An endeavor has been made in this article to bring forth a simple, easy and novel way of implementing PC-to-PC communication via RS-232 serial port using C language. Implementing the asynchronous serial communication this way does not require the reader to be familiar with serial port registers and their programming, and there is no need for constructing user-defined functions for setting the baud rate and format of data, parity and stop bits. Moreover, the speed of data transfer is also greater and the function used for serial programming along with its arguments makes its purpose self-explanatory.

A PC can accommodate, at most, four serial ports but usually a PC has two RS-232 serial ports, COM1 and COM2. Any one of the serial ports can be used in each PC for linking them together. A serial port at the back of a PC is in the form of 9-pin (or sometimes 25-pin) D-type male connector. Table 1 shows pin configurations of 9-pin D-type male connector, which is depicted in Figure 1.

**Serial communication**

Data transfer within a system is generally in parallel. All the bits of the data word are transferred in parallel at the same instant. In some cases, particularly in transferring data over long distances, it is preferred to transfer the data in serial form. The data word from a transmitting system is converted to stream of bits by parallel to serial conversion, and one bit at a time is transferred on a single line to a receiving system. At the receiving end, the word is reconstructed by serial to parallel conversion. The speed of data transfer in serial communication is specified by baud.

The baud unit is named after Jean Maurice Emile Baudot, who was an officer in the French Telegraph Service. He is credited with devising the first uniform-length 5-bit code for characters of the alphabet in the late 19th century. What baud really refers to is modulation rate or the number of times per second that a line changes state. This is not always the same as bits per second (bps). If we connect two serial devices together using direct cables then baud and bps are, in fact, the same. But when modems are in question, this isn’t the case.

**Asynchronous serial communication**

Asynchronous data transfer is used for low speed communication, typically at standard rates such as
2400, 4800, 9600, 19200 baud etc. The asynchronous communication format does not use any synchronising clock or timing signal.

**Framing** – Transmission of a character starts with a start bit (logic 0), followed by the character bits (LSB first), a parity bit and ends with one or two stop bits (logic 1). This is referred to as one frame. Process of adding the start, parity and stop bits with character bits is referred to as framing. When no character is sent, the transmitter outputs logic high. The line remains in logic 1 (idle state) till the transmission of next character begins with another start bit.

**Error detection** – Error in asynchronous communication is detected in three ways, by checking parity error, framing error and overrun error. The parity error informs that the received data has wrong parity, indicating that the noise was encountered during reception. The framing error informs that the received data does not have the start and stop bits at their proper places. The overrun error indicates that a new data has been received before the previous data could be taken away.

**RS/EIA-232**

Short for Recommended Standard-232, a standard interface approved by EIA (Electronic Industries Association) for connecting serial devices, specifies signal voltages, signal timing, signal function, a protocol for information exchange and mechanical connectors. To ensure reliable communication and to enable the interconnection of equipment produced by different manufacturers, the interfacing standard RS-232 was set by EIA in 1960. Since then it has gone through a number of modifications, including a change in its name. RS-232A, RS-232B, RS-232C, EIA-232D and EIA-232E are the subsequent versions of this standard. The standard has been referred to as RS-232 (instead of EIA-232) throughout this article due to its popularity.

The RS-232 standard supports two types of connectors – a 25-pin D-type connector (DB-25) and a 9-pin D-type connector (DB-9). The type of serial communications used by PCs requires only nine pins, so either type of connector will work equally well. Since modern PCs employ only 9-pin D-type connectors, only this configuration has been discussed in this article, including connections and programming.

In RS-232 parlance, the device that connects to the interface is called Data Communications Equipment (DCE) and the device to which it connects is called Data Terminal Equipment (DTE). This standard was mainly designed to connect DTE that is sending and receiving serial data (such as a computer) and DCE that is used to send data over long distances (such as a modem).

**To distinguish between DTE and DCE:**

- Measure the DC voltages between (DB-9) pins 3 and 5 and between pins 2 and 5. Be sure that the black lead is connected to pin 5 (GND) and the red lead to whichever pin you are measuring.
  - If the voltage on pin 3 (TxD) is more negative than -3V, then it is a DTE, otherwise it should be near zero volts.
  - If the voltage on pin 2 (RxD) is more negative than -3V, then it is a DCE.
  - If both pins 3 and 2 have a voltage of at least 3V,
then either you are measuring incorrectly, or your device is not a standard RS-232 device. Call technical support.

- In general, a DTE provides a voltage on TxD, RTS & DTR, whereas a DCE provides voltage on RxD, CTS, DSR & DCD.

### Programming an RS-232 serial port using C

**Library File Inclusion**

```c
#include<bios.h>
```

**Function Declaration/Syntax**

```c
Unsigned _bios_serialcom(int cmd, int port, charabyte);
```

**Brief Description**

The function `_bios_serialcom()` uses BIOS interrupt 0x14 to perform various RS-232 communications over the I/O port given in the port. The function arguments along with their significance are given in Table 2 below.

### Table 2: Various function arguments and their significance

<table>
<thead>
<tr>
<th>Argument</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>abyte</td>
<td>OR combination of bits that specifies COM port settings</td>
</tr>
<tr>
<td>cmd</td>
<td>Specifies the I/O operation to perform</td>
</tr>
<tr>
<td>port</td>
<td>Identifies the I/O port</td>
</tr>
</tbody>
</table>

**Function Argument Specifications**

- **Port argument** – The serial port that is selected for RS-232 communication is specified in the port argument as given in Table 3.

### Table 3: Port argument specification

<table>
<thead>
<tr>
<th>Value of 'port' argument</th>
<th>Port selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>COM1</td>
</tr>
<tr>
<td>1</td>
<td>COM2</td>
</tr>
<tr>
<td>2</td>
<td>COM3</td>
</tr>
<tr>
<td>3</td>
<td>COM4</td>
</tr>
</tbody>
</table>

- **Cmd Argument** – The I/O operation to be performed is specified by means of 'cmd' argument as given in Table 4. When the value of 'cmd' argument is set to either _COM_RECEIVE or _COM_STATUS, the value in 'abyte' argument is ignored.

- **Abyte argument** – When the value of 'cmd' argument is set to _COM_INIT, the COM port settings are specified by the 'abyte' argument. The 'abyte' argument is an OR combination of the following values (one from each group in Table 5).

### Table 4: Cmd argument specifications

<table>
<thead>
<tr>
<th>Value of 'cmd' argument</th>
<th>Function performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>_COM_INIT</td>
<td>Sets the communication parameters to the value in 'abyte' argument</td>
</tr>
<tr>
<td>_COM_SEND</td>
<td>Sends the character in 'abyte' argument out over the communications line</td>
</tr>
<tr>
<td>_COM_RECEIVE</td>
<td>Receives a character from the communications line</td>
</tr>
<tr>
<td>_COM_STATUS</td>
<td>Returns current status of the communications port</td>
</tr>
</tbody>
</table>

### Table 5: Abyte argument specifications

<table>
<thead>
<tr>
<th>Value of 'abyte' argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_COM_CHR7</td>
<td>7 data bits</td>
</tr>
<tr>
<td>_COM_CHR8</td>
<td>8 data bits</td>
</tr>
<tr>
<td>_COM_STOP1</td>
<td>1 stop bit</td>
</tr>
<tr>
<td>_COM_STOP2</td>
<td>2 stop bits</td>
</tr>
<tr>
<td>_COM_NOPARITY</td>
<td>No parity</td>
</tr>
<tr>
<td>_COM_ODDPARITY</td>
<td>Odd parity</td>
</tr>
<tr>
<td>_COM_EVENPARITY</td>
<td>Even parity</td>
</tr>
<tr>
<td>_COM_110</td>
<td>110 baud</td>
</tr>
<tr>
<td>_COM_150</td>
<td>150 baud</td>
</tr>
<tr>
<td>_COM_300</td>
<td>300 baud</td>
</tr>
<tr>
<td>_COM_600</td>
<td>600 baud</td>
</tr>
<tr>
<td>_COM_1200</td>
<td>1200 baud</td>
</tr>
<tr>
<td>_COM_2400</td>
<td>2400 baud</td>
</tr>
<tr>
<td>_COM_4800</td>
<td>4800 baud</td>
</tr>
<tr>
<td>_COM_9600</td>
<td>9600 baud</td>
</tr>
</tbody>
</table>

### Return value

For all values of ‘cmd’ argument, the function `_bios_serialcom()` returns a 16-bit unsigned integer.

- The upper 8 bits of the return value are status bits.
- The lower 8 bits vary depending upon the value of ‘cmd’ argument specified as given in Table 6.

The following format shows the details of all the return bits:

#### Lower Byte of Return Value

- D0 = Received line signal detect
- D1 = Ring indicator
- D2 = Data set ready
- D3 = Clear to send
- D4 = Change in receive line signal detector
- D5 = Trailing edge ring indicator
- D6 = Change in data set ready
- D7 = Change in clear to send

#### Upper Byte of Return Value

- D8 = Data ready
Table 6: Significance of lower byte of return value

<table>
<thead>
<tr>
<th>Value of “cmd” argument</th>
<th>Lower 8 bits of return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_COM_INIT or _COM_STATUS</td>
<td>Lower bits are defined as shown in format given below.</td>
</tr>
<tr>
<td>_COM_SEND</td>
<td>...</td>
</tr>
<tr>
<td>_COM_RECEIVE</td>
<td>Byte read is in the lower bits of the return value (if there is no error i.e. no upper error bits are set to 1)</td>
</tr>
</tbody>
</table>

D9 = Overrun error  
D10 = Parity error  
D11 = Framing error  
D12 = Break detect  
D13 = Transmit holding register empty  
D14 = Transmit shift register empty  
D15 = Time out (set to 1 if any byte value could not be sent)

3. Source code

/* PC-to-PC communication – by VARUN JINDAL */
/* B.E. (E&C) - final year, Panjab University, Chandigarh */
#include<stdio.h>
#include<conio.h>
#include<bios.h>
#define SETTINGS (_COM_9600 | _COM_CHR8 | _COM_NOPARITY | _COM_STOP1)

// baud rate = 9600, 8 data bits, no parity bit, 1 stop bit */

void main(void)
{
    unsigned in,out,status;
    int port;

clearscr();

    printf("Select Port(Enter '0' for COM1 and '1' for COM2): ");
    scanf("%d", &port);

    printf("Press ESC to exit

    textcolor(YELLOW);
    cprintf("Data Received:");

    _bios_serialcom(_COM_INIT,port,SETTINGS);

    for(;;)
    {
        status=_bios_serialcom(_COM_STATUS,port,0);
        if (status & 512)
            printf("Overrun Error\n");
        if (status & 1024)
            printf("Parity Error\n");
        if (status & 2048)
            printf("Framing Error\n");
        if(status & (512 | 1024 | 2048)) /* if any error */
            break;

        if(status & 256) /* if data ready */
            {
                if((out=_bios_serialcom(_COM_RECEIVE,port,0) & 255)!=0)
                    putch(out);
            }
        if(kbhit()) /* if a keystroke is currently available */
            {
                in=getch(); /* get a character without echoing onto the screen */
                if(in==27) /* if ESC */
                    break;
                _bios_serialcom(_COM_SEND,port,in);
            }
    }
}

Figure 3: Null modem cable configuration

3. Description steps

1. Objective – To implement PC-to-PC communication by inputting the data through one port and receiving the same through another and vice-versa (Turbo C++ version 3.0).
2. Connection diagram – The connection diagram (Figure 3) shows the wiring of the null-modem that is intended for RS232 asynchronous communications (most PC-based systems). This configuration is called so because each PC terminal detects as if some modem is connected to it rather than the other PC. The two PC terminals are connected through TxD, RxD and GND pins. The Data Terminal Ready (DTR, pin 4) is looped back to Data Set Ready (DSR, pin 6) and Data Carrier Detect (DCD, pin 1) on both PCs. When DTR is asserted active, then the DSR and DCD immediately become active. At this moment, the computer thinks the virtual modem to which it is connected is ready and has detected the carrier of the other modem. The lines Request to Send (RTS, pin 7) and Clear to Send (CTS, pin 8) have been linked together. When the computer wishes to send data, it asserts RTS high, which in turn asserts CTS high, meaning thereby that the virtual modem has the room for storing the data and the computer can send it.
5. Data transfer procedure
- Connect the two PCs together using the 3-wire link.
- Run the program given in the source code on both PCs.
- Before sending the data, set both the sending and receiving PC terminals to the same baud rate, and same format of data bits, parity bits and stop bits using the macro (pre-processor directive) ‘SETTINGS’ in the source code.

6. Limitations
- When the source code is compiled and run on both PCs, the characters typed in one computer should appear on the other computer’s screen and vice-versa.

7. Applications
- On the guidelines presented in this article, useful functions such as file transfer, chatting etc can be implemented. By using hardware circuitry employing infrared/laser diodes, even wireless PC-to-PC communication can be easily implemented.

Designers’ note:
The executable code is available on request.

**Figure 4** Loopback connector

---

**Serial Communications**

**Quality second-user test & measurement equipment**
Tel: 02476 650 702 Fax: 02476 650 773
Web: www.telnet.uk.com  Email: sales@telnet.uk.com
All equipment is used – with 30 days guarantee and 90 days in some cases. Add carriage and VAT to all goods.

1 Stoney Court, Hotchkin Way, Binley Industrial Estate Coventry CV3 2RL ENGLAND

---

4. Testing – It is usually difficult to work on both PCs when a programmer wishes to check his/her source code for PC-to-PC communication. The best possible way to overcome this problem is to use a loopback connector (shown in Figure 4), which enables the programmer to write source-code for programming serial port with single PC.

A loopback connector usually consists of a connector without a cable and includes internal wiring to re-route signals back to the sender. When the computer receives data, it does not know whether the signals it receives come from a remote DCE device or from a loopback connector. Using loopback connector, proper operation can be checked.

5. Data transfer procedure
- Connect the two PCs together using the 3-wire link.
- Run the program given in the source code on both PCs.
- Before sending the data, set both the sending and receiving PC terminals to the same baud rate, and same format of data bits, parity bits and stop bits using the macro (pre-processor directive) ‘SETTINGS’ in the source code.

When the source code is compiled and run on both PCs, the characters typed in one computer should appear on the other computer’s screen and vice-versa.

6. Limitations
- Here, we are using RS-232 serial asynchronous communication, so the communication speed is less than that of parallel data transfer, where 8-bit data is sent at a time rather than bit by bit.

7. Applications
- On the guidelines presented in this article, useful functions such as file transfer, chatting etc can be implemented. By using hardware circuitry employing infrared/laser diodes, even wireless PC-to-PC communication can be easily implemented.

Designers’ note:
The executable code is available on request.

---

**Figure 4** Loopback connector

---

**Serial Communications**

**Quality second-user test & measurement equipment**
Tel: 02476 650 702 Fax: 02476 650 773
Web: www.telnet.uk.com  Email: sales@telnet.uk.com
All equipment is used – with 30 days guarantee and 90 days in some cases. Add carriage and VAT to all goods.

1 Stoney Court, Hotchkin Way, Binley Industrial Estate Coventry CV3 2RL ENGLAND

---

4. Testing – It is usually difficult to work on both PCs when a programmer wishes to check his/her source code for PC-to-PC communication. The best possible way to overcome this problem is to use a loopback connector (shown in Figure 4), which enables the programmer to write source-code for programming serial port with single PC.

A loopback connector usually consists of a connector without a cable and includes internal wiring to re-route signals back to the sender. When the computer receives data, it does not know whether the signals it receives come from a remote DCE device or from a loopback connector. Using loopback connector, proper operation can be checked.

5. Data transfer procedure
- Connect the two PCs together using the 3-wire link.
- Run the program given in the source code on both PCs.
- Before sending the data, set both the sending and receiving PC terminals to the same baud rate, and same format of data bits, parity bits and stop bits using the macro (pre-processor directive) ‘SETTINGS’ in the source code.

When the source code is compiled and run on both PCs, the characters typed in one computer should appear on the other computer’s screen and vice-versa.

6. Limitations
- Here, we are using RS-232 serial asynchronous communication, so the communication speed is less than that of parallel data transfer, where 8-bit data is sent at a time rather than bit by bit.

7. Applications
- On the guidelines presented in this article, useful functions such as file transfer, chatting etc can be implemented. By using hardware circuitry employing infrared/laser diodes, even wireless PC-to-PC communication can be easily implemented.

Designers’ note:
The executable code is available on request.