

16.1 Introduction

RS-232 is one of the most widely used techniques used to interface external equipment to computers. It uses serial communications where one bit is sent along a line, at a time. This differs from parallel communications which sends one or more bytes, at a time. The main advantage that serial communications has over parallel communications is that a single wire is needed to transmit and another to receive. RS-232 is a de facto standard that most computer and instrumentation companies comply with. It was standardized in 1962 by the Electronics Industries Association (EIA). Unfortunately this standard only allows short cable runs with low bit rates. The standard RS-232 only allows a bit rate of 19 600 bps for a maximum distance of 20 metres. New serial communications standards, such as RS-422 and RS-449, allow very long cable runs and high bit rates. For example, RS-422 allows a bit rate of up to 10 Mbps over distances up to 1 mile, using twisted-pair, coaxial cable or optical fibres. The new standards can also be used to create computer networks. This chapter introduces the RS-232 standard and gives simple programs which can be used to transmit and receive using RS-232. The following chapter shows how Turbo Pascal can be used to transmit data through the parallel port.

16.2 Electrical characteristics

16.2.1 Line voltages

The electrical characteristics of RS-232 define the minimum and maximum voltages of a logic '1' and '0'. A logic '1' ranges from -3 V to -25 V , but will typically be around -12 V . A logical '0' ranges from 3 V to 25 V , but will typically be around $+12\text{ V}$. Any voltage between -3 V and $+3\text{ V}$ has an indeterminate logical state. If no pulses are present on the line the voltage level is equivalent to a high level, that is -12 V . A voltage level of 0 V at the receiver is interpreted as a line break or a short circuit. Figure 16.1 shows an example transmission.

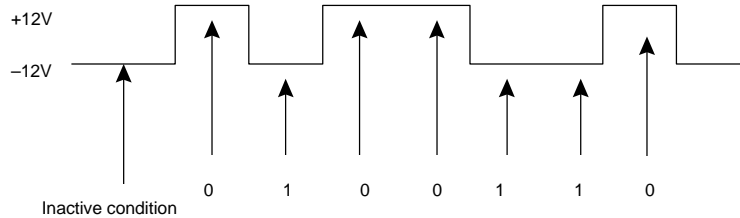


Figure 16.1 RS-232 voltage levels

16.2.2 DB25S connector

The DB25S connector is a 25-pin D-type connector and gives full RS-232 functionality. Figure 16.2 shows the pin number assignment. A DCE (the terminating cable) connector has a male outer casing with female connection pins. The DTE (the computer) has a female outer casing with male connecting pins. There are three main signal types: control, data and ground. Table 16.1 lists the main connections. Control lines are active HIGH, that is they are HIGH when the signal is active and LOW when inactive.

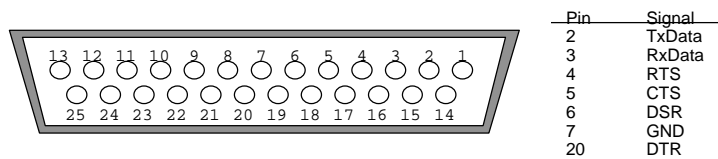


Figure 16.2 RS-232 DB25S connector

16.2.3 DB9S Connector

The 25-pin connector is the standard for RS-232 connections but as electronic equipment becomes smaller there is a need for smaller connectors. For this purpose most PCs now use a reduced function 9-pin D-type connector rather than the full function 25-way D-type. As with the 25-pin connector the DCE (the terminating cable) connector has a male outer casing with female connection pins. The DTE (the computer) has a female outer casing with male connecting pins. Figure 16.3 shows the main connections.

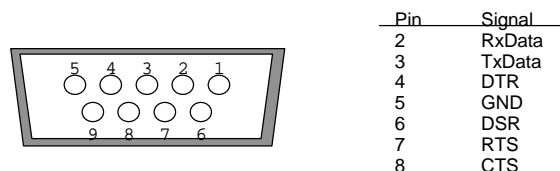


Figure 16.3 RS-232 DB9S Interface

Table 16.1 Main pin connections used in 25-pin connector

<i>Pin</i>	<i>Name</i>	<i>Abbreviation</i>	<i>Functionality</i>
1	Frame Ground	FG	This ground normally connects the outer sheath of the cable and to earth ground
2	Transmit Data	TD	Data is sent from the DTE (computer or terminal) to a DCE via TD
3	Receive Data	RD	Data is sent from the DCE to a DTE (computer or terminal) via RD
4	Request To Send	RTS	DTE sets this active when it is ready to transmit data
5	Clear To Send	CTS	DCE sets this active to inform the DTE that it is ready to receive data
6	Data Set Ready	DSR	Similar functionality to CTS but activated by the DTE when it is ready to receive data
7	Signal Ground	SG	All signals are referenced to the signal ground (GND)
20	Data Terminal Ready	DTR	Similar functionality to RTS but activated by the DCE when it wishes to transmit data

16.2.4 PC connectors

All PCs have at least one serial communications port. The primary port is named COM1 : and the secondary is COM2 :. There are two types of connectors used in RS-232 communications, these are the 25- and 9-way D-type. Most modern PCs use either a 9-pin connector for the primary (COM1 :) serial port and a 25-pin for a secondary serial port (COM2 :), or they use two 9-pin connectors for serial ports. The serial port can be differentiated from the parallel port in that the 25-pin parallel port (LPT1 :) is a 25-pin female connector on the PC and a male connector on the cable. The 25-pin serial connector is a male on the PC and a female on the cable. The different connector types can cause problems in connecting devices. Thus a 25-to-9 pin adapter is a useful attachment, especially to connect a serial mouse to a 25-pin connector.

16.3 Frame format

RS-232 uses asynchronous communications which has a start-stop data format. Each character is transmitted one at a time with a delay between them. This delay is called the inactive time and is set at a logic level high (-12 V) as shown in Figure 16.4. The transmitter sends a start bit to inform the receiver that a character is to be sent in the following bit transmission. This start bit is always a '0'. Next, 5, 6 or 7 data bits are sent as a 7-bit ASCII character, followed by a parity bit and finally either 1, 1.5 or 2 stop bits. Figure 16.4 shows a frame format and an example transmission of the character 'A', using odd parity. The rate of transmission is set by the timing of a single bit. Both the transmitter and receiver need to be set to the same bit-time interval. An internal clock on both sets this interval. These only have to be roughly synchronized and approximately at the same rate as data is transmitted in relatively short bursts.

Error control is data added to transmitted data in order to detect or correct an error in transmission. RS-232 uses a simple technique known as parity to provide a degree of error detection.

A parity bit is added to transmitted data to make the number of 1s sent either even (even parity) or odd (odd parity). A single parity bit can only detect an odd number of errors, that is, 1, 3, 5, and so on. If there is an even number of bits in error then the parity bit will be correct and no error will be detected. This type of error coding is not normally used on its own where there is the possibility of several bits being in error.

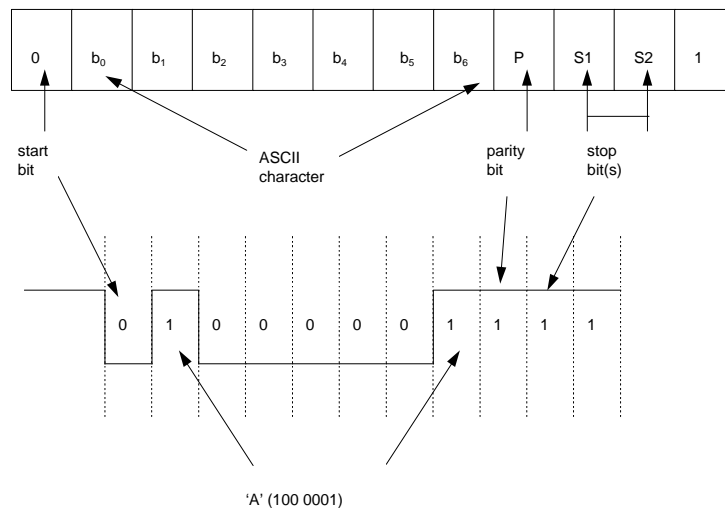


Figure 16.4 RS-232 frame format

Baud rate

One of the main parameters which specify RS-232 communications is the rate of transmission at which data is transmitted and received. It is important that the transmitter and receiver operate at, roughly, the same speed.

For asynchronous transmission the start and stop bits are added in addition to the 7 ASCII character bits and the parity. Thus a total of 10 bits are required to transmit a single character. With 2 stop bits, a total of 11 bits are required. If 10 characters are sent every second and if 11 bits are used for each character, then the transmission rate is 110 bits per second (bps). Table 16.2 lists how the bit rate relates to the characters sent per second (assuming 10 transmitted bits per character). The bit rate is measured in bits per second (bps).

	Bits
ASCII character	7
Start bit	1
Stop bit	2
Total	10

Table 16.2 Bits per second related to characters sent per second

<i>Speed(bps)</i>	<i>Characters/second</i>
300	30
1200	120
2400	240

In addition to the bit rate, another term used to describe the transmission speed is the baud rate. The bit rate refers to the actual rate at which bits are transmitted, whereas the baud rate relates to the rate at which signalling elements, used to represent bits, are transmitted. Since one signalling element encodes one bit, the two rates are then identical. Only in modems does the bit rate differ from the baud rate.

16.4 Communications between two nodes

RS-232 is intended to be a standard but not all manufacturers abide by it. Some implement the full specification while others implement just a partial specification. This is mainly because not every device requires the full functionality of RS-232, for example a modem requires many more control lines than a serial mouse.

The rate at which data is transmitted and the speed at which the transmitter and receiver can transmit/receive the data dictates whether data handshaking is required.

16.4.1 Handshaking

In the transmission of data there can be either no handshaking, hardware handshaking or software handshaking. If no handshaking is used then the receiver must be able to read the received characters before the transmitter sends another. The receiver may buffer the received character and store it in a special memory location before it is read. This memory location is named the receiver buffer. Typically, it may only hold a single character. If it is not emptied before another character is received then any character previously in the buffer will be overwritten. An example of this is illustrated in Figure 16.5. In this case the receiver has read the first two characters successfully from the receiver buffer, but it did not read the third character as the fourth transmitted character has overwritten it in the receiver buffer. If this condition occurs then some form of handshaking must be used to stop the transmitter sending characters before the receiver has had time to service the received characters.

Software handshaking involves sending special control characters. These include the DC1-DC4 control characters. Hardware handshaking involves the transmitter asking the receiver if it is ready to receive data. If the receiver buffer is empty it will inform the transmitter that it is ready to receive data. Once the data is transmitted and loaded into the receiver buffer the transmitter is informed not to transmit any more characters until the character in the receiver buffer has been read. The main hardware handshaking lines used for this purpose are:

- CTS – Clear To Send.
- RTS – Ready To Send.
- DTR – Data Terminal Ready.
- DSR – Data Set Ready.

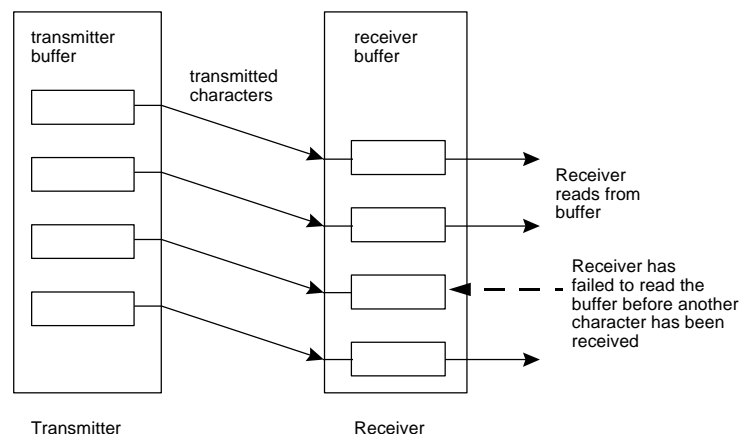


Figure 16.5 Transmission and reception of characters

16.4.2 RS-232 set-up

Windows 95/NT allows the serial port setting to be set by selecting Control Panel → System → Device Manager → Ports (COM and LPT) → Port Settings. The settings of the communications port (the IRQ and the port address) can be changed by selecting Control Panel → System → Device Manager → Ports (COM and LPT) → Resources for IRQ and Addresses. Figure 16.6 shows example parameters and settings. The selectable baud rates are typically 110, 300, 600, 1200, 2400, 4800, 9600 and 19200 baud for an 8250-based device. With a 16650 compatible UART speed also gives enhanced speeds of 38400, 57600, 115200, 230400, 460800 and 921600 baud. Notice that the flow control can either be set to software handshaking (Xon/Xoff), hardware handshaking or none.

The parity bit can either be set to none, odd, even, mark or space. A mark in the parity option sets the parity bit to a '1' and a space sets it to a '0'.

In this case COM1: is set at 9600 baud, 8 data bits, no parity, 1 stop bit and no parity checking.

16.4.3 Simple no-handshaking communications

In this form of communication it is assumed that the receiver can read the received data from the receiver buffer before another character is received. Data is sent from a TD pin connection of the transmitter and is received in the RD pin connection at the receiver. When a DTE (such as a computer) connects to another DTE, then the transmit line (TD) on one is connected to the receive (RD) of the other and vice versa. Figure 16.7 shows the connections between the nodes.

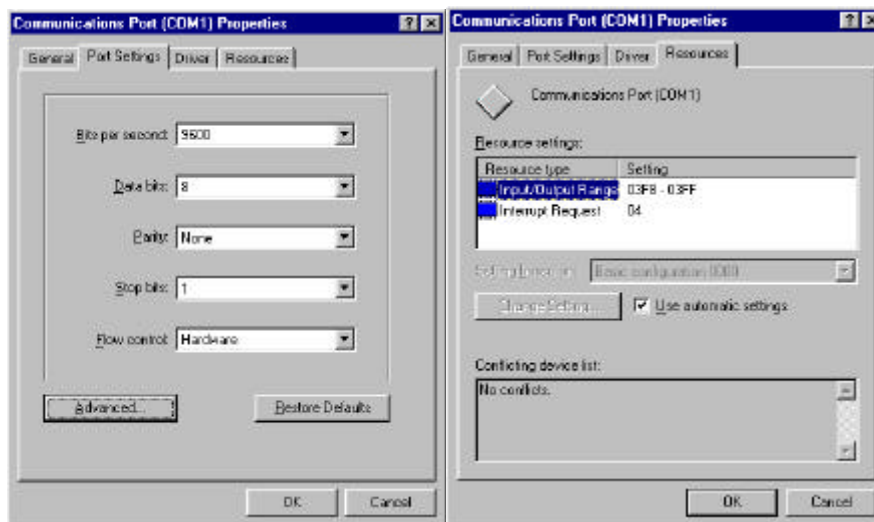


Figure 16.6 Changing port setting and parameters

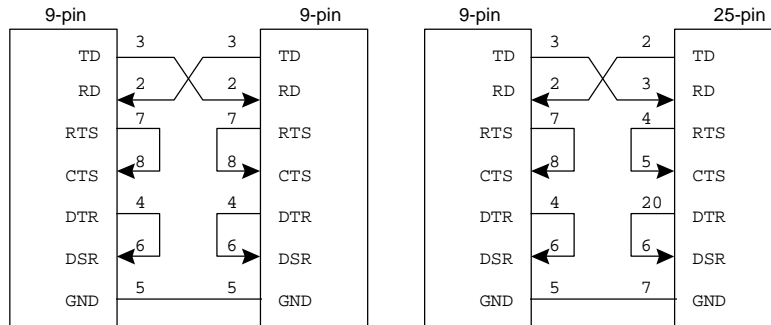


Figure 16.7 RS-232 connections with no hardware handshaking

16.4.4 Software handshaking

There are two ASCII characters that start and stop communications. These are X-ON (^S , Cntrl-S or ASCII 11) and X-OFF (^Q , Cntrl-Q or ASCII 13). When the transmitter receives an X-OFF character it ceases communications until an X-ON character is sent. This type of handshaking is normally used when the transmitter and receiver can process data relatively quickly. Normally, the receiver will also have a large buffer for the incoming characters. When this buffer is full it transmits an X-OFF. After it has read from the buffer the X-ON is transmitted, see Figure 16.8.

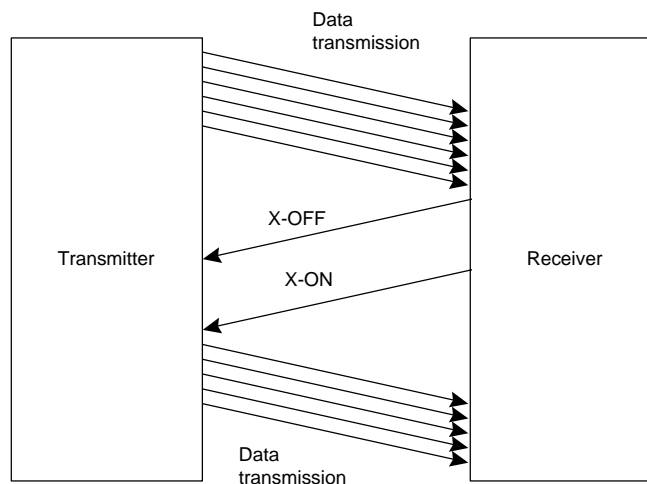


Figure 16.8 Software handshaking using X-ON and X-OFF

16.4.5 Hardware handshaking

Hardware handshaking stops characters in the receiver buffer from being overwritten. The control lines used are all active HIGH. When a node wishes

to transmit data it asserts the RTS line active (that is, HIGH). It then monitors the CTS line until it goes active (that is, HIGH). If the CTS line at the transmitter stays inactive then the receiver is busy and cannot receive data, at the present. When the receiver reads from its buffer the RTS line will automatically go active indicating to the transmitter that it is now ready to receive a character.

Receiving data is similar to the transmission of data, but the lines DSR and DTR are used instead of RTS and CTS. When the DCE wishes to transmit to the DTE the DSR input to the receiver will become active. If the receiver cannot receive the character it will set the DTR line inactive. When it is clear to receive it sets the DTR line active and the remote node then transmits the character. The DTR line will be set inactive until the character has been processed.

16.4.6 Two-way communications with handshaking

For full handshaking of the data between two nodes the RTS and CTS lines are crossed over (as are the DTR and DSR lines). This allows for full remote node feedback (see Figure 16.9).

16.5 Programming RS-232

Normally, serial transmission is achieved via the RS-232 standard. Although 25 lines are defined usually only a few are used. Data is sent along the TD line and received by the RD line with a common ground return. The other lines used for handshaking are RTS (Ready to Send) which is an output signal to indicate that data is ready to be transmitted and CTS (Clear to Send), which is an input indicating that the remote equipment is ready to receive data.

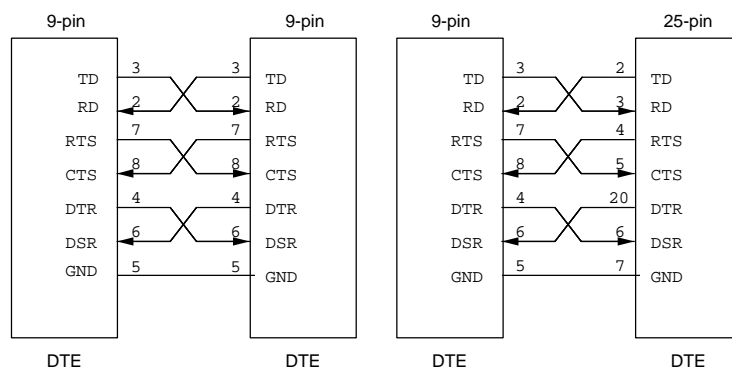


Figure 16.9 RS-232 communications with handshaking

The 8250 IC is commonly used in serial communications. It can either be mounted onto the motherboard of the PC or fitted to an I/O card. This section discusses how it is programmed.

16.5.1 Programming the serial device

The main registers used in RS-232 communications are the Line Control Register (LCR), the Line Status Register (LSR) and the Transmit and Receive buffers (see Figure 16.10). The Transmit and Receive buffers share the same addresses.

The base address of the primary port (COM1 :) is normally set at 3F8h and the secondary port (COM2 :) at 2F8h. A standard PC can support up to four COM ports.

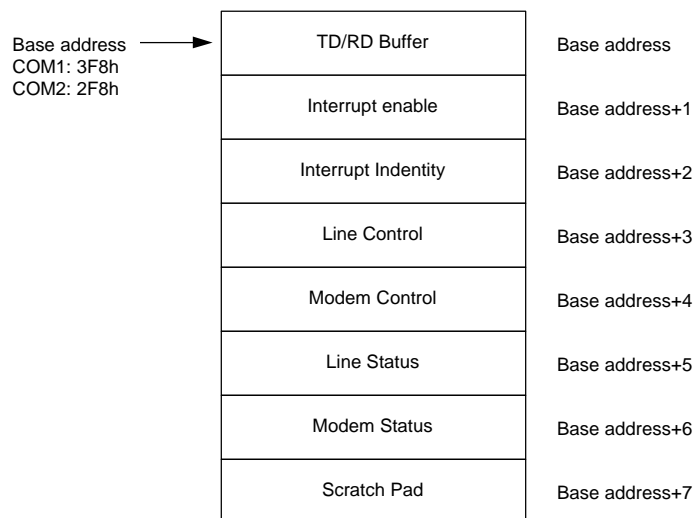


Figure 16.10 Serial communication registers

16.5.2 Line Status Register (LSR)

The LSR determines the status of the transmitter and receiver buffers. It can only be read from, and all the bits are automatically set by hardware. The bit definitions are given in Figure 16.11. When an error occurs in the transmission of a character one (or several) of the error bits is (are) set to a '1'.

One danger when transmitting data is that a new character can be written to the transmitter buffer before the previous character has been sent. This overwrites the contents of the character being transmitted. To avoid this the status bit S_6 is tested to determine if there is still a character in the buffer. If there is then it is set to a '1', else the transmitter buffer is empty.

To send a character:

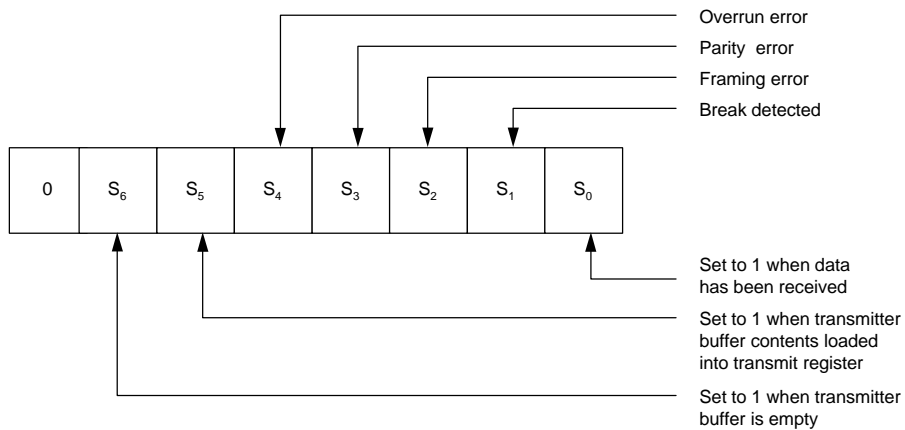


Figure 16.11 Line Status Register

Test Bit 6 until set;
Send character;

A typical Pascal routine is:

```
repeat
  status := port[LSR] and $40;
until (status=$40);
```

When receiving data the S₀ bit is tested to determine if there is a bit in the receiver buffer. To receive a character:

Test Bit 0 until set;
Read character;

A typical Pascal routine is:

```
repeat
  status := port[LSR] and $01;
until (status=$01);
```

Figure 16.12 shows how the LSR is tested for the transmission and reception of characters.

16.5.3 Line Control Register (LCR)

The LCR sets up the communications parameters. These include the number of bits per character, the parity and the number of stop bits. It can be written to or read from and has a similar function to that of the control registers used in the PPI and PTC. The bit definitions are given in Figure 16.13.

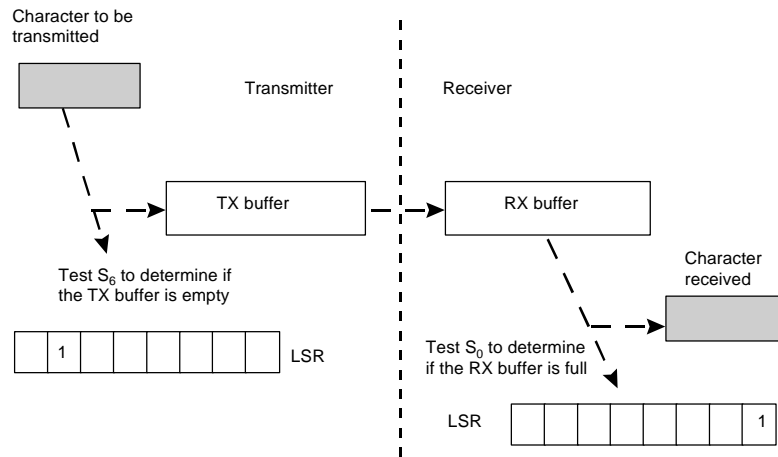


Figure 16.12 Testing of the LSR for the transmission and reception of characters

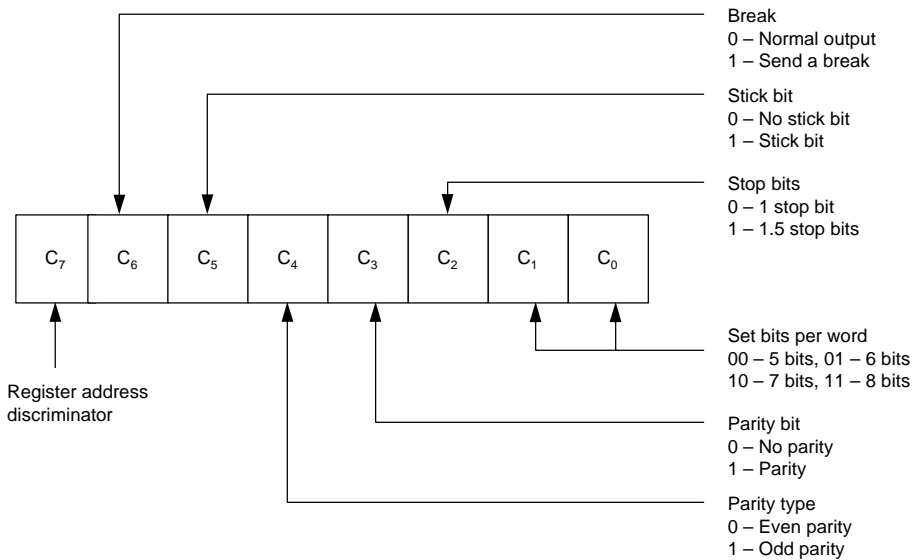


Figure 16.13 Line Control Register

The msb, C_7 , must be set to a '0' in order to access the transmitter and receiver buffers, else if it is set to a '1' the baud rate divider is set up. The baud rate is set by loading an appropriate 16-bit divisor into the addresses of transmitter/receiver buffer address and the next address. The value loaded depends on the crystal frequency connected to the IC. Table 16.3 shows divisors for a crystal frequency is 1.8432 MHz. In general the divisor, N , is related to the baud rate by:

$$\text{Baud rate} = \frac{\text{Clock frequency}}{16 \times N}$$

For example, for 1.8432 MHz and 9600 baud $N = 1.8432 \times 10^6 / (9600 \times 16) = 12$ (000Ch).

Table 16.3 baud rate divisors

<i>baud rate</i>	<i>Divisor (value loaded into Tx/Rx buffer)</i>
110	0417h
300	0180h
600	00C0h
1200	0060h
1800	0040h
2400	0030h
4800	0018h
9600	000Ch
19200	0006h

16.5.4 Register addresses

The addresses of the main registers are given in Table 16.4. To load the baud rate divisor, first the LCR bit 7 is set to a '1', then the LSB is loaded into divisor LSB and the MSB into the divisor MSB register. Finally, bit 7 is set back to a '0'. For example, for 9600 baud, COM1 and 1.8432 MHz clock then 0Ch is loaded in 3F8h and 00h into 3F9h.

When bit 7 is set at a '0' then a read from base address reads from the RD buffer and a write operation writes to the TD buffer. An example of this is shown in Figure 16.14.

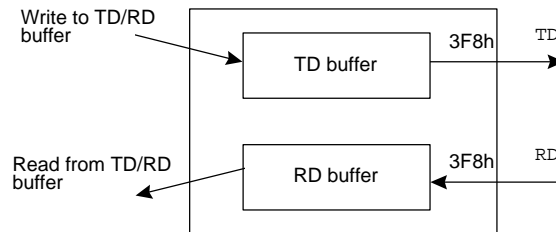


Figure 16.14 Read and write from TD/RD buffer

Table 16.4 Serial communications addresses

Primary	Secondary	Register	Bit 7 of LCR
3F8h	2F8h	TD buffer	'0'
3F8h	2F8h	RD buffer	'0'
3F8h	2F8h	Divisor LSB	'1'
3F9h	2F9h	Divisor MSB	'1'
3FBh	2FBh	Line Control Register	
3FDh	2FDh	Line Status Register	

16.6 RS-232 programs

Figure 16.15 shows the main RS-232 connection for 9- and 25-pin connections without hardware handshaking. The loopback connections are used to test the RS-232 hardware and the software, while the null modem connections are used to transmit characters between two computers. Program 16.1 uses a loop back on the TD/RD lines so that a character sent by the computer will automatically be received into the receiver buffer. This set up is useful in testing the transmit and receive routines. The character to be sent is entered via the keyboard. A *CNTRL-D* (^D) keystroke exits the program.

Program 16.2 can be used as a sender program (send.c) and Program 16.3 can be used as a receiver program (receive.c). With these program the null modem connections shown in Figure 16.15 are used.

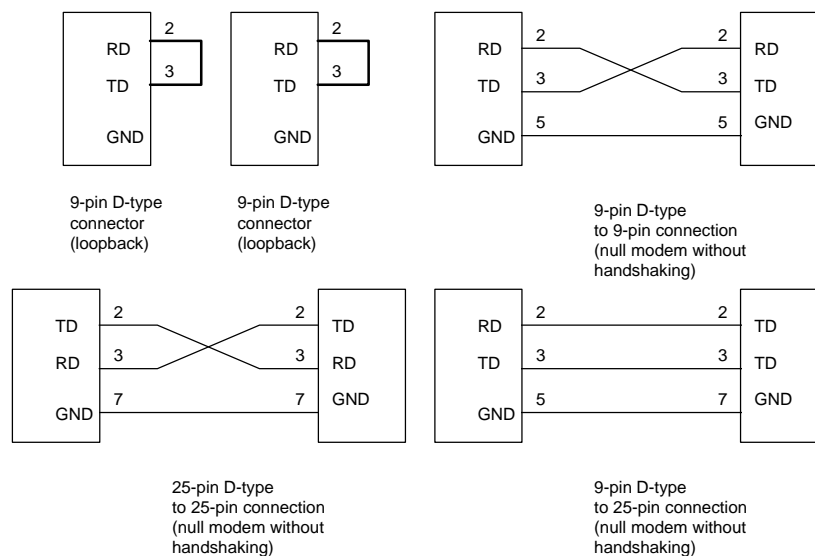


Figure 16.15 System connections

 **Program 16.1**

```
program prog16_1(input,output);
(* This program transmits a character from COM1: and receives *)
(* it via this port. The TD is connected to RD. *)
uses crt;
const TXDATA = $3F8; LSR = $3FD;
      LCR = $3FB; CNTRLD = #4;
var inchar, outchar:char;

procedure setup_serial;
begin
  port[LCR] := $80; (* set up bit 7 to a 1 *)
  port[TXDATA] := $0C;
  port[TXDATA+1] := $00;
  (* load TxRegister with 12, crystal frequency is 1.8432 MHz *)
  port[LCR] := $0A
  (* Bit pattern loaded is 00001010b, from msb to lsb these are: *)
  (* Access TD/RD buffer, normal output, no stick bit *)
  (* even parity, parity on, 1 stop bit, 7 data bits *)
end;

procedure send_character(ch:char);
var status:byte;
begin
  repeat
    status := port[LSR] and $40;
  until (status=$40);
  (*repeat until bit Tx buffer is empty *)
  port[TXDATA] := ord(ch); (*send ASCII code *)
end;

function get_character:char;
var status,inbyte:byte;
begin
  repeat
    status := port[LSR] and $01;
  until (status=$01);
  inbyte := port[TXDATA];
  get_character:= chr(inbyte);
end;

begin
  setup_serial;

  repeat
    outchar:=readkey;
    send_character(outchar);
    inchar:=get_character;
    writeln('Character received was ',inchar);
  until (outchar=CNTRLD);

end.
```

 **Program 16.2**

```
program prog16_2(input,output);
(* sender.pas *)

uses crt;

const
    TXDATA = $3F8;    LSR = $3FD;    LCR = $3FB;

var    outchar:char;

procedure setup_serial;
begin
    port[LCR] := port[LCR] or $80;
    (* set up bit 7 to a 1 *)
    port[TXDATA] := $0C;
    port[TXDATA+1] := $00;
    (* load TxRegister with 12 *)
    (* crystal frequency is 1.8432MHz *)
    port[LCR] := port[LCR] and $7F
    (* set up bit 7 to a 0 *)
    (* bit 7 must be a 0 to access TxBuff or RxBuff *)
    (* serial port has been set up *)
end;

procedure send_character(ch:char);
var    status:byte;
begin
    repeat
        status := port[LSR] and $40;
    until (status=$40);
    (*repeat until bit Tx buffer is empty *)

    port[TXDATA] := ord(ch); (*send ASCII code *)
end;

begin
    setup_serial;
    repeat
        outchar:=readkey;
        send_character(outchar);
    until (outchar=#4);
end.
```

 **Program 16.3**

```
program prog16_3(input,output);
(* receive.pas *)

uses crt;

const
    TXDATA = $3F8;    LSR = $3FD;    LCR = $3FB;
```



```

var inchar:char;

procedure setup_serial;
begin
    port[LCR] := port[LCR] or $80;
        (* set up bit 7 to a 1 *)
    port[TXDATA] := $0C;
    port[TXDATA+1] := $00;
        (* load TxRegister with 12 *)
        (* crystal frequency is 1.8432MHz *)
    port[LCR] := port[LCR] and $7F
        (* set up bit 7 to a 0 *)
        (* bit 7 must be a 0 to access TxBuff or RxBuff *)
        (* serial port has been set up *)
end;

function get_character:char;
var status,inbyte:byte;
begin
    repeat
        status := port[LSR] and $01;
    until (status=$01);

    inbyte := port[TXDATA];

    get_character:= chr(inbyte);
end;

begin
    setup_serial;

    repeat
        inchar:=get_character;
        write(inchar);
    until (inchar=#4);
end.

```

16.7 Exercises


- 16.7.1** Write a program that continuously sends the character ‘A’ to the serial line. Observe the output on an oscilloscope and identify the bit pattern and the baud rate.
- 16.7.2** Write a program that continuously sends the characters from ‘A’ to ‘Z’ to the serial line. Observe the output on an oscilloscope.
- 16.7.3** Complete Table 16.5 to give the actual time to send 1000 characters for the given baud rates. Compare these values with estimated values.

Table 16.5 baud rate divisors

<i>baud rate</i>	<i>Time to send 1000 characters (sec)</i>
110	
300	
600	
1200	
2400	
4800	
9600	
19200	

Note that approximately 10 bits are used for each character thus 960 characters/sec will be transmitted at 9600 baud.

- 16.7.4** Modify Program 16.1 so that the program prompts the user for the baud rate when the program is started. A sample run is shown in Test run 16.1.

```
 Test run 16.1
Enter baud rate required:
1 110
2 150
3 300
4 600
5 1200
6 2400
7 4800
8 9600
>> 8
RS232 transmission set to 9600 baud
```

- 16.7.5** One problem with Programs 16.2 and 16.3 is that when the return key is pressed only one character is sent. The received character will be a carriage return which returns the cursor back to the start of a line and not to the next line. Modify the receiver program so that a line feed will be generated automatically when a carriage return is received. Note a carriage return is an ASCII 13 and line feed is a 10.

- 16.7.6** Modify the `get_character()` routine so that it returns an error flag if it detects an error or if there is a time-out. Table 16.6 lists the error flags and the returned error value. If a character is not received within 10 seconds an error message should be displayed.

Test the routine by connecting two PCs together and set the transmitter with differing RS-232 parameters.

Table 16.6 Error returns from `get_character()`

<i>Error condition</i>	<i>Error flag return</i>	<i>Notes</i>
Parity error	-1	
Overrun error	-2	
Framing error	-3	
Break detected	-4	
Time-out	-5	<code>get_character()</code> should time-out if no characters are received within 10 seconds.