

4.2 Serial Communication:

Serial communication can send data between two devices at high speed up to 300 kbps using the RS232 protocol that specify how data is formatted for transmitting and when and how each node may transmit, Events that may occur at a serial port include sending and receiving of data, Changes in handshaking signals, and sending and receiving of error messages. There are two ways for an application to cause and detect these events. One way is to have the program automatically jump to a routine when an event occurs. The application responds quickly and automatically to activity at the port, without having to waste time checking, only to learn that no activity has occurred. This type of programming is called **event-driven** because an external event can break in at any time and cause the program's execution to branch to a particular routine. Event-driven programming approach is used PC and PIC microcontroller, to eliminate data loss and save the CPU time in this project.

RS 232: is designed to handle communications between two devices, with a distance Limit of 50 to 100 feet, depending on the bit rate and cable type. The pins and signals associated with the 9-pin connector are described below in Fig. 4-3 and the corresponding Serial Port Pin and Signal Assignments in Table 4-1.

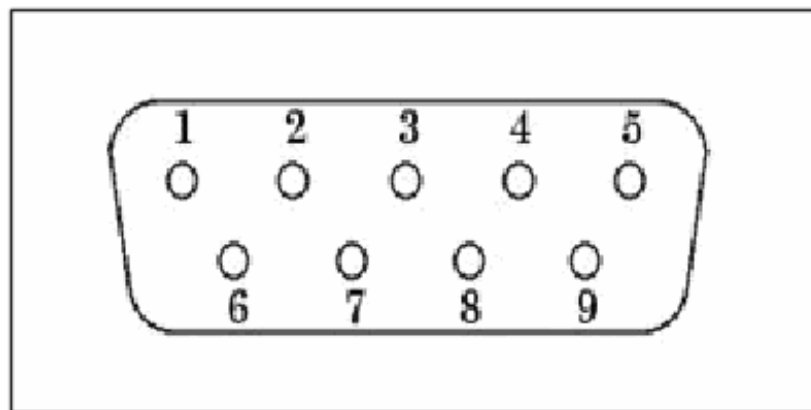


Figure 4-3: Serial port pins.

Serial Port Pin and Signal Assignments			
Pin	Label	Signal Name	Signal Type
1	CD	Carrier Detect	Control
2	RD	Received Data	Data
3	TD	Transmitted Data	Data
4	DTR	Data Terminal Ready	Control
5	GND	Signal Ground	Ground
6	DSR	Data Set Ready	Control
7	RTS	Request to Send	Control
8	CTS	Clear to Send	Control
9	RI	Ring Indicator	Control

Table 4-1: Serial port pins assignment.

Why serial communication:

1. It's very common. Every PC has one or more RS-232 ports and most of microcontroller support serial communication on hardware level.
2. Interface chips make it easy to convert 5V Logic to RS-232 voltage level.
3. Links can be 50 to 100 feet long. Due to high voltage level used in RS 232.
4. You need just three wires for a 2-way link, this can save microcontroller I/O pins in other hand A parallel link typically has eight data Lines, two or more control signals, and several ground wires.

Synchronous and Asynchronous Communication:

The RS-232 standard supports two types of communication protocols: synchronous and asynchronous.

- **Synchronous communication:**

Using the synchronous protocol, all transmitted bits are synchronized to a common clock signal. The two devices initially synchronize themselves to each

other, and then continually send characters to stay synchronized. Even when actual data is not really being sent, a constant flow of bits allows each device to know where the other is at any given time. That is, each bit that is sent is either actual data or an idle character. Synchronous communications allows faster data transfer rates than asynchronous methods, because additional bits to mark the beginning and end of each data byte are not required.

- **Asynchronous communication:**

Using the asynchronous protocol, each device uses its own internal clock resulting in bytes that are transferred at arbitrary times. So, instead of using time as a way to synchronize the bits, the data format is used. In particular, the data transmission is synchronized using the start bit of the word, while one or more stop bits indicate the end of the word. The requirement to send these additional bits causes asynchronous communications to be slightly slower than synchronous. However, it has the advantage that the processor does not have to deal with the additional idle characters. Most serial ports operate asynchronously.

Serial Data Format:

The serial data format includes one start bit, between five and eight data bits, and one stop bit. A parity bit and an additional stop bit might be included in the format as well. The diagram below illustrates the serial data format.

The format for serial port data is often expressed using the following notation:

Number of data bits - parity type - number of stop bits

For example, 8-N-1 is interpreted as eight data bits, no parity bit, and one stop bit, while 7-E-2 is interpreted as seven data bits, even parity, and two stop bits. The remaining bits are called *framing bits* because they frame the data bits, in the project we use 8-N-1 format because it's supported by both PC and Microcontroller, in Fig. 4-6 sample code for using MSCOMM ACTIVX for asynchronous communication. Fig 4-7A and Fig 4-7B show a sample code for PIC microcontroller.

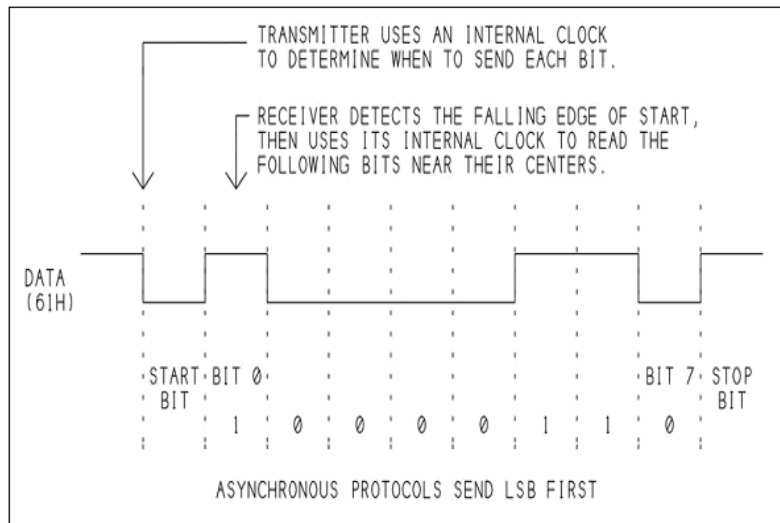


Figure 4-4: Serial data format [].

How Are the Bits Transmitted?

By definition, serial data is transmitted one bit at a time. The order in which the bits are transmitted follows these steps:

1. The start bit is transmitted with a value of 0.
2. The data bits are transmitted. The first data bit corresponds to the least significant bit (LSB), while the last data bit corresponds to the most significant bit (MSB).
3. The parity bit (if defined) is transmitted.
4. One or two stop bits are transmitted, each with a value of 1.

The number of bits transferred per second is given by the **baud rate**. The transferred bits include the start bit, the data bits, the parity bit (if defined), and the stop bits, but refer that the logic 0 in the RS232 protocol is defined as equal to or more positive than +5V, and a logic 1 is defined as equal to as or more negative than -5V.

Start and Stop Bits:

As described in Synchronous and Asynchronous Communication, most serial ports operate asynchronously. This means that the transmitted byte must be identified by start and stop bits. The start bit indicates when the data byte is about to begin and the stop bit(s) indicates when the data byte has been transferred. The process of identifying bytes with the serial data format follows these steps:

1. When a serial port pin is idle (not transmitting data), then it is in an "on" state.

2. When data is about to be transmitted, the serial port pin switches to an "off" state due to the start bit.
3. The serial port pin switches back to an "on" state due to the stop bit(s). This indicates the end of the byte.

Data bits:

The data bits transferred through a serial port might represent device commands, sensor readings, error messages, and so on. The data can be transferred as either binary data or as text (ASCII) data.

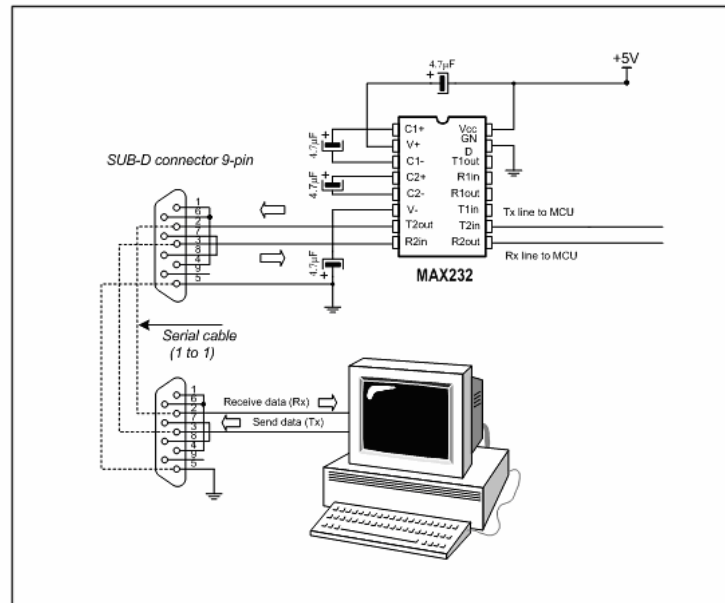


Figure 4-5: The connection between MAX232 with serial port [1].

Converting between 5V Logic and RS-232:

As maintained before the RS232 has voltage level different from the microcontroller inputs and outputs 5V Logic, Interfacing 5V Logic to an RS-232 port requires converting to and from RS-232 Levels. This is accomplished using the MAX232 IC that will convert 5V logic to RS232 logic, the connection of MAX232 with PC and microcontroller is shown in Fig 4-5.

```

Public Class serial
    Inherits System.Windows.Forms.Form

    Private Sub MSComm1_OnComm(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MSComm1.OnComm

        Dim data As String

        If MSComm1.CommEvent = 2 Then ' event type is receiving data (event
symbol =2)
            data = MSComm1.Input ' receive data as string ASCII format
        End If

    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        With MSComm1
            .RThreshold = 1 'fire Rx (receive event)when ever
            'Buffer has 1 byte
            .InputLen = 1 'input data length, 1 byte at time
            .Settings = "9600,n,8,1" '9600 baud rate , no parity , 8 data_
            'Bits, 1 stop bit
            .DTREnable = False 'disable DTR (handshaking signal)

            .CommPort = 1 'open port
            .PortOpen = True 'open com1 port
        End With
    End Sub

    Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        MSComm1.PortOpen = False 'close com1 port

    End Sub

    Private Sub senddata(ByVal j As Integer)
        MSComm1.Output() = txt1.Text 'send ascii data u can

    End Sub
End Class

```

Figure 4-6: A simple code to control the serial port using VB.Net.

In the following Figures a sample code shows the interfacing with PIC Microcontroller by using interrupt procedure in order to avoid the dependency on time, which gives the ability to send and receive data at any time during the working time (as shown in Fig 4-7A and Fig 4-7B).

Program Master1

```
Dim i As Byte
Dim j As Byte
Dim add As Byte
Dim num As Byte
Dim data as byte[4]
Dim data1 as byte[4]
Dim data2 as byte[4]
Dim data3 as byte[4]

sub procedure interrupt

    If RCSTA.OERR = 0 Then          ' no errors
        If PIR1.RCIF = 1 Then      ' interrupt receive
            If i = 0 Then          ' byte after interrupt
                data[0] = rcreg
                add = data[0] and $F0 ' determining the address
                num = data[0] and $0F ' determining the number of bytes
                i = i + 1
            Else
                data([i] = rcreg)    ' receiving the data
                i = i + 1
            End If

            If i = (num + 1) Then    ' finish the receiving
                i = 0

                If j = 1 Then        ' identefying the data
                    For i = 0 To num
                        data1[i] = data[i]
                    Next i
                    i = 0
                End If
            End If
        End If
    End If
end sub
```

Figure 4-7A: Part of the code in the PIC Microcontroller.

```

        If j = 2 Then
            For i = 0 To num
                data2[i] = data[i]
            Next i
            i = 0
        End If

        If j = 3 Then
            For i = 0 To num
                data3[i] = data[i]
            Next i
            i = 0
            j = 0
        End If

        add = add + $10
        txreg = add
        j = j + 1
    End If

End If
Else
    RCSTA(0.4 = 0)
    RCSTA(0.4 = 1)
End If

End Sub

main:
    trisb = 0                ' initializing the registers
    portb = 0
    i = 0
    j = 1

    INTCON = %11000000
    PIE1 = %00100000
    TXSTA.4 = 0
    Rcsta.7 = 1
    rcsta.4 = 1
    rcsta.6 = 0
    txsta.2 = 1
    txsta.6 = 0
    txsta.txen = 1
    spbrg = 129
    txreg = $10

end.

```

Figure 4-7B: Part of the code in the PIC Microcontroller.