ecute programs, but the memory diagnostics indicate that the memory is not working properly, probe your project board for correct logic and correct timing at the memory chips.

4.4 This experiment requires a small memory on a project board for which wiring can be easily changed. The project board of Experiment 4.3 is quite suitable for this purpose. The objective of this experiment is to examine failure modes of static RAM when setup times fail to meet chip specifications.

a) Disconnect one address line, A9, from the address driver output. All chips should have this pin floating. Exercise your project board and determine the effects of this change.

b) Create a delay by passing A9 through a series of six inverters of a 7404 chip. The lead inverter should be driven by the address-bus driver, and the last inverter in the chain should drive the address lines of the memory chips. Observe the effect of this change on an oscilloscope and determine whether the address setup time has been violated. Add another six gates in the chain if necessary so that the address A9 definitely changes state during a period when it should be stable. Exercise the memory and observe the effects of this change.

c) Remove the delay from A9 and insert the delay into the WRITE line so that WRITE changes at the 2114 after CHIP SELECT. Observe the effects of this change.

5 / SERIAL INTERFACING

This chapter is the first of the chapters that cover detailed information on interfacing microcomputers to specific devices. A natural starting point is the serial interface because it is probably the least complex electrical interface to implement. It requires only one signal wire to carry all data flow in one direction. (Two wires are required for two-directional flow—that is, one wire for input data and one for output data.) Serial interfaces do, however, have some logic to convert between parallel and serial data streams, and this logic is not necessary for other types of interfaces. But because serial links have a small number of wires and a correspondingly low cost, they are standardized into a few widely used protocols for which there exist LSI interface chips. The large volume of production for these interfaces has reduced the cost per serial port to a few dollars for parts in spite of the additional logic required for transforming data between parallel and serial data streams.

The structure of a typical serial I/O port is shown in Fig. 5.1. Note that the structure is similar to the general port structure outlined earlier in this textbook. That is, the port contains a bus interface through which the microprocessor can send commands to the port, read port status, and access input/output data registers in the port. An interrupt line notifies the microprocessor of the completion of an operation. What distinguishes this port from the general structure of an I/O port is the conversion that occurs between serial and parallel data streams. The bus interface with the microprocessor is a parallel interface through which eight (or sixteen) data bits are transmitted in a single transaction. The interface with the outside world is a serial interface, in which those same data bits are transmitted on a single output wire or received on a single input wire, with the bits appearing on these lines sequentially in time.

The figure shows an output register XMIT loaded in parallel from the microprocessor data bus. This register, in turn, is connected in parallel to a shift register. Data loaded into XMIT are subsequently passed to the shift register, then shifted serially onto a single output line. Input data are received sequentially and passed to a shift register, which collects the bits until the register is full. Subsequently, the input data are passed in parallel from the shift register to the register identified as RCV, and from there to the microprocessor bus.

The reason that both the transmit and receive sections have two registers relates to the need to buffer data in transit through the port. It is possible, for example, to combine RCV and its corresponding shift-register into one register, but this creates a serious timing constraint in the port. There may be an indefinite delay between the receipt of a full 8-bit datum from the outside world, and the availability of a bus cycle during which that datum can be transferred to the microprocessor. Unless the port can safely buffer a received datum for a short period of time, there is a very high risk that a received datum will be overwritten by the leading bits of the next arriving character. Hence, RCV gives the

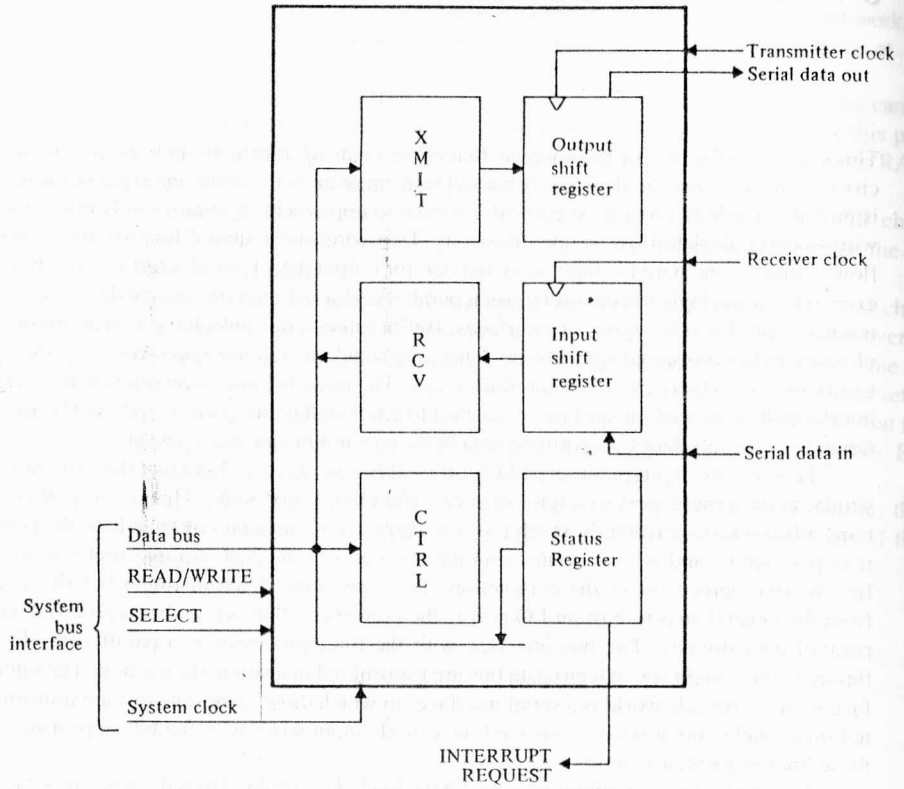

FIGURE 5.1 The structure of a typical serial I/O port.

necessary additional buffering for input data. As each successive bit of the input stream appears, the shift register shifts to make room for it. When the last bit of a character is received, the completed character moves to RCV, from where it is eventually moved to the processor. Thus, by buffering a received datum in RCV, the shift register is free to accept the next byte of information from the serial data stream.

Once a character is transferred to the buffer register RCV, the microprocessor has an opportunity to remove the character from the I/O port and copy it to memory where further operations can be done. The microprocessor can determine the availability of the character either by testing the status bit of the port, which distinguishes between a full buffer and an empty buffer, or by fielding an interrupt generated at the completion of character reception. In the latter case, the microprocessor must distinguish the interrupting port from all other ports by testing port status or by accepting a transfer vector from the port in response to an interrupt-acknowledge signal.

Although we have focused on the input stream to indicate why a buffer register is commonly used, the output stream also benefits from a buffer register configured as in the figure. In this case, the microprocessor can fill the output buffer while the last character is still being shifted out of the shift register. If the output buffer XMIT is full when the shift register empties, the XMIT datum is automatically transferred to the shift register, thereby freeing XMIT for another datum. The buffering of the output register improves performance, but is not essential for the correct operation of the interface. If XMIT and its corresponding shift register are combined into a single register, then when that register empties, it remains empty until reloaded from the microprocessor bus. This leaves the transmit line in an idle state until the microprocessor services the transmitter again. To eliminate the idle period in the absence of buffering, the microprocessor must respond to a service request in the very brief interval between the end of one character and the beginning of the next. With buffering as shown in the figure, when an output datum is transferred from XMIT to the shift register for output, XMIT can accept the next datum from the microprocessor at any time during the transmission of the datum from the shift register. When the shift register outputs the last bit of its datum, it reloads from XMIT, and transmits the first bit of that datum with no idle time between data. Thus, this scheme eliminates idle time on the output line, provided that the microprocessor reloads XMIT before the shift register empties again, but this reload interval is at least one 8-bit character interval, and not just the brief period between characters that is available in an unbuffered transmitter.

Because buffering is so important to correct operation and good performance, virtually all forms of serial I/O ports use buffering as shown in Fig. 5.1. The cost of buffering is negligible when implemented in LSI form; and, in fact, some interface devices contain several buffer registers per port in order to attain additional reliability and higher performance.

5.1 SERIAL I/O PROTOCOLS

A *communications protocol* is a convention for data transmission that includes such functions as timing, control, formatting, and data representation. Protocols generally fall into two categories depending on the clocking of the data on the serial link:

1. *Asynchronous protocols:* Successive data appear in the data stream at arbitrary times, with no specific clock control governing the relative delays between data.
2. *Synchronous protocols:* Each successive datum in a stream of data is governed by a master data clock and appears at a specific interval in time.

Commonly used synchronous and asynchronous protocols deliver serial data in 8-bit characters. Asynchronous protocols treat each character as an individual message, and the characters appear in the data stream at arbitrary relative times. Within each character, however, the bits are transmitted at a fixed predetermined clock rate. Hence these protocols are actually synchronous within a character and asynchronous between characters, but they are called asynchronous because the asynchronous timing between characters is their distinguishing characteristic.

Synchronous protocols produce a stream of data at a fixed clock rate with the clock governing not only the bits within a character but the character-to-character timing as well. Asynchronous protocols tend to be simpler, slower, and more widely used than synchronous protocols. In asynchronous protocols, since each character in the data stream is independent of the preceding and following characters, neither the transmitter nor the receiver needs to retain state information while processing a sequence of characters in a message. Although simplicity has its attractions, the asynchronous protocols have at least a 20% overhead per character of control information transmitted with each character. This overhead is substantially lower in synchronous protocols where control information is not required on a per-character basis. These protocols group characters into blocks of characters and place the control information at the beginning and end of blocks. Hence the control information per data byte can be made much smaller, provided that blocks are long; however, the consequence is that the receiver and transmitter must retain more state information. The synchronous protocol also has hardware to extract a synchronous clock from incoming data, which is a more demanding operation than the edge detection performed by an asynchronous receiver. So synchronous protocols tend to require more complexity in the transmitter and receiver than do asynchronous protocols, but they lead to greater effective use of communications bandwidth.

Another important difference between the protocols is the error correction and detection capability of the basic protocol. In the asynchronous protocols each message is a single 8-bit datum. Some protection from errors is available through the use of one of those 8-bits as a parity check. With one parity check, any single-bit error or odd number of bit errors is detectable, but not correctable. But when an even number of bits is incorrectly received, the parity check is "satisfied," and the errors are thus undetectable. Although it is possible to impose error-correction and detection capability on sequences of characters through the use of parity checks over blocks of data, such a capability is outside the specifications of the asynchronous protocol, and is therefore incorporated into system firmware or software and not into the I/O port itself.

However, synchronous transmission protocols, for messages that are typically tens or hundreds of bytes in length, usually include sufficient redundancy to detect the most probable errors in any given block of data; moreover these protocols provide additional information for other control purposes. The use of redundancy at the block level instead of at the character level gives higher error-protection for a given percentage of check bits in the data stream, and thus makes more efficient use of the communications bandwidth. Moreover, with additional control information in each block of synchronous data, higher-level functions can be defined as part of the synchronous protocol and built into the I/O port. One such function, for example, is automatic retransmission of a block of data when a receiver discovers an error in the block. In contrast, retransmission of individual characters sent on an asynchronous channel is very difficult because there is no easy way for a receiver to tell the transmitter which character to retransmit, nor for the transmitter to distinguish between a new character or a retransmitted character. For synchronous protocols, control information in each block can contain the sequence number of the block, so that the receiver can request retransmission of specific blocks. The retransmitted blocks are readily identified by their sequence numbers.

5.2 ASYNCHRONOUS PROTOCOLS

To study asynchronous protocols, we separate the timing conventions from the electrical connection requirements. There is only one timing convention in widespread use, but electrical connections usually follow any one of three systems:

1. RS-232-C.
2. 20 mA current loop.
3. RS-422, RS-423, and RS-449.

RS denotes "recommended standard" and refers to official published standards of the Electronic Industries Association, or EIA. Each of the three interconnection systems is described in detail after we examine timing conventions that apply to all three systems.

Figure 5.2 illustrates the bit timing for a single data byte transmitted serially on an asynchronous link. The idle line is assumed to be in a high or 1 state. Each character begins with a 0 bit, followed by 8 data bits, then followed by 1, 1½, or 2 closing 1 bits. A bit interval is a fixed period of time governed by a local clock in the transmitter and receiver. Signaling frequencies common today are 300 Hz, 600 Hz, 1200 Hz, etc., up to 19.2 kHz. Slower, obsolescent equipment runs at 110 Hz, 134.5 Hz, and 150 Hz. Interfaces often support these rates for the purposes of compatibility, even though new equipment rarely signals at these rates. Within the 8-bit data portion of a character, the data are transmitted least-significant bit first. Hence, the figure shows the pattern in time 1000 0010 for the ASCII (American Standard Code for Information Interchange, pronounced "ask-ee") encoding of the letter A, which when written with the least-significant bit on the right is $0100\ 0001 = 41_{16}$, which readers should recognize as the correct encoding of the letter A. (See Appendix A.)

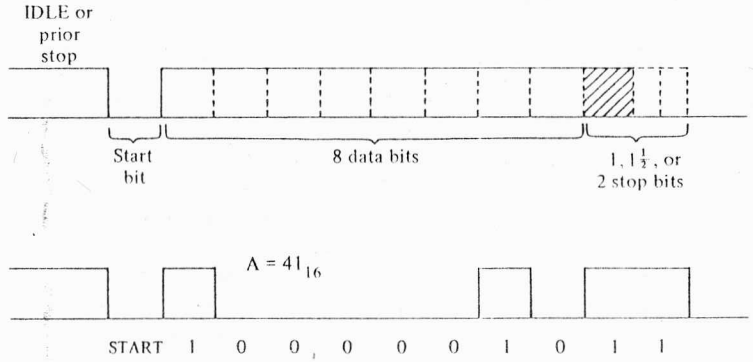

FIGURE 5.2 Asynchronous serial transmission format.

The start and stop bits serve a very important purpose. Obviously they identify the beginning and end of each character; but more important, they permit a receiver to resynchronize a local clock to each new character. Since a character can begin at an arbitrary

time, the receiver has to discover when that edge occurs with such accuracy that it is able to sample the next 10 to 11 bits correctly. The receiver's clock is not identical to the transmitter's clock, so sample points at the receiver will, in general, be spaced less than or greater than a bit period as defined by the transmitter clock. A fast receiver samples successive bits earlier and earlier in a bit period. A slow receiver does the opposite. The receiver must not let its relative clock speed cause it to sample the wrong bit. The best strategy is that the receiver samples each bit as closely as possible to the center of the bit period. If the receiver makes a good estimate of the start of the first bit, then it can sample this bit after waiting for one-half of a bit period after the leading transition on the start bit. Thereafter the receiver waits one bit period and samples again until it reaches the last bit.

This strategy works correctly when the opening transition is known accurately, provided that the receiver clock is close enough to the transmitter clock in frequency that the last bit is sampled within one half-period of its true center position. This means that the receiver clock, in relation to the transmitter clock, cannot gain or lose more than half a bit in position over 10 to 11 clock periods. Hence, we require both clocks to be accurate within a 5% error margin, which is easily realized with today's technology. But this sampling strategy depends on discovering the leading edge of a character, then timing all subsequent samples from this point. To make the leading edge discoverable, the start bit is made different from both the idle-line state and the closing bit of a character. Hence, the leading edge can be observed when the new character occurs on an idle line, or when one character immediately follows another. Because of the skew of the bit transitions caused by variances in thresholds and rise/fall times, the 5% margin of error is a greater margin of error than actually required, but the error margin is still a few percentage points at slow clock rates.

Assuming that the leading transition of a character is readily observable, the next problem is to use this as a time base for sampling the following bits. Most receivers use a fast clock for this purpose. Figure 5.3 shows a receiver clock that runs at 16 times the bit frequency. With this clock, the receiver can determine the beginning of a character to within 1/16 of a bit period. All sampling is done on the basis of the 16× clock in the following steps:

1. When the leading transition of a character occurs, a counter register is cleared.
2. The counter increments for each tick of the 16× clock.
3. When the counter first reaches the value 8, it has reached the middle of the start bit. At this point, the start bit is sampled, and the counter is cleared.
4. Subsequently, each time the counter reaches the value 16, the waveform is sampled, and the counter is cleared. The sampling is repeated until the last stop bit is sampled.
5. If the stop bit (or bits) is correct, the character is accepted, loaded into a buffer, and the process begins again at Step 1.

Although this example uses a 16× clock at the receiver, the receiver clock can be other multiples of the bit rate. Higher receiver clock rates yield greater resolution, but the resolution is not particularly advantageous beyond the resolution available with a 16× clock.

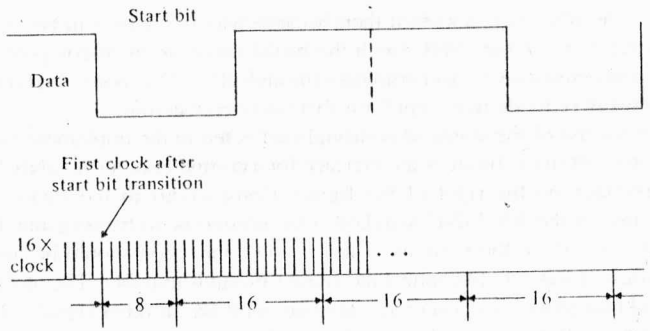

FIGURE 5.3 Character resynchronization with a 16× clock.

Note that the timing constraints are not very critical. The limit on the timing variation between receiver and transmitter is easily met with crystal-controlled clocks, since these circuits have a stability of at least one part in 10^5. Less accurate but acceptable clocks can be implemented with simple RC timing circuits. Of course there is a cost for making the protocol relatively insensitive to clock speed. This is the cost of using a start and stop bit of opposite polarity, which uses at least 20% of the available communications bandwidth. Even this 20% overhead is not quite sufficient to achieve the clock insensitivity required. When transmitting a long sequence of characters without idle time between them, a transmitter that is slightly faster than a receiver may eventually overrun the receiver by one bit, thereby losing a whole character or worse. It is necessary to insert idle periods every so often in the transmitted bit stream to prevent this occurrence, which reduces useful bandwidth even more. We address this topic in greater depth later in this chapter.

The RS-232-C Interface

The RS-232-C Interface Standard is a widely used popular standard of the Electronic Industries Association. The EIA RS standards cover a very broad range of areas, going well beyond communications and computers. The RS-232-C standard was originally developed to foster data communications on public telephone networks. The interface to a telephone network is normally made through a device known as a *modem*, from *mo*dulator-*dem*odulator. This device translates 0s and 1s from a sequence of high and low voltages into a sequence of high and low frequencies, which are compatible with telephone networks. At the other end of a connection, another modem retranslates the frequencies back into a sequence of high and low voltages. The RS-232-C standard, then, originally provided a specification for connecting remote devices together by using the telephone network as an intermediate medium, with interfacing accomplished by modems. In the mid-60s during the early phases of the development of time-shared computers, remote access over serial links was done almost exclusively through telephone

connections. The RS-232-C standard then became widely adopted in both terminal and computer equipment. In the 1980s, with the proliferation of microcomputers, terminals are usually connected directly to computers through RS-232-C ports, and do not use the telephone network or modems except for truly remote connections.

The original use of the standard is strongly reflected in the implementation shown in Fig. 5.4. On the left in the figure is an interface for a computer or terminal that connects to a modem interface on the right of the figure. Conspicuous in the figure are the two grounds defined in the RS-232-C standard. One ground is a chassis ground that is tied directly to the shields in the systems. This ground connection should be made between two devices only if it is safe to connect the chassis grounds together. The other ground is a signal ground that provides a common reference point for all other signals. This connection is mandatory. But because the signal grounds are not necessarily isolated from the chassis ground, RS-232-C has an inherent potential ground-loop problem. While the standard is quite useful for short distances, for longer distances it becomes unreliable and hazardous. The published standard recommends that each device should have a cable not in excess of 50 feet, which permits the total cable length between a pair of devices to reach 100 feet. For longer cables, the standard strongly recommends that other means of interconnection be used. The RS-232-C standard describes 21 signals and a 25-pin physical connector for asynchronous communication. Details of the signals are in Appendix B. The following discussion highlights the more important signals.

Returning to Fig. 5.4, we see that the terminal/computer interface on the left and the modem on the right have a pair of wires dedicated to TRANSMIT and RECEIVE functions. These are compatible signals because TRANSMIT is a modem input and a

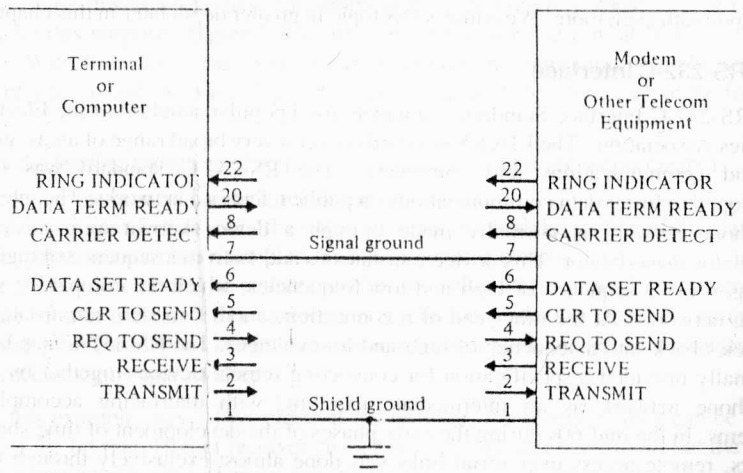

FIGURE 5.4 RS-232-C interface with communications equipment.

computer/terminal output; the converse applies to RECEIVE. Other signals reflect the telephone protocol of the modem. Specifically, REQ TO SEND and CLR TO SEND relate to the characteristics of half-duplex telephone lines. Such lines are capable of bidirectional traffic, but they can send in only one direction at a time. The terminal signals a modem with REQ TO SEND when it has a character to transmit, but the character has to be queued until the modem changes from a receive to a transmit mode. When transmission is possible, the CLR TO SEND signal is returned to the terminal, and transmission can begin. The modem can also turn off CLR TO SEND if the modem moves into the receive mode from transmit mode.

The CLR TO SEND and REQ TO SEND lines are held to a constant voltage when the telephone connection is full-duplex—that is, when transmit and receive functions can occur simultaneously on independent wires. Full-duplex links are commonplace today so that the half-duplex protocol is seldom used with modems; the protocol, however, is built into the serial-interface devices widely available for RS-232-C use, and the control signals are occasionally used for purposes other than the one originally intended.

Two READY signals are included in the RS-232-C standard. DATA SET READY signifies that the modem is operational, and DATA TERM READY is the corresponding signal for computers and terminals. These signals are sometimes connected to the power supply and become asserted when the device is powered on. The DATA SET READY signal for a modem indicates more than just a power-on condition; the modem is actually connected to a communications line, and is not in a test mode or a disconnected state. The READY signals are then passed across the cable link so that the equipment on the opposite end of the cable can sense the condition.

CARRIER DETECT and RING INDICATOR are related to telephone functions. The RING INDICATOR is asserted during the period that a ringing tone is present on the communications line. When ringing is sensed at a computer I/O port, the port should post an interrupt, at which point the computer can initiate a connection to the caller. CARRIER DETECT is a signal that indicates that a remote connection is currently active. If the connection should break for any reason, the CARRIER DETECT signal is lost, and this too should cause an interrupt at a computer I/O port.

The standard incorporates other signals not shown in the figure. These relate to testing and to secondary channels. For interconnections between computers and terminals that do not use a telephone network, the signals in the figure are sufficient for nearly all functions.

Now we come to an interesting problem. Figure 5.4 shows that a computer or terminal can be wired to connect directly to a modem. But when wired in this fashion they cannot be connected directly to each other. The connections show both the computer and terminal transmitting on the same pin and receiving on the same pin. It is obvious that these pins cannot be connected directly together. There are several ways to solve the problem. One way is to build terminals that have the same interface as the modems. Then they can connect directly to computers. But unfortunately these terminals are incompatible with the modems and the telephone network. Another common solution is to correct the problem in the interconnecting cable as shown in Fig. 5.5. Note that the TRANSMITTED DATA and RECEIVED DATA lines are crossed so that the devices can, at least, transmit

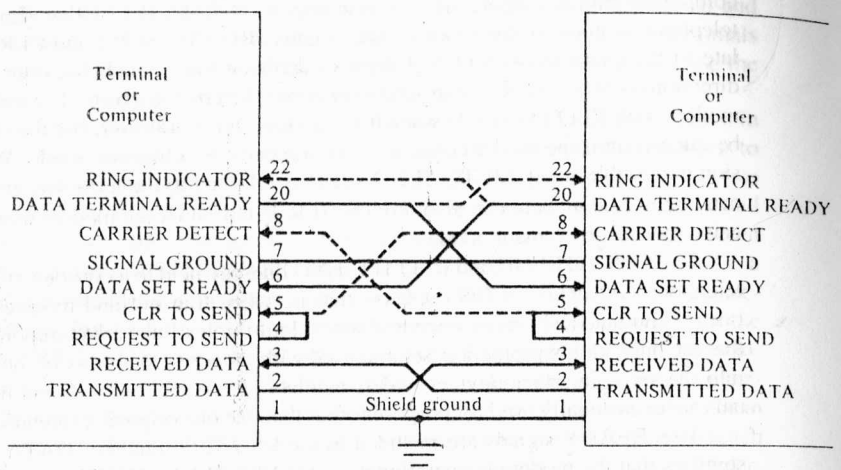

FIGURE 5.5 RS-232-C interface, terminal/computer to terminal/computer.

and receive properly. Given the full-duplex nature of the hard-wired connection, REQUEST TO SEND and CLR TO SEND no longer serve a useful modem-like function. Hence, REQUEST TO SEND is folded back as CLR TO SEND, and a transmission request is thereby always granted. The dotted line shows this signal used as CARRIER DETECT at the other end, since the presence of REQUEST TO SEND is functionally similar to the detection of a carrier in a communications channel. The last set of signals, DATA SET READY and DATA TERMINAL READY are crossed in the cable so that each end of the link can detect the presence of a ready condition on the other end. Note the dotted line that carries the READY signal to the RING INDICATOR input. Dotted lines in this drawing signify connections that are sometimes made in practice, but are less common than the connections, as represented by solid lines, that cross couple or feed back pairs of signals.

Electrical conventions for RS-232-C circuits are shown in Fig. 5.6. The voltage levels are quite different from the TTL and MOS voltage levels. Voltages in RS-232-C systems are symmetric with respect to ground, and are at least 3 V for a logic 0, and −3 V or less for a logic 1. (The standard occasionally uses the terms "Mark" and "Space" to denote logic 1 and logic 0, respectively.) In actual practice, the voltage levels are powered by ±12-V and ±15-V supplies, so that the voltage swing between logic 1 and logic 0 may be 20 or more volts. Inexpensive translator circuits are available for changing the voltage levels. The MC1488 transmitter accepts TTL input levels and produces RS-232-C output levels. The actual output voltage is a function of the supply voltage on the MC1488, and these are usually set at ±15 V or at ±12 V. The MC1489 receiver uses a standard 5 V supply for signal receiving; and this is all that is necessary, although an additional small voltage supply can be connected to the receiver to alter the switching threshold. Although the switching threshold with a 5 V supply is above 0 V, and not symmetric with respect to the signaling voltages, it is well within the specifications for RS-232-C voltages. The receiver also has about 1 V of noise protection through hysteresis.

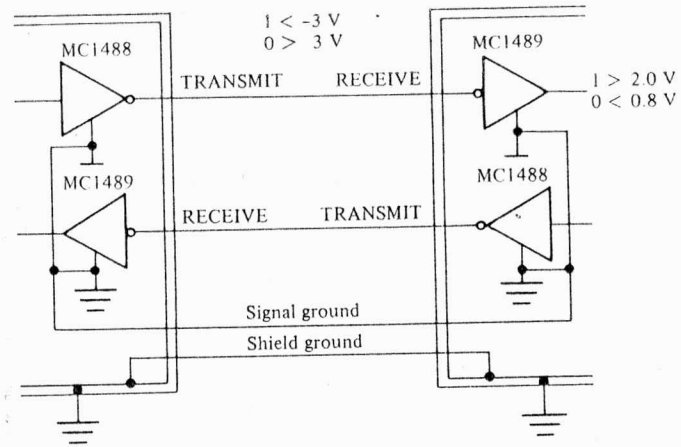

FIGURE 5.6 An RS-232-C interface showing gates for changing voltage levels.

The large voltage swing on RS-232-C signals is required for noise immunity on the communications link. With a common signal-ground between transmitter and receiver, there is no opportunity for double-ended signaling, and therefore common-mode noise is inherently coupled into the signaling system. TTL voltage levels are simply too sensitive to noise to work over long distances unless the common-mode noise can be eliminated. TTL has at best 1.2 V between the logic 0 (0.8 V or less) and logic 1 (2.0 V or greater), so noise voltages of the order of 0.5 V can be severely disruptive. But common-mode noise voltages easily climb to a few volts in the presence of electric motors, photocopiers, typewriters, and the like. Thus the common signal-ground is largely responsible for forcing the higher transmission voltages found in the RS-232-C standard. Even for these voltages the standard covers signaling rates only up to 20 kHz and at distances on the order of 30 m, the maximum distance at which signal grounds can be safely connected.

The 20-Milliampere Current Interface

Another popular serial-interconnection methodology is current controlled, rather than voltage controlled, as in the RS-232-C standard. The current control relies on currents of 20 mA to encode a logic 1, and zero current to encode a logic 0. The basic idea of the electrical connection appears in Fig. 5.7. The transmitter on the left side of the figure is shown with a 20 mA current source and a switch. The receiver on the right-hand side of the figure has a current detector. The transmitter simply opens and closes the switch, which pulses current through the loop and which is sensed by the receiver. The reverse