EMBEDDED

Chapter 6

Communication Interfaces

CHAPTER OBJECTIVES

After reading this chapter, you will be able to:

- Appreciate the need for communication interfaces
- Learn the details of various serial communication interfaces
- Understand the Ethernet and wireless LAN interfaces
- Grasp the details of IEEE 1394 interface
- Gain knowledge of infrared and Bluetooth wireless interfaces

Most of the embedded systems have to interface with the external world. This requirement may be to transmit the data to a PC or workstation, or to interact with another system for sharing the data. To meet this requirement, the embedded systems need to be provided with communication interfaces. In this chapter, we will study the various standard communication interfaces that can be provided to the embedded systems. We will discuss the details of RS232/UART, RS422/RS485, IEEE 1394, Universal Serial Bus (USB), Ethernet as well as wireless interfaces such as IrDA, IEEE 802.11 and Bluetooth.

6.1 Need for Communication Interfaces

The need for providing communication interfaces arises due to the following reasons:

- The embedded system needs to send data to a host (a PC or a workstation). The host will analyse of data and present the data through a Graphical User Interface.
- The embedded system may need to communicate with another embedded system to transmit/receive data. Providing a standard communication interface is preferable rather than providing a proprietary interface.
- A number of embedded systems may need to be networked to share data. Network interfaces need to be provided in such a case.

- An embedded system may need to be connected to the Internet so that anyone can access the embedded system. An example is a real-time weather monitoring system. The weather monitoring system can be Internet-enabled using TCP/IP protocol stack and HTTP server.
- Mobile devices such as cell phones and palmtops need to interact with other devices such as PCs and laptops for data synchronization. For instance, you need to ensure that the address book on your palmtop is the same as that on your laptop. When the palmtop comes near the laptop, automatically the two can form a network to exchange data.
- For some embedded systems, the software may need upgradation after it is installed in the field. The software can be upgraded through communication interfaces.

Due to these reasons, providing communication interfaces based on standard protocols is a must. Not surprisingly, many micro-controllers have on-chip communication interfaces such as a serial interface to meet these requirements. Now, we will discuss the following communication interfaces.

In Brief...

Embedded systems are provided with communication interfaces for monitoring and control by a host system or a node on a network. Through these interfaces, an embedded system can also be accessed over the Internet.

- RS 232/UART
- RS 422, RS 485
- Universal Serial Bus
- Infrared
- IEEE 1394 Firewire
- Ethernet
- IEEE 802.11 wireless interface
- Bluetooth

For each of these interfaces, we will discuss the hardware details and the protocol stack that needs to be implemented in software.

6.2 RS232/UART

RS232 is a standard developed by Electronic Industry Association (EIA). This is one of the oldest and most widely used communication interfaces. The PC will have two RS232 ports designated as COM1 and COM2. Most of the micro-controllers have an on-chip serial interface. The evaluation boards of the processors are also connected to the host system using RS232.

RS232 is used to connect a DTE (Data Terminal Equipment) to a Data Circuit Terminating Equipment (DCE). A DTE can be a PC, serial printer or a plotter. DCE can be a modem, mouse, digitizer or a scanner. RS232 interface specifies the physical layer interface only. The specifications describe the physical, mechanical, electrical, and procedural characteristics for serial communication.

In Brief...

RS232 standard is used for serial communication between two devices for speeds up to 115.2 Kbps over distances up to 100 meters.

RS 232 is a standard for serial communication, i.e. the bits are transmitted serially. The communication between two devices is in full duplex, i.e. the data transfer can take place in both directions.

6.2.1 RS232 Communication Parameters

When two devices have to communicate through RS232, the sending device sends the data character by character. The bits corresponding to the character are called data bits. The data bits are prefixed with a bit called start bit, and suffixed with one or two bits called stop bits. The receiving device decodes the data bits using the start bit and stop bits. This mode of communication is called asynchronous communication because no clock signal is transmitted. In addition to start bit and stop bits, an additional bit called parity bit is also sent. Parity bit is used for error detection at the receiving end.

For two devices to communicate with each other using RS232, the communication parameters have to be set on both the systems. And, for a meaninginful communication, these parameters have to be the same. The various communication parameters are listed below:

Data rate: The rate at which data communication takes place. The PC supports various data rates such as 50, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200 bps. The oscillator in the RS232 circuitry operates at 1.8432 MHz and it is divided by 1600 to obtain the 115200 data rate.

Data bits: Number of bits transmitted for each character. The character can have 5 or 6 or 7 or 8 bits. If you send ASCII characters, the number of bits is 7.

Start bit: The bit that is prefixed to the data bits to identify the beginning of the character.

Stop bits: These bits are appended to the data bits to identify the end of character. If the data bits are 7 or 8, one stop bit is appended. If the data bits are 5 or 6, two stop bits are appended.

Parity: The bit appended to the character for error checking. The parity can be even or odd. For even parity, the parity bit (1 or 0) will be added in such a way that the total number of bits will be even. For odd parity, the parity bit will make the total number of bits odd. If the parity is set to 'none', the parity bit is ignored. For example, if the data bits are 1010110, the parity bit is 0 if even parity is used; and the parity bit is 1 if odd parity is used. At the receiving end, the device will calculate the parity bit and if the received parity bit matches with the calculated parity bit, it can be assumed that the data is without errors. But this is not the reality. If two bits are in error, the receiver cannot detect that there is an error!

Flow Control: If one of the devices sends data at a very fast rate and the other device cannot absorb the data at that rate, flow control is used. Flow control is a protocol to stop/resume data transmission. This protocol is also known as handshaking. If you are sure that there will be no flow control problem, there is no need for handshaking. You can do hardware handshaking in RS232 using two signals: Request to Send (RTS) and Clear to Send (CTS). When a device has data to send, it asserts RTS and the receiving device asserts CTS. You can also do software handshaking — a device can send a request to suspend data transmission by sending the character Control S (0x13). The signal to resume data transmission is sent using the character Control Q (0x11). This software handshaking is also known as XON/XOFF.

In Brief...

The communication parameters to be set for serial communication are: data rate, number of data bits, number of stop bits, parity and flow control.

Notes...

Communication using RS232 involves lot of overhead due to the additional bits to be transmitted—for 7-bit data, the additional bits to be transmitted are 1 start bit, 1 stop bit and 1 parity bit.

6.2.2 RS 232 Connector Configurations

RS232 standard specifies two types of connectors: 25-pin connector and 9-pin connector. In the 25-pin configuration, only a few pins are used. The description of these pins is given in Table 6.1.

Pin number	Function (abbreviation)
1	Chassis ground
2	Transmit data (TXD)
3	Receive data (RXD)
4	Request To Send (RTS)
5	Clear To Send (CTS)
6	Data Set Ready (DSR)
7	Signal Ground (GND)
8	Carrier Detect (CD)
20	Data Terminal Ready (DTR)
22	Ring Indicator (RI)

Table 6.1: 25-pin Connector Pin Details

For the 9-pin connector, the pin details are given in Table 6.2.

Pin number	Function (abbreviation)
1	Carrier Detect (CD)
2	Receive Data (RXD)
3	Transmit Data (TXD)
4	Data Terminal Ready (DTR)
5	Signal Ground (GND)
6	Data Set Ready (DSR)
7	Request to Send (RTS)
8	Clear to Send (CTS)
9	Ring Indicator (RI)

Table 6.2: 9-pin Connector Pin Details

For transmission of 1's and 0's, the voltage levels are defined in the standard. The voltage levels are different for control signals and data signals. These voltage levels are given in Table 6.3. The voltage level is with reference to the local ground and hence RS232 uses unbalanced transmission.

Signal	Voltage Level
Data input	+3 volts and above for 0-3 volts and below for 1
Data output	+5 volts and above for 0-5 volts and below for 1
Control input	+3 volts and above for 1 (ON)-3 volts and below for 0 (OFF)
Control output	+5 volts and above for 1(ON)-5 volts and below for 0 (OFF)

Table 6.3: Voltage Levels for Data and Control Signals

In Brief...

Two types of connectors are used in RS232—25-pin connector and 9-pin connector. The three important pins are—2 for transmit, 3 for receive and 7 for signal ground in a 25-pin connector.

Note that the voltage levels used in RS232 are different from voltage levels used in embedded systems (as most chips use 5 volts and below only). Another problem is that the processor gives out the data in parallel format, not in serial format. These problems are overcome through the use of UART (Universal Asynchronous Receive Transmit) chips.

Notes...

RS232 uses unbalanced transmission i.e., the voltage levels are measured with reference to the local ground. Hence, this transmission is susceptible to noise.

6.2.3 UART

In Brief...

Universal Asynchronous Receive Transmit (UART) chip converts the parallel data received from the processor into serial format and sends it to RS232 level shifter. It also takes the serial data from the RS232 level shifter and converts into parallel format.

The processors process the data in parallel format, not in serial format. To bridge the processor and the RS232 port, Universal Asynchronous Receive Transmit (UART) chip is used. UART has two sections: receive section and transmit section. Receive section receives the data in serial format, converts it into parallel format and gives it to the processor. The transmit section takes the data in parallel format from the processor and converts it into serial format. The UART chip also adds the start bit, stop bits and parity bit. Many micro-controllers have on-chip UART. However, the necessary voltage level conversion has to be done to meet the voltage levels of RS232. This is achieved using a level shifter as shown in Fig. 6.1.

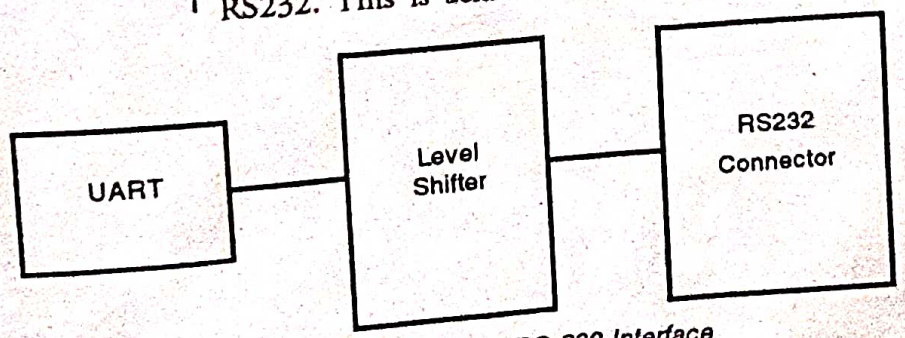


Fig. 6.1: Hardware for RS 232 Interface

UART chip operates at 5 Volts. The level conversion to the desired voltages is done by the level shifter, and then the signals are passed on to the RS232 connector.

RS232 standard specifies a distance of 19.2 meters. However, you can achieve distances up to 100 meters using RS232 cables. The data rates supported will be dependent on the UART chip and the clock used.

Most of the processors including Digital Signal Processors have on-chip UART. ICs such as MAX 3222, MAX 3241 of Maxim can be used as level shifters.

6.2.4 Null Modem Cable Connection

If you want to connect two DTEs such as two PCs, you need to interconnect the two RS232 ports using a null modem cable. The null modem cable connections are shown in Fig. 6.2. Fig. 6.2(a) shows the connections for 25-pin connectors and Fig. 6.2(b) shows the connections for 9-pin connectors. However, the minimal connections required are for TD, RD and GND. If you are using 25-pin connector, it is sufficient if you connect pin 2 to pin 3, pin 3 to pin 2 and pin 7 to pin 7. This is the minimal configuration.

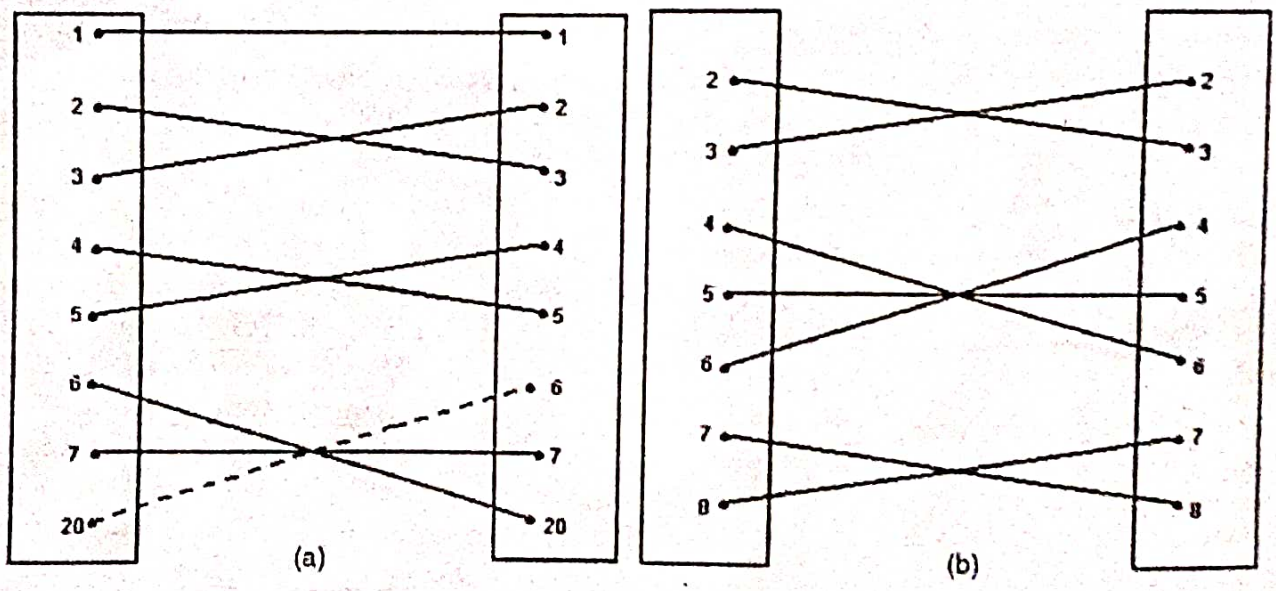


Fig. 6.2 Null Modem Cable Connections: (a) 25-pin Connectors (b) 9-pin Connectors

To make two devices communicate with each other using RS232 interface, you need to connect the two PCs using a null modem cable and set the communication parameters on both the devices. You can experiment with various parameters for serial communication using the HyperTerminal program on Windows. Fig. 6.3 shows the screenshot to set the serial communication parameters. To obtain this screen, you need to do the following:

In Brief...

Using a null modem cable and HyperTerminal program, you can establish serial communication between two machines running the Windows operating system.

- Click on Start Menu
- Go to Programs
- Go to Accessories
- Go to Communications
- Go to HyperTerminal
- Double click on HYPERTRM.EXE

- Enter a name for New Connection "xxxx" and Click OK
- You will get a screen with the title "Connect To". Select COM1 or COM2 from the "Connect Using" list box and click OK.

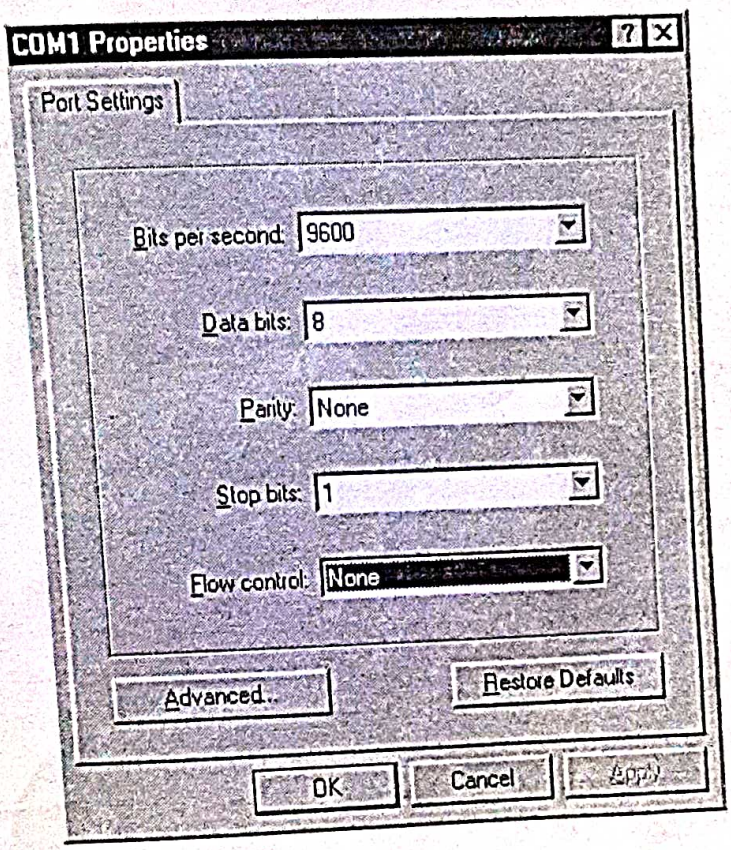


Fig. 6.3: Screenshot to set the Serial Communication Parameters

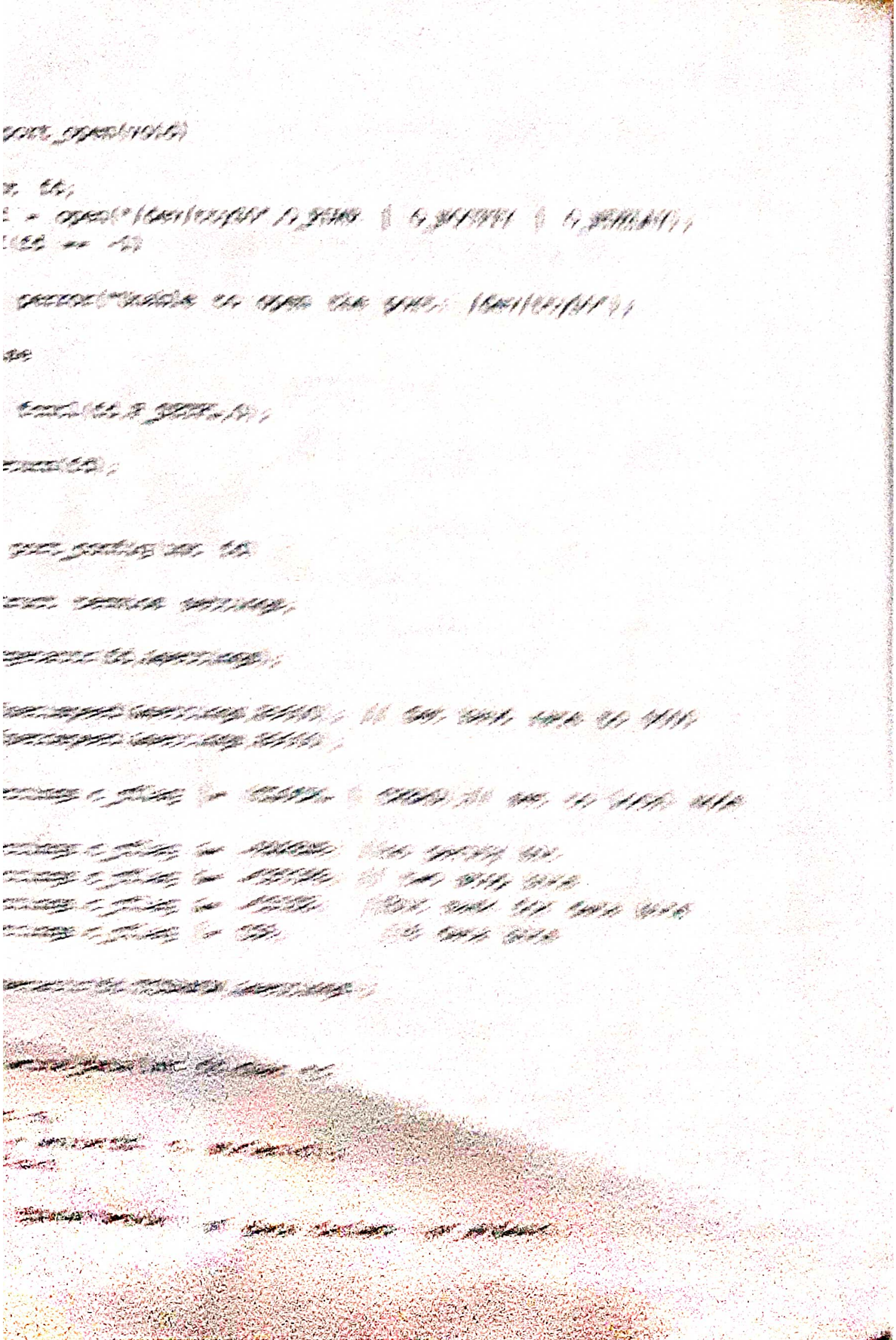
6.2.5 Serial Communication Programming

In this section, we will study how to establish communication between two PCs, one PC running the Windows operating system and another running the Linux operating system. On the Windows system, we will use the HyperTerminal program and on the Linux system we will write a C program to send and receive data from the Windows system. You need to interconnect the two PCs through a null modem cable to test this program.

On the Windows system, you need to run the HyperTerminal as described earlier. On the Linux system you need to compile and execute the code given in Listing 6.1.

Listing 6.1 serialcom.c

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
```



```
int port_open(void)
{
   int fd;
   fd = open("/dev/ttyS0",O_RDWR | O_NOCTTY | O_NDELAY);
   if(fd == -1)
   {
      perror("Unable to open the port: /dev/ttyS0");
   }
   else
   {
      fcntl(fd,F_SETFL,0);
   }
   return(fd);
}

void port_config(int fd)
{
   struct termios settings;

   tcgetattr(fd,&settings);

   cfsetispeed(&settings,B9600); // Set baud rate to 9600
   cfsetospeed(&settings,B9600);

   settings.c_cflag |= (CLOCAL | CREAD);// set to local mode

   settings.c_cflag &= ~PARENB; //no parity bit
   settings.c_cflag &= ~CSTOPB; // two stop bits
   settings.c_cflag &= ~CSIZE;   //bit mask for data bits
   settings.c_cflag |= CS8;       //8 data bits

   tcsetattr(fd,TCSANOW,&settings);
}

void write_data(int fd,char *c)
{
   int n;
   n = write(fd, c, strlen(c));
   if(n<0)
   {
      fputs("write() of data failed! \n",stderr);
```

```
  }
}

void read_data(int  fd)
{
  char array[255];
  char *ptr;
  int nbytes;

  ptr = array;
  while((nbytes = read(fd, ptr, array + sizeof(array)-ptr-1)) > 0)
  {
     ptr += nbytes;
     if((ptr[-1] == '\n') || (ptr[-1] == '\r'))
        break;
  }
  *ptr = '\0';
  printf("Received String: %s",array);
}

void port_close(int  fd)
{
   close(fd);
}

int main(void)
{
    int  fd;
    fd   = port_open();
    port_config(fd);
    write_data(fd,"Hello World");
    read_data(fd);
    port_close(fd);
    return 0;
}
```

The function **port_open(void)** opens the serial port. **/dev/ttyS0** is the device file. You can find out the device file of your Linux installation in the /dev directory and use that file name instead of **/dev/ttyS0**.

The function **port_config(int fd)** is to configure the port parameters.

The function **write_data (int fd,** char *c) is to write data onto the port. The pointer to a character array is passed from this function.

The function `read_data(int fd)` is to read the data from the port.

The function `port_close(int fd)` closes the port.

The main(void) function calls the functions to open the port, configure the port parameters, write the data, read the data and then close the port.

In Brief...

Serial communication programming involves opening the device file, configuring the port by setting the communication parameters, reading/writing data and then closing the port.

You can compile this program using the following shell command. Note that the $ sign is the system prompt.

```
$gcc serialcom.c
```

The file a.out will be created. You can execute this file by giving the command

```
$./a.out
```

The message "Hello World" will be displayed on the Windows system. You can type some text on the Windows system and it will be displayed on the Linux system.

6.3 RS422/RS485

RS422 standard for serial communication is used in noisy environments. Using this standard interface, the distance between two devices can be up to 1200 meters. Twisted copper cable is used as the transmission medium. Unlike RS232 in which the voltage levels are measured with reference to local ground, in RS422, voltage difference between the two copper wires represents the logic levels. Hence, RS422 uses balanced transmission. Two channels are used for transmit and receive paths. As compared to RS232, RS422 is better suited to work in noisy environments over longer distances because of balanced transmission.

In Brief...

In RS422 and RS485, twisted copper pair is used as transmission medium. The voltage difference between the two wires represents logic levels, i.e. balanced transmission is used and hence this transmission is immune to noise.

RS485 is a variation of RS422 to connect a number of devices in a network. An RS485 controller chip is used on each device. A network using RS485 protocols operates in a Master/Slave configuration. Up to 512 devices can be networked. Using one twisted pair, half-duplex communication can be achieved and using two twisted pairs, full-duplex communication can be achieved.

Chips such as MAX3488 are used for RS422. MAX 3483 is an RS 485 controller for half-duplex communication and MAX 3491 is for full-duplex communication.

6.4 US

Universal Serial Bus has gained immense popularity in recent years. Desktops, laptops, printers, display devices, video cameras, hard disk drives, CDROM drives, audio equipment etc. are now available with USB interface. Using USB, a number of devices can be networked using Master/Slave architecture. A host, such as the PC, is designated as a master. As shown in Fig. 6.4, a number of devices, up to a maximum of 127, can be connected in the form of an inverted tree. On the host, such as a PC,