Introduction

An endeavour has been made in this article to bring forth simple, easy and novel way of implementing the Full-Duplex PC to PC communication (Chatting) via RS-232 serial port using 'C' language. Implementing the serial communication this way does not require the reader to be familiar with serial port registers & their programming and there is no need for constructing user-defined functions for setting the baud rate and format of data, parity & 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, viz, 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 shown in Fig. 1.

<table>
<thead>
<tr>
<th>Pin number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DCD (data carrier detect)</td>
</tr>
<tr>
<td>2</td>
<td>RxD (receive data)</td>
</tr>
<tr>
<td>3</td>
<td>TxD (transmit data)</td>
</tr>
<tr>
<td>4</td>
<td>DTR (data terminal)</td>
</tr>
<tr>
<td>5</td>
<td>GND (signal ground)</td>
</tr>
<tr>
<td>6</td>
<td>DSR (data set ready)</td>
</tr>
<tr>
<td>7</td>
<td>RTS (request to send)</td>
</tr>
<tr>
<td>8</td>
<td>CTS (clear to send)</td>
</tr>
<tr>
<td>9</td>
<td>RI (ring indicator)</td>
</tr>
</tbody>
</table>

Table 1. Pin Functions of 9-Pin Connector of RS-232 Serial Port

Fig. 1: 9-Pin D-type Connector of RS-232 Serial Port

Programming RS-232 Serial Port using 'C'

Library File Inclusion
#include<bios.h>

Function
 bios_serialcom();

Function Declaration/Syntax
Unsigned bios_serialcom(int cmd, int port, char abyte);

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.

<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>

Table 2. Various Function Arguments and their Significance

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.

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 set to 0 or 1, respectively.

<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>

Table 3. Port Argument Specification
argument is ignored.

A byte 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.).

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

**Lower Byte of Return Value**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Received line signal detect</td>
</tr>
<tr>
<td>1</td>
<td>Ring indicator</td>
</tr>
<tr>
<td>2</td>
<td>Data set ready</td>
</tr>
<tr>
<td>3</td>
<td>Clear to send</td>
</tr>
<tr>
<td>4</td>
<td>Change in receive line signal detector</td>
</tr>
<tr>
<td>5</td>
<td>Trailing edge ring indicator</td>
</tr>
<tr>
<td>6</td>
<td>Change in data set ready</td>
</tr>
<tr>
<td>7</td>
<td>Change in clear to send</td>
</tr>
<tr>
<td>8</td>
<td>Overrun error</td>
</tr>
<tr>
<td>9</td>
<td>Parity error</td>
</tr>
<tr>
<td>10</td>
<td>Framing error</td>
</tr>
<tr>
<td>11</td>
<td>Break detect</td>
</tr>
<tr>
<td>12</td>
<td>Transmit holding register empty</td>
</tr>
<tr>
<td>13</td>
<td>Transmit shift register empty</td>
</tr>
<tr>
<td>14</td>
<td>Time out (set to 1 if abyte value could not be sent)</td>
</tr>
<tr>
<td>15</td>
<td>Time out (set to 1 if abyte value could not be sent)</td>
</tr>
</tbody>
</table>

Upper Byte of Return Value

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Request to Send (RTS, pin 6)</td>
</tr>
<tr>
<td>1</td>
<td>Pin 7) and Data Carrier Detect (DCD, pin 1)</td>
</tr>
<tr>
<td>2</td>
<td>Data Terminal Ready (DTR, pin 4)</td>
</tr>
<tr>
<td>3</td>
<td>Data Set Ready (DSR, pin 2)</td>
</tr>
<tr>
<td>4</td>
<td>Clear to Send (CTS, pin 8)</td>
</tr>
<tr>
<td>5</td>
<td>Break Detect</td>
</tr>
<tr>
<td>6</td>
<td>Break Detect</td>
</tr>
<tr>
<td>7</td>
<td>Break Detect</td>
</tr>
<tr>
<td>8</td>
<td>Break Detect</td>
</tr>
<tr>
<td>9</td>
<td>Break Detect</td>
</tr>
<tr>
<td>10</td>
<td>Break Detect</td>
</tr>
<tr>
<td>11</td>
<td>Break Detect</td>
</tr>
<tr>
<td>12</td>
<td>Break Detect</td>
</tr>
<tr>
<td>13</td>
<td>Break Detect</td>
</tr>
<tr>
<td>14</td>
<td>Break Detect</td>
</tr>
<tr>
<td>15</td>
<td>Break Detect</td>
</tr>
</tbody>
</table>

Description Steps

**1. Objective** - To establish Full-Duplex PC to PC Communication by implementing Chatting using 'C' (Turbo C++ version 3.0).

**2. Connection Diagram** - The connection diagram shows the wiring of the null-modem that is intended for.

4. Data Transfer Procedure

- Connect the two PCs together using the 3-wire link.
- Run the program given in the source code on both the 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 & stop bits using the macro(pre-processor directive) 'SETTINGS' in the source code.
- When the source code is compiled and run on both the PCs, the character strings (whose termination is indicated by pressing ENTER key) typed in the text entry window of one PC should appear in the communication display window of both the PCs.

5. Sample Settings

- Baud rate = 9600 bits per second
- 8 data bits
- Even parity bit
- 1 stop bit
#include<stdio.h>
#include<conio.h>
#include<bios.h>

#define SETTINGS (_COM_9600 | _COM_CHR8 | _COM_EVENPARITY | _COM_STOP1)
/* baud rate = 9600, 8 data bits, even parity bit, 1 stop bit */

void display(char *, int COLOR);

void main(void) {
  unsigned in, out, status, count_in=0, count_out=0;
  int port, i;
  char msg_in[75], msg_out[75];
  textbackground(BLACK);
  clrscr();

gotoxy(3, 1);
textcolor(GREEN);
cprintf("Full-Duplex PC to PC Communication via RS-232 Serial Port using 'C':Chatting");
gotoxy(7, 3);
textcolor(LIGHTCYAN);
cprintf("By VARUN JINDAL, B.E.(E&C)-7th semester,
Panjab University, Chandigarh");
gotoxy(15, 5);
textcolor(LIGHTGRAY);
cprintf("Select Port(Enter '0' for COM1 and '1' for
COM2) :");
scanf("%d", &port);

textcolor(LIGHTGRAY + BLINK);
cprintf("\\n Press ESC to exit Hit ENTER to send outgoing message");
gotoxy(10, 9);
textcolor(MAGENTA + BLINK);
putch(1);  
cprintf("-Outgoing message");
gotoxy(53, 9);
textcolor(BLUE + BLINK);
putch(1);  
cprintf("-Incoming message");

window(10, 11, 70, 23);
textbackground(YELLOW);
clrscr();

/* Communication Display Window */
window(1, 25, 80, 25);
textbackground(BLUE);
textcolor(YELLOW);
clrscr();

/* Text Entry Window */
window(1, 1, 80, 25);
gotoxy(2, 25);
putch(2);

_bios_serialcom(_COM_INIT, port, SETTINGS);

window(4, 25, 80, 25);

for(;;) {
  status = _bios_serialcom(_COM_STATUS, port, 0);
  if (status & 512)  
    gotoxy(10, 1);
    textcolor(BLUE);
    cprintf("Overrun Error");
  if (status & 1024)  
    gotoxy(10, 2);
    textcolor(MAGENTA);
    cprintf("Parity Error");
  if (status & 2048)  
    gotoxy(10, 3);
    textcolor(RED);
    cprintf("Framing Error");
  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)
        msg_in[count_in++] = out;
      if (out == 'r')
        {
          gotoxy(count_out+1, 1);
          display(msg_in, BLUE);
          count_in = 0;
          gotoxy(count_out+1, 1);
        }
    }  

  if (kbhit())    /* if a keystroke is currently available */
    {
      gotoxy(count_out+1, 1);
      in = getche();  /* get a character by echoing onto the
    screen */
      if (in == 27)   /* if ESC */
        break;
      if (in == 'b' && count_out > 0)
        {
          count_out = count_out - 2;
          cprintf(" ");
        }
      else
        msg_out[count_out] = in;
      if (count_out == 74)
        {
          in = 'r';
          msg_out[74] = 'r';
        }
      count_out++;
      if (in == '\r')
        {
          display(msg_out, MAGENTA);
          for (i = 0; i < count_out; i++)
            _bios_serialcom(_COM_SEND, port, msg_out[i]);
          count_out = 0;
          clrscr();
        }
    }
}

void display(char arr[75], int COLOR) {
  static int line = 1;
  int i;
  window(10, 11, 70, 23);
  gotoxy(1, line);
  textcolor(COLOR);
  textbackground(YELLOW);
  cprintf("\\n");
putch(1);
  for(i = 0; arr[i] != 'r'; i++)
    putch(arr[i]);
  line = wherey();
  window(4, 25, 80, 25);
  textcolor(YELLOW);
  textbackground(BLUE);