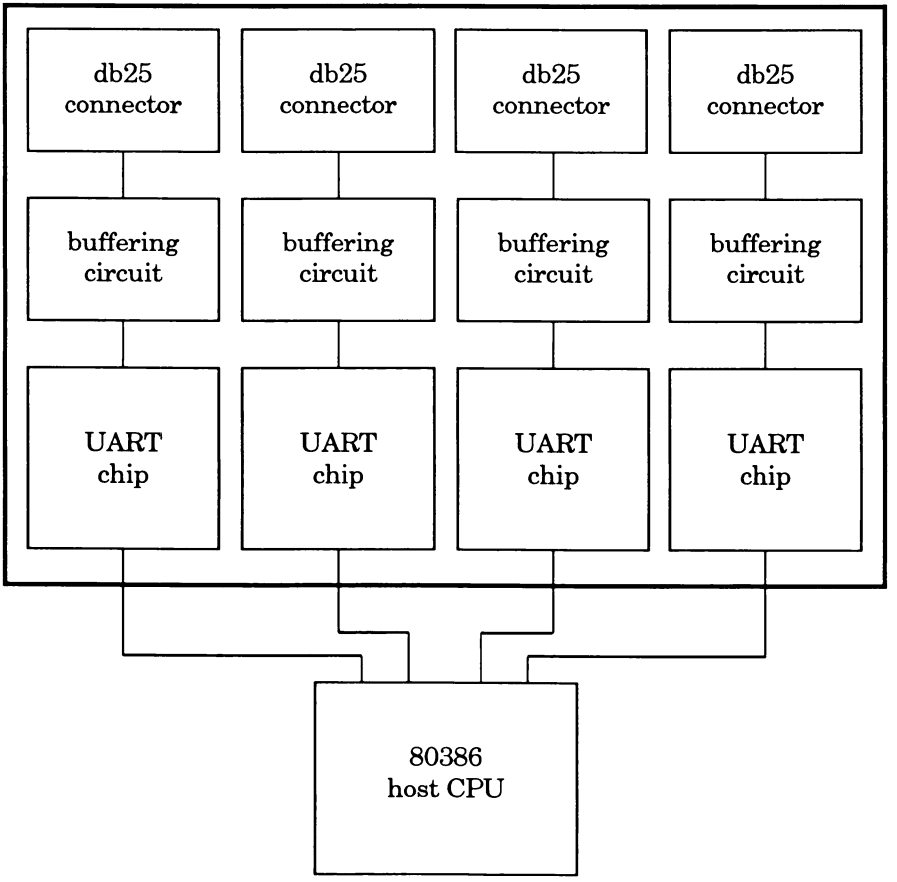

Figure 8 - 1

A 4-port dumb serial card connected directly to the host CPU.

In order to reduce the processing overhead on the CPU when a number of serial ports are to be used "smart" serial cards may be used. These cards have a CPU on the card which is dedicated to servicing the ports and interacting with the host CPU. Figure 8-2 show the basic layout of a 4-port smart card. These cards typically contain a special block of RAM memory which can be read and written by both the on-board CPU and the host. Smart cards provide additional ports with only a negligible amount of increase in processing load on the host CPU. All that is required of the host is to copy data to and from the dual port RAM. The on-board CPU does the rest.

Figure 8-2, on the following page, shows a 4-port smart serial card operated by an on-board CPU.

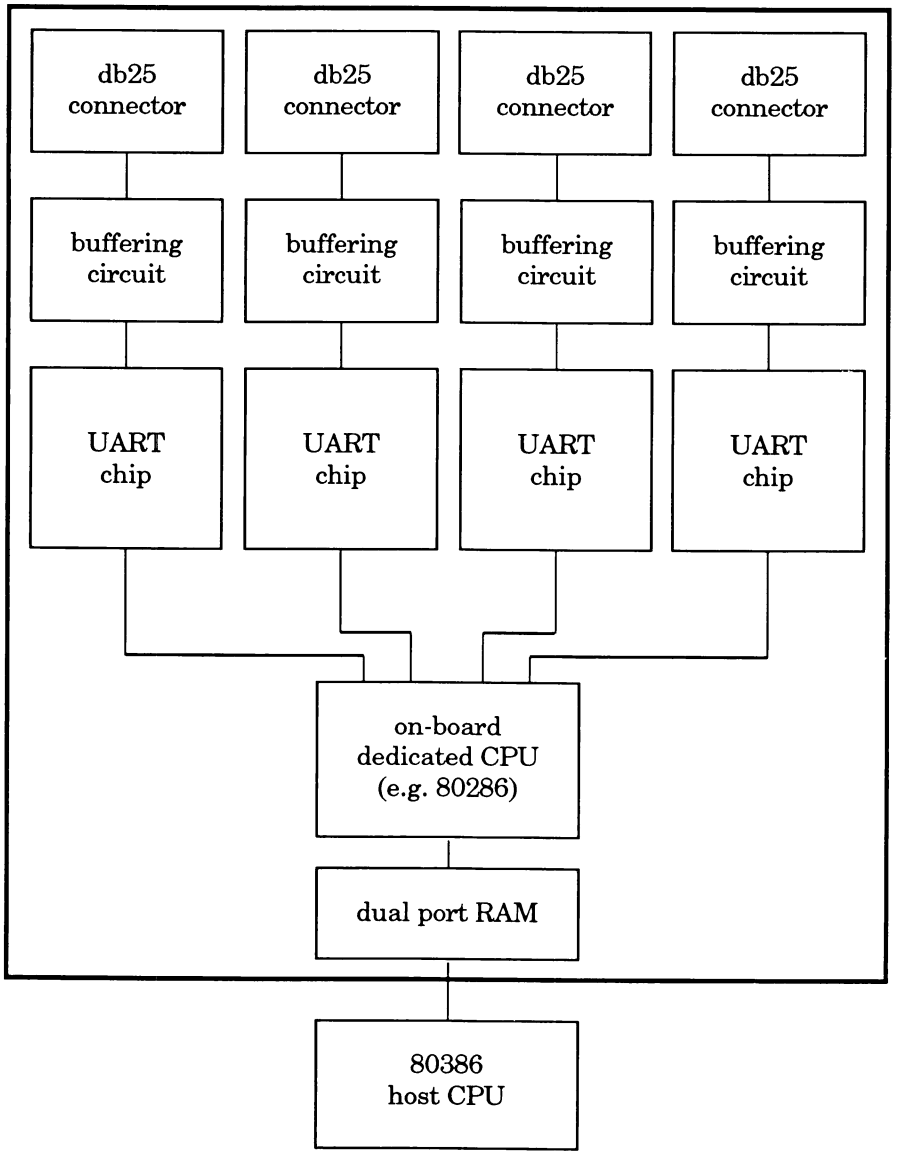
db25 connector
db25 connector
db25 connector
db25 connector
buffering circuit
buffering circuit
buffering circuit
buffering circuit
UART chip
UART chip
UART chip
UART chip
on-board dedicated CPU (e.g. 80286)
dual port RAM
80386 host CPU

Figure 8 - 2

Serial Communications Software Interfaces

The ROM BIOS does provide support for serial communications through a set of int14 system calls. However, the communications method used is what is known as polled mode. A polled mode of serial communications is where an application must continually test the hardware to determine when data is ready in the receiver or when the transmitter is empty and available for the next character. Limitations inherent in this approach make it virtually useless. The constraints on the CPU's attention are too great and the degree of control available through the BIOS calling interface is insufficient to support reliable handshaking and error control at other than very slow baud rates.

The standard INT 21H system calls for serial communication as defined by PC/MS-DOS also work in polled mode since these system calls are basically just an interface to the BIOS at a higher level. When a serial communications device driver is not used, PC-MOS defaults to the same polled mode through its INT 14H and INT 21H interface.

Application programs, such as a telecommunications package, access a serial port by taking direct control of the hardware rather than relying on the BIOS services. Polled mode is also unsuitable for communications with the serial terminal type of workstations used in a multiuser environment. When serial workstations are to be used with MOS, a serial device driver must be used to provide the degree of hardware interaction necessary for high speed operation.

When a communications device driver is installed the serial port's circuitry is programmed to operate in a different manner. Rather than continuously polling the port, the CPU will be interrupted in whatever it is doing when a serial port requires attention. Once the port is serviced, the CPU resumes the function it was performing before the interrupt occurred.

Whenever a character arrives at the serial port from a peripheral device, the serial port chip will generate what is referred to as an IRQ interrupt to gain the CPU's attention. This is an acronym for Interrupt ReQuest. IRQ's will be covered in more detail later in this chapter and in Chapter 9. When a transmit operation is taking place, the serial port chip will generate an IRQ after each character has been transmitted so that the next character may be loaded into the port. This approach allows for much higher baud rates and better overall system throughput.

The type of serial communications device driver used depends on the hardware. The driver supplied with MOS, called $SERIAL.SYS, supports interrupt driven communications with serial ports of the dumb variety. As illustrated earlier in this chapter, a dumb port is one which is operated directly by the host CPU (in contrast to the smart variety, which is managed by a dedicated CPU). $SERIAL.SYS can be used with a system's built-in serial ports as well as those on an add-in board of the dumb variety.

When installed, a serial driver replaces the BIOS INT14 interface with its own extended version (refer to the PC-MOS Technical Reference Manual for details on this). $SERIAL.SYS can operate up to 32 serial ports in interrupt driven mode and, by using the MOS SERINIT command or an INT14 call, it is possible to use a maximum baud rate of 115K.

A high degree of customization in the operating parameters for each port is provided. Handshaking, buffer size, IRQ level, and other special configuration parameters may be specified for each port placed under the control of $SERIAL.SYS.

When using a smart serial card, you must use a matching device driver which has been designed to MOS's specifications. The dedicated processor on the smart board operates the serial ports in their interrupt driven mode. However, these IRQ's are entirely separate from those of the host CPU. The exact interface implemented between the host CPU and the dedicated CPU depends on the board's design. In general, the device driver will insulate the user from having to be concerned with this beyond having to reserve some upper memory away from FREEMEM, and possibly having to avoid using a certain host IRQ level.

Physical and Logical Serial Ports

When a serial port is accessed through the COM1 or COM2 device driver interface the access is said to be at a logical level. Accessing a port directly through its I/O address (e.g. 03F8h or 02F8h) is an example of a physical type of access. Theoretically, when data is read from or written to a logical COMn device driver, the actual physical port which is used could be located at any valid I/O address.

The primary reason for using device drivers is to provide isolation between the logical and physical I/O access layers. However, in a PC/AT type of computer the standard has been established for the first serial port, COM1, to always be addressed at I/O location 03F8h. When a second port is installed, its standard I/O address is 02F8h.

As a result of the PC/AT hardware standard, the association between the logical COM1 device of PC-DOS and the physical 03F8h serial port is a fixed relationship (with the same being true for 02F8h and COM2). In most application programs where a reference to COM1 or COM2 is made in a setup menu or the documentation, what is actually being referred to is the corresponding physical port rather than the logical COM device. Practically any application program which interacts with a serial port will do so through direct hardware programming rather than use the device driver interface or the INT14 services provided by the BIOS.

When a serial communications device driver is not specified in CONFIG.SYS, a PC-MOS system will default to the same logical-to-physical relationship that exists in a PC-DOS system. With the high baud rates, flexible handshaking control and better throughput afforded by an interrupt driven type of serial driver, reasons not to use one are hard to find.

Accessing a serial port purely through the logical device driver interface becomes practical, and even necessary, in the case of supporting serial terminal workstations. With regards to serial printers and similar peripherals; while the use of a serial driver is not mandated, it makes little sense not to use one. The $SERIAL.SYS device driver included with PC-MOS is an example of such a driver.

A serial driver provides two important features: isolation between the logical and physical I/O layers, and interrupt driven I/O. Interrupt driven serial communications is essential for MOS to be able to interact with serial workstations in a multitasking environment. MOS must attend to too many other functions to be repeatedly polling even one serial port.

Isolation is important to allow flexibility in the configuration of a system. Ports which will be used with serial workstations can be placed under the control of the serial driver, while any ports which must be available for applications to program directly can be left alone.

Figure 8-3 illustrates a computer system with two serial ports on the motherboard and an expansion card which provides four more "dumb" ports.

The use of IRQ4 by a built-in 03F8 port is usually fixed by the internal circuitry. When a second built-in port is included, its standard address is 02F8h and it interrupts the host CPU on IRQ3. The same IRQ level can be used by more than one port as long as all ports are managed by one serial driver. Multiport cards typically allow the IRQ level to be selected by on-board switches or jumpers.

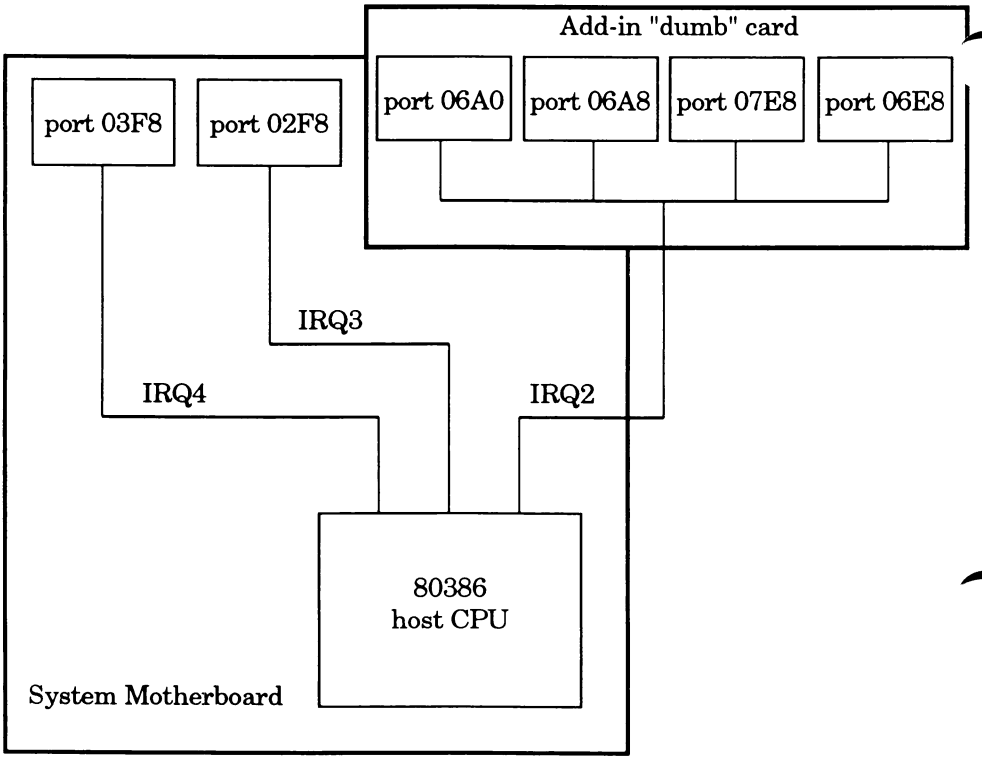

Figure 8 - 3

A system with two built-in serial ports and 4 additional ports on an expansion card.

The $SERIAL.SYS Device Driver

In MOS, the device driver $SERIAL.SYS is used to place simple (dumb) serial ports under interrupt driven control. A serial board with a dedicated CPU (a smart board) must use a device driver written especially for it, and the syntax conventions will likely be different than what is about to be described here for $SERIAL.SYS.

To define the ports to $SERIAL.SYS, each port's address must be listed in a parameter line following the CONFIG.SYS statement DEVICE=SERIAL.SYS. If you need to support 6 serial terminal workstations, you could use the following line:

```
DEVICE=$SERIAL.SYS /AD=03F8,IN=4/AD=02F8,IN=3/AD=06A0,IN=2~
/AD=06A8, IN=2/AD=07E8,IN=2/AD=06E8,IN=2
```

The AD= parameter identifies the port address and the IN= parameter identifies the IRQ level that port will use. The $SERIAL.SYS statement above will result in logical COM1 being associated with physical 03F8h, COM2 with 02F8h, COM3 with 06A0H, COM4 with 06A8H, COM5 with 07E8H and COM6 with 06E8H. The tilde (~) at the end of the first line tells MOS to regard the next CONFIG.SYS line as an extension of the current one. Long lines will often be required when entering port specifications with a serial driver.

The association between the port's address and logical port number is not predetermined as it is in DOS (where COM1 always means 03F8h). With a serial device driver, the definition of ports is much more flexible. Logical-to-physical port numbering depends entirely on the order of the declarations in the driver's CONFIG.SYS parameter line. For example, the following statement will result in port 07E8 being driven by the COM1 device, 06A0 by COM2, etc.

```
DEVICE=$SERIAL.SYS /AD=07E8,IN=2/AD=06A0,IN=2/AD=06A8,IN=2~
/AD=03F8,IN=4/AD=06E8,IN=2/AD=02F8,IN=3
```

MOS supports COM1 through COM24 based on the number of ports declared to the serial driver or drivers. When specifying a port number in the ADDTASK command, only the number is used -- leave off the COM, for example:

```
ADDTASK 256,,,AUTOEXEC,PCTERM,1,19200
```

Three other optional specifications which may be defined in $SERIAL.SYS for each port are the input buffer size (IB=), output buffer size (OB=), and the type of handshaking to be used (HS=). Refer to the PC-MOS User Guide for details on the default values and range of values for theses parameters. An important thing to note about $SERIAL.SYS specifications is that their definition carries on unless they are explicitly re-specified. For example, the statement below takes advantage of this fact, showing that once IRQ level 2 is defined (by the IN=2 after address 06A0) it need not be redeclared for any subsequent ports which use the same IRQ level.

```
DEVICE=$SERIAL.SYS /AD=03F8,IN=4/AD=02F8,IN=3/AD=06A0,IN=2~
/AD=06A8/AD=07E8/AD=06E8
```

While convenient in many cases, you must also be careful with this automatic presumption feature. If you specify OB=2000 for the first port, you should be aware that this will be in effect for all other ports too. To eliminate confusion, the most reliable method can be to explicitly specify all definitions associated with each port.

Enlarging the output buffer size to 2000 bytes or greater for a port associated with a serial terminal workstation will improve system response when a full screen update or PAMswitch is done to one of these workstations. This works because the entire text screen's contents can be copied into the serial driver's buffer in one operation, allowing the screen updating process to run to completion immediately rather than having to wait for the serial driver to send data to the terminal. The sizing of output buffers is a balancing act between memory and performance -- larger buffers will improve throughput but consume more of the SMP.

Most telecommunications programs require direct access to the 03F8h port. If this port has been defined to $SERIAL.SYS, there will be a conflict. The following line leaves 03F8 available for access by other programs and results in COM1 driving 06A0, COM2 driving 06A8, etc.

```
DEVICE=$SERIAL.SYS /AD=06A0,IN=2/AD=06A8/AD=07E8/AD=06E8~
/AD=02F8,IN=3
```

The fact that many communications programs, mouse drivers, etc. refer to COM1 or COM2 in their setup procedures, but actually operate directly with the hardware, can be confusing.

Chaining Multiple Drivers

In some cases it may be desirable to install both a dumb multiport serial card and a smart card in the same computer. Such a situation could arise when you need to add more ports to a system which already has a dumb multiport board and you wish to invest in a smart board rather than another dumb board. $SERIAL.SYS would be used to drive the ports on the dumb board, with the smart board using the driver supplied with it.

The first of these two drivers to appear in CONFIG.SYS would have its logical port numbers assigned starting with COM1. The first port declared to the second driver would be assigned the next logical port number following the last port of the first driver. For example, if the first driver is put in charge of 4 ports, the first port controlled by the second driver will be known by the device name COM5. The MOS INFO command displays a summary of which ports are managed by which driver.

Setting Up a Workstation Port

A workstation can be connected to a built-in serial port or a port on a multiport expansion board. Smart serial cards offload communications processing from the host CPU to improve overall system throughput. Serial terminal workstations can typically operate up to a maximum speed of 38400 baud. MOS is capable of 115K baud. The speed of the terminal is the limiting factor. When running serial cables for a workstation don't coil up any extra cable as this can cause a degradation of the communications signal. Stretch out any extra cable or have the cable cut to the exact length.

Prior to release 4.00, when ADDTASK was used to start a PCTERM type of serial workstation task, the XPC handshaking protocol was established by the PCTERM.SYS terminal driver regardless of any type of handshaking that may have been defined to $SERIAL.SYS. Although the XPC protocol will be used in the majority of cases, to provide for special cases a new feature was introduced in release 4.00. Now, if an explicit declaration is made on the $SERIAL.SYS parameter line it will be used, with XPC being the default when no explicit declaration exists. The workstation must, of course, be configured to use the matching protocol.

When selecting a size for the workstation's $SERIAL.SYS output buffer, choose one which is large enough to handle the amount of data involved in a full screen update. This will improve system throughput when paging through a document in a word processor or when PAMswitching. 2000 bytes is typically a good size for a text mode. Any larger will probably be just a waste of memory. If remote access will be supported through a modem, it is important to establish security controls. See Chapter 10 for a detailed example.

An option with $SERIAL.SYS which was introduced with release 3.00 of MOS allows greater control of the line noise inherent in remote serial workstations connected with modems. Specifying CN=R for a port causes the serial driver to ignore any data arriving at the port unless the carrier detect signal is active. Modem lines are especially susceptible to noise when a dial-up phone line connection is being made or broken. Through the use of CN=R, data is only received when the modem and phone line are in a stable state. Certain smart serial cards also support this option, although the parameter used will be different.

Setting Up a Telecommunications Port

If you are planning to use telecommunications programs with a modem it is usually best to keep the built-in 03F8h port available since most communications programs are designed to use this port. If a port will only be used for telecommunications then do not include the modem port's address in the $SERIAL.SYS parameter line.

Setting Up a Shared Modem Port

As stated above, when a port will be used for modem telecommunications it must not be under the control of $SERIAL.SYS. Doing so will cause conflicts, since two different software entities will be trying to operate the same port in an interrupt driven mode. For remote workstation access, the port must be controlled by $SERIAL.SYS and for telecommunications use it must not be.

The MOS DSPORT command is available to accommodate the case where a modem port needs to be shared between remote workstation access and use with telecommunications programs. This command instructs the serial device driver to temporarily ignore a certain port. Note that this command only works for ports which have a unique IRQ level. To understand this more clearly, consider the following $SERIAL.SYS parameter line:

```
DEVICE=$SERIAL.SYS /AD=03F8,IN=4/AD=06A0,IN=2/AD=06A8
```

With this port configuration, only one port is using IRQ level 4. This is the COM1 port with address 03F8. Both COM2 and COM3 are sharing IRQ level 2. In this situation, a MOS DSPORT 1 command may be issued to temporarily disable the serial driver's control of the 03F8 port. However, since the 06A0 and 06A8 ports both share IRQ2, the MOS DSPORT 2 command can not be used.

For a typical sample case, say that a system has a serial workstation task in existence, as task #1. This task uses port 03F8h with a modem, and the port is the first one declared to $SERIAL.SYS, making it port #1.

When a remote user dials in, they can access the host just like any other user except for the baud rate limitation imposed by the modem and phone line. When the modem is not in use by a remote workstation and you want to use it to access a bulletin board or mainframe, issue the following commands before starting up your telecommunications program:

```
REMTASK 1
MOS DSPORT 1
```

When you are done with this use of the port, using ADDTASK to restart task #1 will automatically re-activate the port in terms of $SERIAL.SYS control. Alternatively, any application or system process which will call upon the serial communications driver to initialize the port will re-activate it. The MOS SERINIT command falls into this category.

Setting Up a Printer Port

The MOS SERINIT command must be used to initialize a serial printer port before it is used. In addition, you will need a MOS ROUTE command to re-direct LPT device data to the appropriate COM device (see Chapter 7). With regards to handshaking; the port definition in the serial driver must match the type for which the printer is configured. When hardware handshaking is to be used, the printer cable must also be appropriate per the manufacturer's specifications. For software handshaking (e.g. XON/XOFF or XPC), you can generally use a three-wire cable.

LANLink

LANLink™ is used to interconnect two or more computers in a network. Both peer-to-peer and server/satellite topologies are supported. In addition, machines may be linked through the use of serial ports or the standard parallel ports which are usually used to connect printers.

When serial ports are used, the same system interconnection can alternately provide for a workstation-to-host relationship. This can be accomplished by running a terminal emulator program such as PC-EmuLink in what is otherwise the satellite machine. At the server end, ADDTASK will be used to start a serial workstation task. When you later wish to activate a server-to-satellite network connection, invoke LANSERVE.COM within the workstation task at the host/server machine. The satellite machine will now function as a standalone computer with network access to the server.

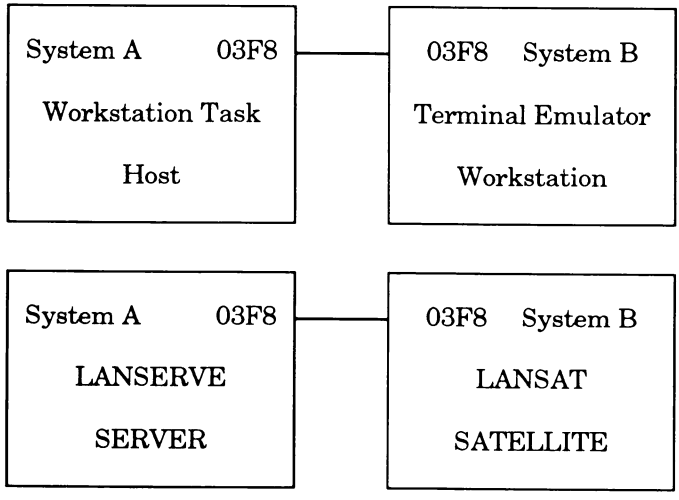

Figure 8 - 4

To clarify this further, figure 8-4 shows two computers interconnected through their 03F8 serial ports. Within System A, this port would be declared to $SERIAL.SYS as COM1. A portion of System A's CONFIG.SYS would appear as follows:

```
DEVICE=C:\$SERIAL.SYS /AD=03F8,IN=4
DEVICE=C:\SERVER.SYS
```

Within System A's AUTOEXEC.BAT, the ADDTASK command would be used to start a serial workstation type of task. The ELTERM.SYS terminal device driver is specified since PC-Emulink will be used in the workstation machine.

```
ADDTASK 384,,,STRTBAT,ELTERM,1,19200
```

Within System B, the workstation machine, two important CONFIG.SYS declarations would appear as follows:

```
DEVICE=C:\$SERIAL.SYS /AD=03F8,IN=4
DEVICE=C:\LANSAT.SYS /03F8,115200,D,1
```

The $SERIAL.SYS driver is required for PC-EmuLink to function. Installing LANSAT.SYS as shown will allow the two computers to operate through the serial link at the highest possible transfer rate of 115,200 bits per second.

Now, when PC-EmuLink is invoked within System B, workstation access can be made to System A. When you want to change the relationship of these two machines to that of a server and satellite, the following command would be invoked within the workstation task of System A:

```
LANSERVE /03F8,115200,C,1
```

Upon exiting PC-Emulink in System B, it will be possible for this computer to access the C: volume of System A as the D: drive. By modifying the parameters after LANSAT.SYS and LANSERVE.COM additional drives could be supported across the network link, as well as LPT devices.

In the case were a port will be dedicated to LANLink usage there is no need to declare the port to $SERIAL.SYS. It won't hurt anything to do so, but it is a waste of SMP space since additional buffers must be allocated by the serial driver for each port declared.

When running serial cables for LANLink, especially when using high baud rates, don't coil up extra cable as this can introduce data errors. Stretch out any extra cable or have your cables cut to fit. Also, be very careful with the EXCEPT and ONLY commands. These commands will not work properly in all cases across a network. When a network re-director shell intercepts the disk system calls, the shell will not know about any exception lists set up by EXCEPT and ONLY. This is described in more detail in Chapter 5.