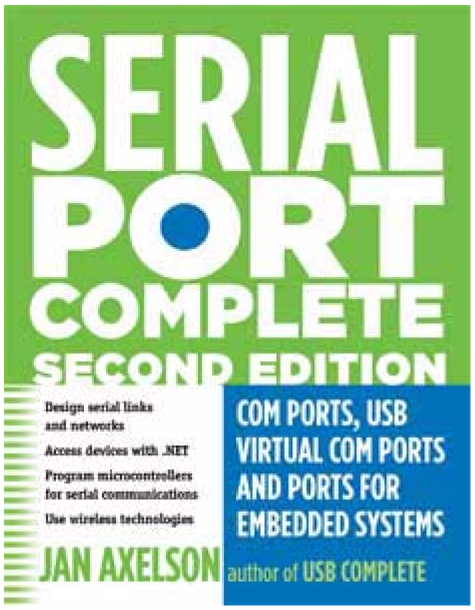
This article is for anyone developing serial-port applications using the Parallax Basic Stamp (*www.parallax.com*). The text is adapted from the first edition of my book *Serial Port Complete*. The PC example code is written for Visual Basic 5 using the MSComm control. Complete code examples from the first edition are available at www.Lvr.com/files/spc.zip.

Serial Port Complete Second Edition adds firmware examples for PICBASIC PRO, Microchip's C18 compiler, and .NET. To make room, I had to cut the Stamp examples, but I wanted to continue to make the information available, so here it is.

More code examples for .NET and PIC microcontrollers are available from:

www.Lvr.com/serport.htm

Jan Axelson
www.Lvr.com


SERIAL PORT COMPLETE
SECOND EDITION
Design serial links and networks
Access devices with .NET
Program microcontrollers for serial communications
Use wireless technologies
JAN AXELSON
COM PORTS, USB VIRTUAL COM PORTS AND PORTS FOR EMBEDDED SYSTEMS
author of USB COMPLETE

Serial Port Complete Second Edition
COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems
Jan Axelson

ISBN 978-1931448-06-2

Lakeview Research LLC
www.Lvr.com

Using the Basic Stamp's Serial Port

Jan Axelson

An example of a microcontroller with a serial port is Parallax Inc.'s Basic Stamp. The Stamp is a microcontroller with a Basic interpreter on-chip. There are several models. Each uses a microcontroller from Microchip's PIC family. The Stamp I uses a 16C56, and the Stamp II, a 16C57.

Because of its enhanced features, the Stamp II is a better choice than the Stamp I for many applications that use serial communications, especially with multiple nodes. The module has the footprint of a 24-pin DIP chip, with 16 pins of I/O, 2 kilobytes of EEPROM for storing user programs, and 24 bytes available for variable storage. For a power supply, it can use an unregulated DC supply from just over +5V to +15V (such as a 9V battery) or a regulated +5V supply.

Figure 1 shows the pinout. In some of the Stamp's documentation, *GND* is referred to as *Vss*, *Sin,* as *RX*, and *Sout*, as *TX*. *Sin* and *Sout* could stand for *SerialIn* and *SerialOut*, but I prefer to think of them as *StampIn* and *StampOut*, because these more precisely describe the direction of data flow.

The Stamp II uses an asynchronous serial interface for programming and debugging. To communicate with the Stamp, Parallax provides the Stamp2 host program, which runs under DOS and includes a text editor for writing programs, plus the ability to load Basic programs into the Stamp II and load and save programs on disk. You can also use the serial port to communicate with other devices.

The Stamp IC is a smaller and simpler version with 14 pins and 8 I/O bits. An earlier version resided on a larger circuit board and used through-hole rather than surface-mount components. Both versions have 256 bytes for storing user programs and 14 bytes of variable storage. Instead of connecting to a PC's serial port, The Stamp I uses a parallel-port interface for programming and debugging. (Serial-port software is available for the Macintosh, which has no parallel printer port.) As on the Stamp II, programs can use asynchronous serial communications to communicate with other devices.

Both Stamps use versions of Parallax's PBasic interpreter. Like Basic-52, PBasic has much in common with other Basics, yet is customized for use in a microcontroller. The Stamps' PIC microcontrollers are smaller and simpler in design than the 8052, so their interpreter is smaller and simpler to match.

Parallax offers carrier boards for both Stamps. Each includes a socket for the Stamp, a connector for PC communications, battery clip, reset button, and prototyping area. You can develop a project right on the carrier board or you can use a breadboard, your own perfboard, or any other method you prefer. Other vendors offer products with added features to make project development easy.

Serial Links

The Stamp II may use as many as five serial-port signals plus a ground wire, though all of these aren't always needed. Table 1 lists the signal lines and their uses.

If you use the provided carrier board and cable, all you have to do is plug the cable into the board and a port on a PC (Figure 2A). The carrier board's connector is wired as a 9-pin RS-232 DCE device, so you don't need a null-modem cable or converter (as described in Chapter 7). The carrier board connects *DSR* and *RTS*. This connection enables the host software to detect which port connects to the Stamp.

If you don't use the carrier board, you can connect a power supply and serial cable directly to the Stamp. If you want the host to autodetect the port, connect *DSR* and *RTS* in your circuits. If you don't include this connection, you can specify a port in the command line when you run the host software: *Stamp2 /2*.

Instead of Parallax's provided cable, you can use any standard RS-232 cable that contains the needed wires. Standard serial cables may have 3, 9, or 25 wires. A standard cable with 9 or more wires will work fine. If you make your own cable, you can make the *DSR/RTS* connection at the PC's connector and eliminate two wires.

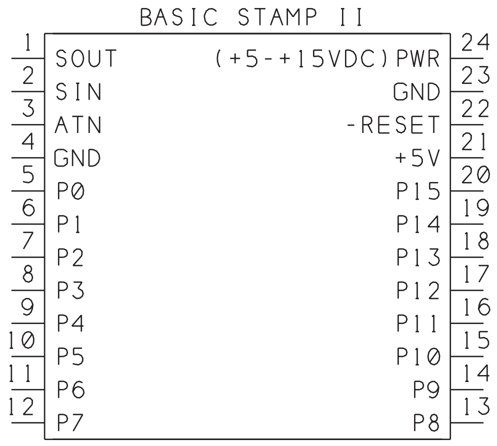


Figure 1: Pinout of the Basic Stamp II.

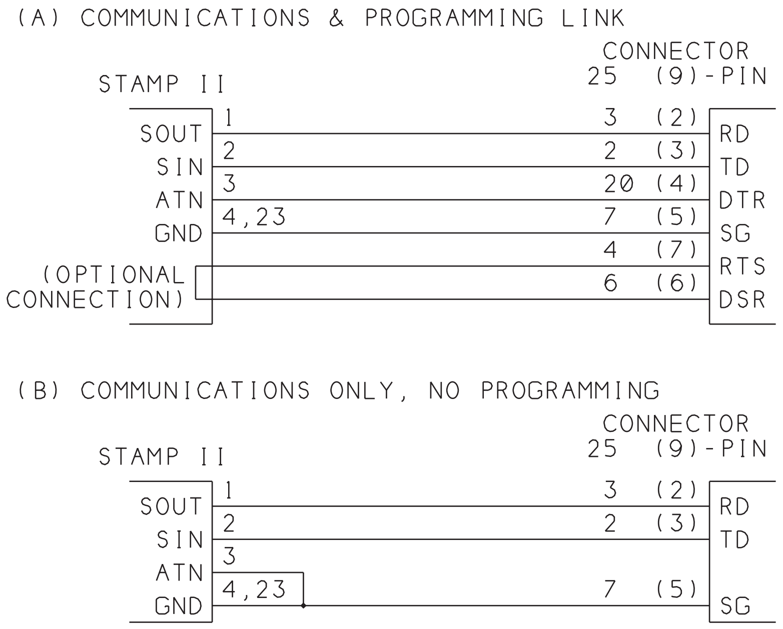


Figure 2: Wiring for Basic Stamp II serial links.

When a Stamp project is complete, the host software's autodetecting and program-downloading abilities are no longer needed (Figure 2B). *ATN* should be held low to prevent accidental resets. To defeat any handshaking enabled at the PC, you can tie *RTS* to *CTS* and tie *DTR, DSR*, and *CD* together in the serial cable or at the serial connector in your circuits. However, it's usually a simple matter to disable all handshaking at the PC, and then you don't need these connections.

When wiring your own circuits, be aware that the Stamp's *-Reset* pin connects to more than the PIC's *-MCLR* (reset) input. Additional circuits at *-MCLR* ensure a clean reset signal on power-up and enable the host software to reset the chip with *ATN*. Don't tie *-Reset* high if you need to communicate with the host software! If you do connect *-Reset* directly to +5V, the host software won't be able to reset or communicate with the Stamp. *-Reset* may be left open or connected to a normally-open pushbutton with a pullup to +5V.

A Firmware UART

Although some PIC microcontrollers have hardware UARTs, the Stamps' microcontrollers don't. Instead, the Basic Stamps' UARTs are implemented in firmware, with the UART's functions programmed into the interpreter.

The serial ports on the Stamp IC and II are similar in many ways. Neither has any buffers at all. To read a byte at the serial port, the Stamp must execute a `Serin` statement and wait for a byte to arrive. If the byte arrives before `Serin` executes, the Stamp won't see it. If no byte arrives, the Stamp IC will wait forever. To prevent endless waits, the Stamp II allows use of a `Timeout` modifier that enables the Stamp to jump to a line label if a specified number of milliseconds pass with no activity at the port.

Because the Stamp's port is implemented in firmware, not hardware, it can include options that hardware ports don't normally have. The outputs can be inverted or not, so they're

Table 1: Serial-port connections between a Basic Stamp II and a PC.

Stamp II		**PC Serial Port**			**Description**
Pin	**Signal**	**Signal**	**9-pin**	**25-pin**	
2	Sin	TD	3	2	Carries data from the PC to the Stamp.
1	Sout	RD	2	3	Carries data from the Stamp to the PC.
-	On carrier board	RTS	7	4	Connects to *DSR* to enable the host software to detect which of a PC's serial ports connects to a Stamp.
-	-	CTS	8	5	No connection.
-	On carrier board	DSR	6	6	See *RTS*.
4, 23	Vss	SG	5	7	Signal ground.
-	-	CD	1	8	No connection.
3	ATN	DTR	4	20	Resets the Stamp. The PC's host software pulses *DTR* high to enable downloading of a new program.

usable with or without inverting drivers and receivers. A Stamp can use any of its I/O bits for serial communications. It can even have multiple serial ports, using different bits for each (though it can use just one at a time). These abilities greatly simplify project development for Stamps used in serial links and networks. The Stamp can reserve *Sin* and *Sout* for PC communications for loading and saving programs, viewing debug messages, and other experimenting, and use other port bits for the network. When all is working fine, you can disconnect *Sin* and *Sout* and use the Stamp on its own in the network.

If you don't need 2-way communications, you can even have a 1-bit port. If you're connecting the serial ports of two Stamps, you can use the Stamp's open baudmode for 2-way communications over 1 bit.

If you want to know how to write a firmware UART, an excellent resource is Scott Edwards' *PIC Source Book,* which contains clearly commented assembly-code routines that emulate each of the Basic Stamp's instructions, including *Serin* and *Serout*.

Options and Features

Each read or write to the Stamp's serial port has to specify the bit rate, pin number, and other information. The values are easily stored in variables or constants, however, so each needs to appear just once in a program. The statement to read a serial port on the Stamp II is:

```
SerIn rpin {\fpin}, baudmode, {plabel,} {timeout, tlabel,}
[InputData]
```

To write, it's:

```
SerOut tpin{\fpin}, baudmode, {timeout, tlabel,} [OutputData]
```

or

```
SerOut tpin, baudmode, {pace} [OutputData]
```

The parameters in the above statements are as follows:

`rpin` and `tpin` specify the pin to use for the serial input and output. Set these to 16 to use the Stamp's *Sin* or *Sout* pin, or 0–15 to select a port pin. `Serout` automatically configures the requested pin as an output and leaves it in that state after transmitting. In a similar way, `Serin` configures the requested pin as an input and leaves it in that state after receiving data or a timeout.

`\fpin` designates a pin for handshaking. On executing `Serin`, the Stamp sets *fpin* low (or high, if using inverted signals) to indicate that it's ready to receive data. `Serout` will wait for its designated *fpin* to be in the appropriate state before sending data.

`baudmode` is a 16-bit word that holds several settings. It specifies the bit rate, the number of data bits and parity, and the polarity of the bits to send and read. It can also enable the Stamp's open baudmode, described below. For most situations, you can select a baudmode value from the table in the Stamp's manual. Listing 1 is a Visual-Basic function that accepts a series of settings and returns a baudmode value to match. The value is calculated like this:

Bits 0–12 equal the period of one bit in microseconds, minus 20. For example, 300 bps has a period of 3333 µsecs. Subtract 20 and the result is 3313. The top bit rate is 38,400 bps, though higher rates may require pacing between bytes, as described below.

Bit 13 is 0 for 8 bits, no parity, or 1 for 7 bits, Even parity.

Bit 14 is 0 for noninverted, 1 for inverted. When connecting two Stamps, both must have the same polarity. RS-232 uses inverted signals. When using RS-232, select inverted if the Stamp's interface doesn't use a MAX232 or other inverting driver, or non-inverted if it does.

Important tip: If you use inverted baudmode for RS-232 output on a pin other than *Sout*, write 1 to the bit on power up. This ensures that the bit will remain high (RS-232 idle state) until the Start bit of the first `SerIn` statement pulls it low. For example, if you're using bit 14 for serial output, place this statement near the beginning of the program:

```
    high 14
```

For `Serout`, bit 15 is 0 for normal operation, and 1 for open baudmode, which enables connecting multiple Stamps to one line. (Chapter 12 has more on open baudmode.) Bit 15 is unused by `Serin`.

`plabel` names a label to jump to on parity error.

`timeout` is the number of milliseconds to wait for incoming data on `SerIn`, or for *fpin* to indicate ready on `SerOut`. `Tlabel` is the label to jump to on timeout.

`pace` gives a delay in milliseconds between bytes. Pacing allows time for the receiving device to process each byte before reading the next. Stamps may require pacing at 9600 bps or higher.

Data Formats

The final parameters, `InputData` and `OutputData`, lists the value or values to write or read at the serial pins. Both of these allow a variety of modifiers that automatically process data, either before sending it or on receiving it. The modifiers can save processing time and storage space in the Stamp and make it easy to communicate with devices that require specific formats.

The `Dec` modifier converts `SerOut` data to the ASCII codes that represent the value's decimal digits. For example, an `OutputData` of `[Dec 5]` writes 53, the ASCII code for 5, to the port. An `OutputData` of `["5"]` accomplishes the same thing. Sending

```
Public Function fncGetBaudMode _
    (BitRate As Long, _
    Parity As Boolean, _
    Inverted As Boolean, _
    OpenBaudMode As Boolean) _
As Long
'Returns the Basic Stamp's baudmode parameter
'for the requested settings.
Dim BitRateValue As Long
Dim ParityValue As Long
Dim InvertedValue As Long
Dim OpenBaudModeValue As Long
BitRateValue = 1000000 \ BitRate - 20
If Parity = True Then
    ParityValue = 8192
Else
    ParityValue = 0
End If
If Inverted = True Then
    InvertedValue = 16384
Else
    InvertedValue = 0
End If
If OpenBaudMode = True Then
    OpenBaudModeValue = 32768
Else
    OpenBaudModeValue = 0
End If
fncGetBaudMode = _
    (BitRateValue + ParityValue + InvertedValue + _
    OpenBaudModeValue)
End Function
```

Listing 1: This Visual-Basic routine calculates the baudmode parameter for the Basic Stamp's Serin and Serout statements.

ASCII codes is useful if the receiver will display the values as ASCII text. Also, some networks send all data as ASCII text, reserving other values for node addressing, control codes, or other uses.

On the receive side, the `Dec` modifier causes `SerIn` to accept ASCII codes for characters 0–9 until receiving a code for a non-numeric character. `SerIn` then converts the codes to their numeric value, and stores the value in the indicated variable. This saves memory on the receiving end, because one byte can store a value represented by two or three characters. For example, input bytes of `[50, 53, 53, 10]` are the ASCII codes for 2, 5, 5, and linefeed. An `InputData` of `[Dec A]`, where `A` is a byte variable, would accept the first three codes, quit on seeing the non-numeric linefeed character, and store the value 255 in `A`.

For Visual-Basic and QuickBasic programmers, `SerOut`'s `Dec` is equivalent to `CStr()`, and `SerIn`'s `Dec` is equivalent to `Val()`.

The `Dec1–Dec5` modifiers wait for the designated number of digits, so there's no need to send a non-numeric character with each value. In another variation, `SDec` and `SDec1–SDec5` enable sending signed values, with "`-`" preceding the digits of a negative value.

In a similar way, `Hex`, `Hex1–Hex4` and `SHex`, `Shex1–SHex4` send and accept hex characters 0–9 and A–F (or a–f). The `IHex ISHex` modifiers require *$* preceding each value. The `Bin`, `SBin`, `IBin`, `ISBin` modifiers do the same for binary values, with binary characters *0* and *1* and a binary prefix of `%`.

Ensuring that the Stamp Sees Incoming Data

`SerIn` has two other modifiers: `Wait` and `Skip`, which can help to ensure that the Stamp sees the values intended for it while ignoring any others.

`Wait` causes the Stamp to ignore all data until it receives a value or values matching the `Wait` parameter(s). This can be useful in a network where a master node sends a node address followed by data intended for that node. If the receiving Stamps use the `Wait` modifier, they will automatically ignore bytes addressed to other nodes. Because a node will continue to wait until it recognizes its `Wait` modifiers, it won't miss any data intended for it.

However, there are limitations to using `Wait`. One is that the data following the node address can't include any bytes that equal another node's address. For example, if a master node sends a node address of 1 followed by a data byte of 7, when node 7 sees the data byte, it will think that master is addressing it. A way around this is to send all data with a `Dec`, `Hex`, or `Bin` modifier and send the node numbers as values (such as 0–7) that don't correspond to any ASCII codes used by these formats.

Another limitation to `Wait` is that nodes can waste a lot of time just watching for their address to come up. If using the timeout parameter, the timeout count quits on any received data, so `Wait` has little use as a data filter. Yet without a timeout, the node can't break out of a `SerIn` if the master fails to send the correct `Wait` modifier.

If a node is responsible for other activities, it may make more sense to have the program periodically read any incoming byte. Then if a byte doesn't match the node address, the program can move on instead of being stuck in a `Wait` statement.

Yet another modifier for `SerIn` is `Skip L`, which skips over *L* bytes of input. A network node might use this to send a series of bytes, with each byte intended for a different node. Node 1 can read the first byte; Node 2 can skip the first byte and read the next; Node 3 skips 2 bytes and reads the third, and so on.

The Stamp I also has `SerIn` and `SerOut` statements, which you can use with any port bits. These are similar to the Stamp II's statements, except that they don't support time-outs, parity, or pacing, and the maximum bit rate is 2400.

Hardware Handshaking

Another option for ensuring that the Stamp sees all data sent to it is to use the Stamp II's `fpin` parameter to designate a port bit for handshaking. `Serout`'s *fpin* may connect to *CTS*, while `Serin`'s *fpin* may connect to *RTS* on a remote PC. The PC transmits only when *CTS* is high, and the Stamp transmits only when *RTS* is high. The main drawback of

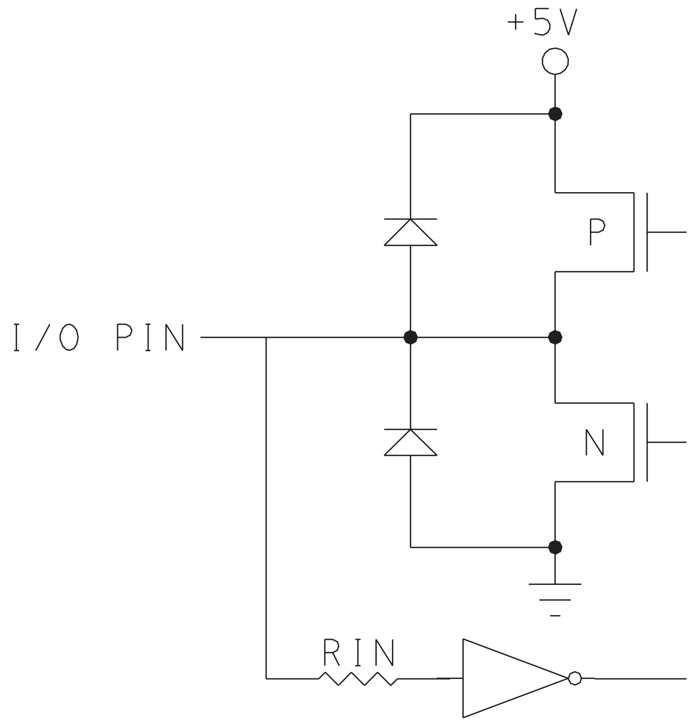


Figure 3: The circuits inside a Basic Stamp's I/O pin.

this approach is the need to use additional port bits, cable wires, and possibly drivers and receivers.

Signal Levels

If the cable is short (15 feet), the Stamp II's `Sin` and `Sout` can connect directly to an RS-232 port. Because the port outputs on both the Stamp I and II have strong drivers, and because the Stamp is capable of inverting the signals, a short link between the port bits and an RS-232 port will also work in most cases, with only a current-limiting resistor at the RS-232 input. Later chapters have more on Stamp interfaces.

Stamp-to-Stamp Links

Basic Stamps can also use their serial ports to communicate with each other. If the distance between Stamps is short (15 feet or so), there's no need for added drivers or receivers. Just connect a serial output on each Stamp to a serial input on the other, and add a ground wire if the Stamps don't share a power supply.

Open Baudmode

`Serout` includes an *open baudmode* option that makes it easy for two or more Stamps to communicate over a single wire. If you're familiar with open-collector or open-drain outputs, open baudmode works in a similar way.

Figure 3 shows the circuits behind a Stamp's I/O pin. The CMOS output has complementary NMOS and PMOS transistors. The output has two driven, or active, states. For a logic high output, the PMOS transistor switches on, creating a low-resistance path from the output pin to +5V. For a logic low output, the NMOS transistor switches on, creating a low-resistance path from the output to ground. An input buffer enables programs to read the logic level at the pin. The resistor and diodes protect the chip by limiting input voltages and currents.

When the pin is being used as an input, neither transistor is on. The output is high impedance and has no effect on the circuits it connects to. The input buffer reads the logic level of whatever circuits connect to the pin. Other terms to describe this state are *off, open,* or *tristated.*

In normal operation, a pin being used as an output switches between active high and active low. In a full-duplex serial link, the serial output connects to an input pin on another Stamp or other device.

Because each pin can act as an output or input, why not make a 1-wire link by connecting bits on two or more Stamps? In fact, this is possible, as long as only one bit at a time is configured as an output. If two connected outputs are enabled at the same time, and if one output is high and the other low, the result is a low impedance path from +5V to ground that causes the bits' output transistors to draw high currents.

To prevent this from happening, all Stamps in the link but one must execute either a `Serin` to receive data or an `Input` statement configure the bit as input. For example, one Stamp sends data with `Serout`. After sending the data, and before another Stamp executes a `Serout`, the transmitting Stamp must execute a `Serin` or `Input` statement to configure the bit as an input.

Open baudmode provides a simpler and safer way to do this. Instead of switching between active high and active low, the open baudmode outputs switch between open and a driven state. A pullup or pulldown resistor determines the state of the line when all outputs that connect to the line are open, and whether the line uses inverted or noninverted data. Figure 4 illustrates.

If `Serout` sends noninverted data, the line uses a pullup to +5V. If all of the outputs are logic 1s, all are open and the pullup brings the line high. Writing *0* to any output pulls the line low because the resistance between that output and GND is much less than the resistance between the pullup and +5V.

If `Serout` uses inverted data, the line uses a pulldown to ground. If all of the outputs are open, the pulldown brings the line low. Writing 0 to any output bit results in a logic high at the pin, which pulls the line high.

With either configuration, an idle (not transmitting) pin is open. As long as only one Stamp transmits at a time, the data will arrive without errors. The Stamps that aren't transmitting can use `Serin` statements to read data on the line.

Open baudmode is for use only when connecting Stamps directly, without buffers or drivers, using any of I/O bits 0–15, with short cables. All connected Stamps must also share a ground connection. This type of link should handle 8 or more Stamps.

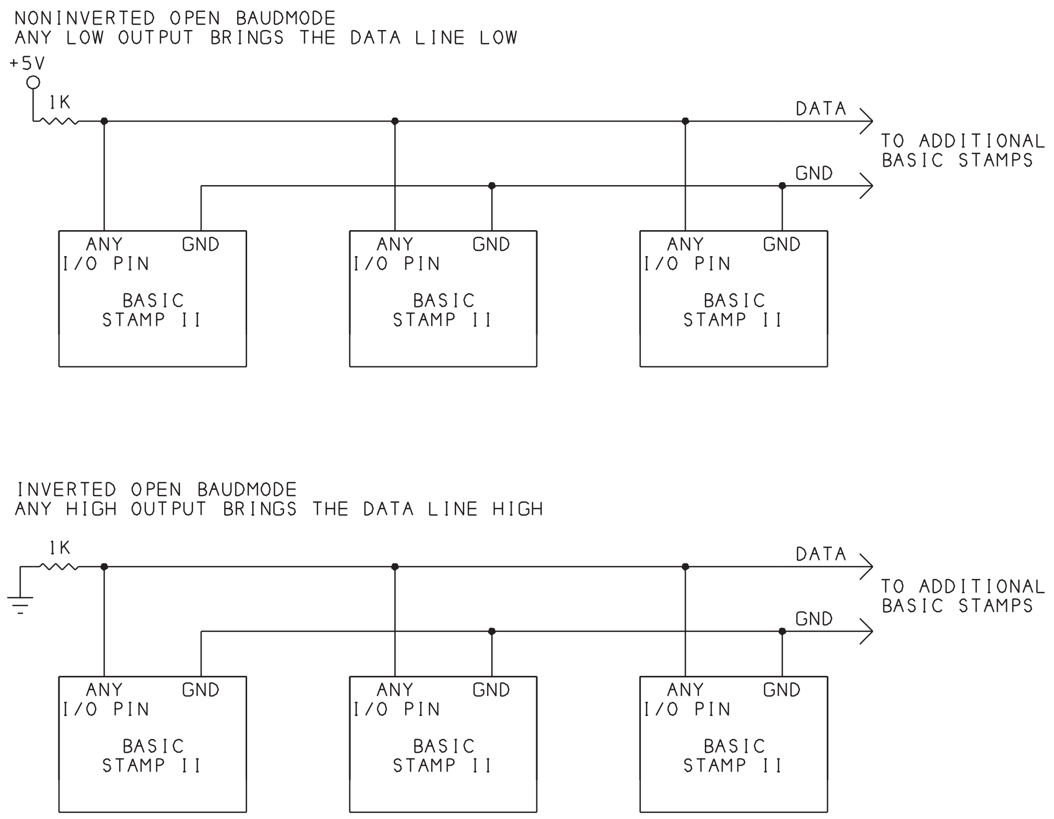

NONINVERTED OPEN BAUDMODE
ANY LOW OUTPUT BRINGS THE DATA LINE LOW
+5V
1K
DATA
GND
TO ADDITIONAL BASIC STAMPS
ANY I/O PIN
GND
BASIC STAMP II
INVERTED OPEN BAUDMODE
ANY HIGH OUTPUT BRINGS THE DATA LINE HIGH
1K
DATA
GND
TO ADDITIONAL BASIC STAMPS
ANY I/O PIN
GND
BASIC STAMP II

Figure 4: The Stamp II's open baudmode allows easy networking for short links.

Other Options for Basic Stamps

Besides the Basic Stamp II's dedicated serial interface, the Basic Stamp I and II can use any other port bits for serial communications. The bits have TTL-compatible inputs and outputs.

A short-range interface from a Stamp's port bit to an RS-232 port requires just one resistor, as Figure 5A shows. This is because the Stamp's output drivers are strong enough to drive a short RS-232 link directly, and the inputs have protection diodes that limit voltages to +0.6V greater than chip's the power supply. The resistor provides additional protection by limiting input current. The Stamp's baudmode parameter must use inverted signals with this interface.

For a true RS-232 interface, you can connect a MAX232 or similar to any of the I/O port bits. If you want to use the Stamp's *Sin* and *Sout* pins, it's a little more complicated because of the Stamp's additional circuits at these pins.

The Stamp's hardware inverts the voltages at *Sin* and *Sout*, and RS-232 interface chips also invert the signals. This is no problem when using *SerIn* and *Serout* statements, because you can set baudmode to invert the signals in firmware. But when using the *Sin* and *Sout* pins for programming and debugging, there's no way to specify inverted signals. A solution is to add inverters to reinvert the signals. You can use ordinary 5V inverters, or spare MAX232 gates for this, as Figure 5B shows.

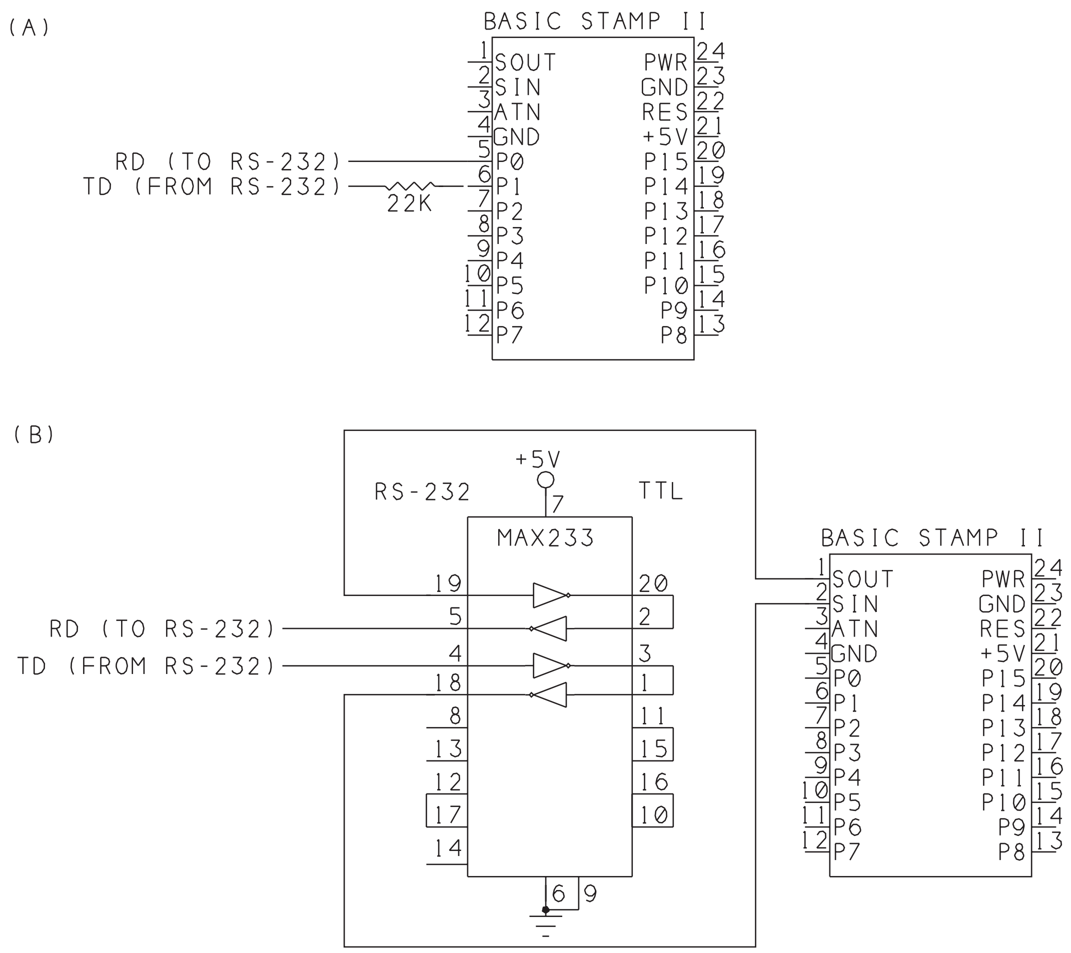


Figure 5: (A) For a short link to an RS-232 port, Basic Stamp I/O pins require only a series resistor at the input bit. (B) For a true RS-232 link at *Sin* and *Sout*, use a MAX233 or similar, with double inverters.

PC-to-Basic Stamp Link

Figure 6 shows a link between a PC and a Basic Stamp II. The link uses a true RS-232 interface to link two of the Stamp's port pins to a PC's serial port. A MAX233 converts to RS-232 voltages. This circuit leaves the Stamp's *Sin* and *Sout* pins unused, though you can connect them to another PC's serial port for development and debugging.

The Stamp communicates with the PC as in the previous example, exchanging blocks of eight bytes. Listing 2 is the Stamp's program code.

Exchanging Data

The Stamp's serial port has no input buffer. Because of this, the PC first sends a byte to get the Stamp's attention (in `fncGetStampsAttention`). Because the Stamp might be busy doing something else when the PC sends its byte, the PC sends the byte repeatedly until it either sees a response or decides that the Stamp isn't going to respond at all.

The PC adds a delay of at least one byte width between the bytes. This is to help the Stamp identify the Start bit. Otherwise, if `SerIn` begins reading in the middle of a byte, it will misread the value. The Stamp's `SerIn` statement skips the first byte it detects, in case it's a partial byte, and waits for the next Start bit. Figure 7's waveforms show a PC signaling a Stamp until the Stamp responds.

The Stamp alternates between watching for incoming serial data and carrying out its own duties. With each `Serin` statement, the Stamp spends an allotted time waiting for incoming data. When the Stamp detects a byte, it branches to a subroutine that sends a byte back to the PC to acknowledge that it has received the byte.

The Stamp then executes a `SerIn` statement and waits for the expected eight bytes. The PC sends the bytes, and the Stamp stores them and sends eight bytes back to the PC. The PC reads the received bytes and the communication is complete.

If the Stamp sees no data, it jumps to the beginning of its program.

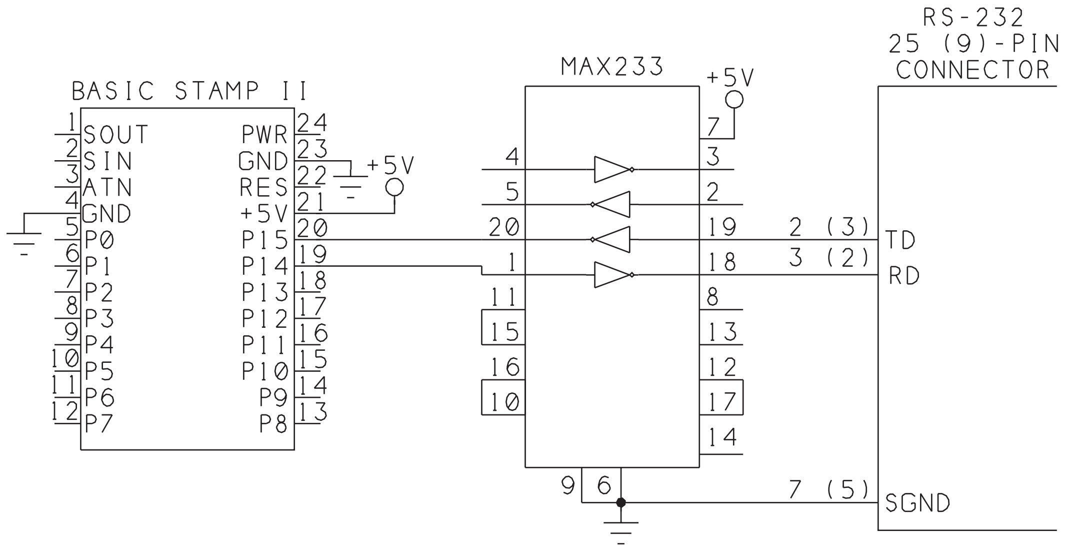


Figure 6: A link between a Basic Stamp and a PC.

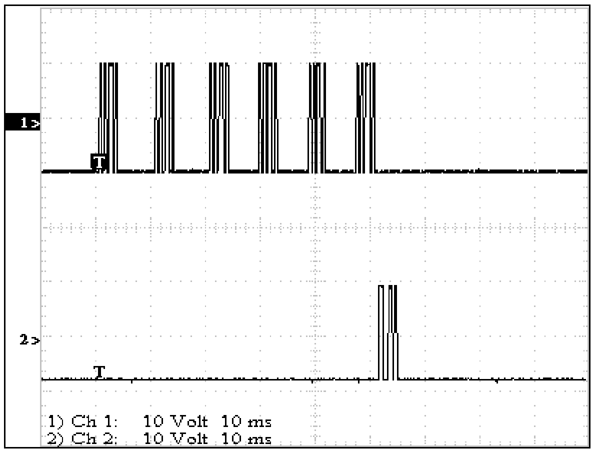


Figure 7: Trace 1 shows a byte sent repeatedly by a PC to a Basic Stamp. When the Stamp responds (trace 2), the PC stops sending.

```
'StampII RS-232 link to PC.
'The PC and Stamp exchange blocks of 8 bytes.
'All debug statements are for troubleshooting & may be removed.

'Variables:
DataIn var byte(8)
DataOut var byte(8)

Constants:
'Serial I/O bits:
'serial transmit output
SerialOutput con 14
'serial receive input
SerialInput con 15

Attention con $A5
Acknowledge con $A6

'Serial transmissions are at 2400 bps, noninverted, 8-N-1
BaudMode con 396

'Timeouts are 2 seconds.
TimeOut con 2000

'Bits 0-13 are undefined (free for any use).
'Bits 14-15 are for serial link:
dir14=1
dir15=0

'Default test data:
DataOut(0)="1"
DataOut(1)="2"
DataOut(2)="3"
DataOut(3)="4"
DataOut(4)="5"
DataOut(5)="6"
DataOut(6)="7"
DataOut(7)="8"
```

Listing 2: Basic Stamp II code for an RS-232 link. (Sheet 1 of 2)

```
'main program loop:
debug "Waiting to receive data...",cr
begin:
gosub NodeActivities
'Wait for a byte or timeout.
'Skip the first byte received. (It may be a partial byte.)
'If the expected byte is received, send a reply
serin serialinput,baudmode,timeout,Begin,[Skip 1, DataIn(0)]
debug "received ",dec DataIn(0),cr
if DataIn(0)=Attention then TransferData
goto begin
end

NoData:
debug "No data received.",cr
GoTo Begin

TransferData:
'Send acknowledge byte.
debug "sending ack",cr
serout SerialOutput,BaudMode, [Acknowledge]
'Wait to ensure PC has stopped sending Attention byte.
pause 100
'Read incoming bytes.
Serin SerialInput,baudmode,timeout,NoData,[DataIn(0), DataIn(1),
 DataIn(2), DataIn(3), DataIn(4), DataIn(5), DataIn(6), DataIn(7)
 ]
debug "Received: ",DataIn(0), DataIn(1), DataIn(2), DataIn(3),
 DataIn(4), DataIn(5), DataIn(6), DataIn(7),cr
'Send 8 bytes
serout SerialOutput,Baudmode,[DataOut(0), DataOut(1), DataOut(2),
 DataOut(3), DataOut(4), DataOut(5), DataOut(6), DataOut(7) ]
debug "Sending: ",DataOut(0), DataOut(1), DataOut(2), DataOut(3),
 DataOut(4), DataOut(5), DataOut(6), DataOut(7),cr
goto begin

NodeActivities:
'Use this routine for any activities the Stamp is responsible for
 on its own.
'Set one output byte to match port bits 0-7.
'DataOut(0)=InL
return
```

Listing 2: Basic Stamp II code for an RS-232 link. (Sheet 2 of 2)

In the example code, the initial bytes exchanged to get and acknowledge the Stamp's attention are specific values, but otherwise have no particular meaning. However, these bytes could contain data or commands as well.

Ensuring that the Stamp Sees Incoming Data

If the Stamp has little else to do except watch for incoming data, it will detect incoming bytes without problems. If the Stamp spends most of its time in other activities, the timing of the communications is more critical.

To guarantee that the Stamp will see a byte sent by the PC, two conditions must be met. First, the interval between bytes sent by the PC should be shorter than the Stamp's `Serin Timeout` value. And the total time that the PC spends trying to get the Stamp's attention should be longer than the interval between `SerIn` statements at the Stamp.

In the example code, the PC sends the byte continuously for up to two seconds, and the Stamp watches its serial port for one second at a time. The PC could delay up to one second between sending each pair of bytes, but sending the bytes continuously ensures the quickest response from the Stamp. The Stamp must execute `SerIn` at least once every two seconds, to ensure that the PC doesn't time out before the Stamp detects an incoming byte. If the PC sees no response after two seconds, it displays a message and moves on.

You can adjust the timing values as needed. For example, if the Stamp may delay as much as 5 seconds between `SerIn` statements, the PC's timeout limit should be greater than 5 seconds.

An RS-485 Network

The example RS-485 network presented below uses a master/slave protocol. The master node communicates with up to seven slave nodes. Each node connects to the network via an RS-485 interface. The master is a personal computer running Visual Basic, and the slaves may be other PCs or Basic Stamps. Each slave detects, reads, and responds to messages directed to it.

You can use the network as a base for designing your own projects using personal computers or microcontrollers of any type, in any combination, and any programming language.

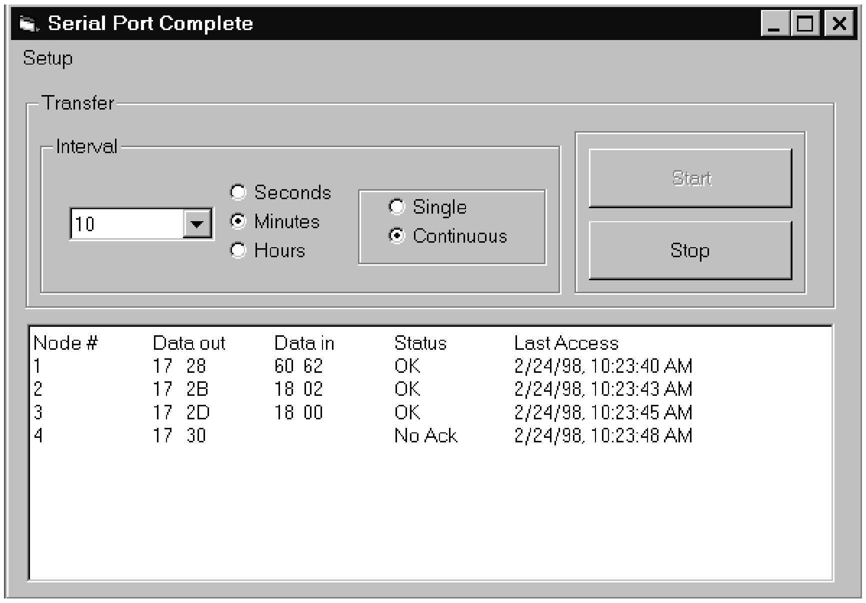
The Protocol

The network uses a communications protocol designed for simplicity and reliability in exchanging short messages. A master polls each slave in turn.

Byte Definitions

The network uses the following conventions:

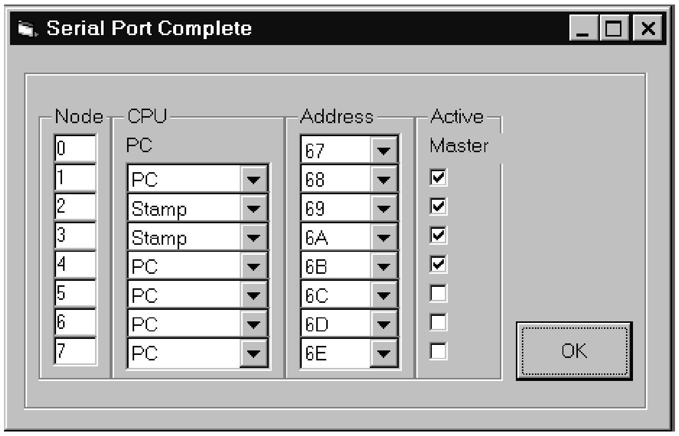
- All data is sent in ASCII Hex format, using the characters *0* through *9* (30h through 39h) and *A* through *F* (41h through 46h). A pair of ASCII Hex bytes represents one binary value.
- Node addresses use ASCII codes for the characters *g* to *n* (67h through 6Eh).
- Messages are a defined length. Each message contains four ASCII Hex data bytes.



Interval: 10 ( ) Seconds [✓ Minutes] ( ) Hours; ( ) Single [✓ Continuous]

```
Node #    Data out    Data in    Status    Last Access
1         17  28      60  62     OK        2/24/98, 10:23:40 AM
2         17  2B      18  02     OK        2/24/98, 10:23:43 AM
3         17  2D      18  00     OK        2/24/98, 10:23:45 AM
4         17  30                 No Ack    2/24/98, 10:23:48 AM
```

Figure 8: The main window for the RS-485 network enables users to select an interval for polling the network, to start and stop polling, and to view the latest results.



Node	CPU	Address	Active
0	PC	67	Master
1	PC	68	✓
2	Stamp	69	✓
3	Stamp	6A	✓
4	PC	6B	✓
5	PC	6C	☐
6	PC	6D	☐
7	PC	6E	☐

Figure 9: The Nodes window enables users to select an address and CPU type for each node, and to indicate whether or not the node is in use.

Polling the Nodes

Figure 8 shows the main window for the program. It uses Chapter 4's template file as a base. The user can do a single poll of the network, or select an interval for continuous polling. A Rich Text box displays the results, including the nodes polled, data sent and received, and time of the last transfer.

The Setup menu includes three selections. The Port Settings form allows selecting a port and bit rate. The Data File form allows saving of transferred data in a file. The Nodes form (Figure 9) displays the eight nodes by number, CPU type, address, and whether or not the node is active (currently in use). All of these are user-selectable, except the master's node type and Active check box.

The master polls each slave in turn, using the following procedure:

- The master sends a slave address, disables its transmitter and waits for a reply.
- All slaves read the address and compare it to their own address.
- The slave that recognizes its address responds by enabling its transmitter, sending its address back to the master as an acknowledgment, then disabling its transmitter and waiting for data.
- When the master receives the acknowledgment, it knows that the slave is waiting for a message. The master sends the message, consisting of four ASCII Hex bytes. It then disables its transmitter and waits for a reply.
- When the slave receives the message, it stores the data, enables its transmitter, and sends five bytes back to the master: its address, plus four ASCII Hex bytes. It then disables its transmitter and takes any action requested by the master's message.
- The master stores the reply from the slave, re-enables its transmitter, and is then free to poll another slave.

The other nodes may read the addresses and messages, but take no action because none of the bytes match their addresses.

Messages

The data bytes in the messages can represent anything at all. The master may send a command to tell the slave to read a sensor, followed by an ASCII Hex value that indicates which sensor to read. Or the master may send a a byte that selects a motor to control, and another that selects a speed and direction for the motor. In reply, a node may send data about switch states, sensor readings, alarm conditions, time and date information, or any digital outputs or logic levels. The slaves also transmit their address with each message to assure the master that the correct node is replying.

Customizing

You can change and adapt this format to fit a particular use. For example, the link may transfer more or fewer data bytes, and it may allow more or fewer nodes.

Table 2 shows the defined uses for each of the 256 byte values. Many values are undefined, and you can assign meanings to these and use them as needed. For example, F0h through FFh may each indicate a command.

The Link

Figure 10 is shows an example of wiring you can use for this network. This is just one possibility. The network may use any appropriate components for a half-duplex link, including high-speed, low-speed, low-voltage, short-distance, and isolated components. A PC may also use an expansion card with an RS-485 interface, allowing higher speeds and eliminating the need to convert RS-232.

Each node uses a 75176B RS-485 transceiver to interface to the network. At the master, the receiver is always enabled, and *RTS* controls the driver enable. At the slaves, the driver and receiver enables are tied together and controlled with one port bit.

Table 2: Uses for the 256 byte values

Range (hex)	Characters	Use
30 -39, 41-46	0-9, A-F	ASCII Hex characters
67-6E	g-n	Node addresses
00-2F, 3A-40, 47-60, 6F-FF	Various	Unused

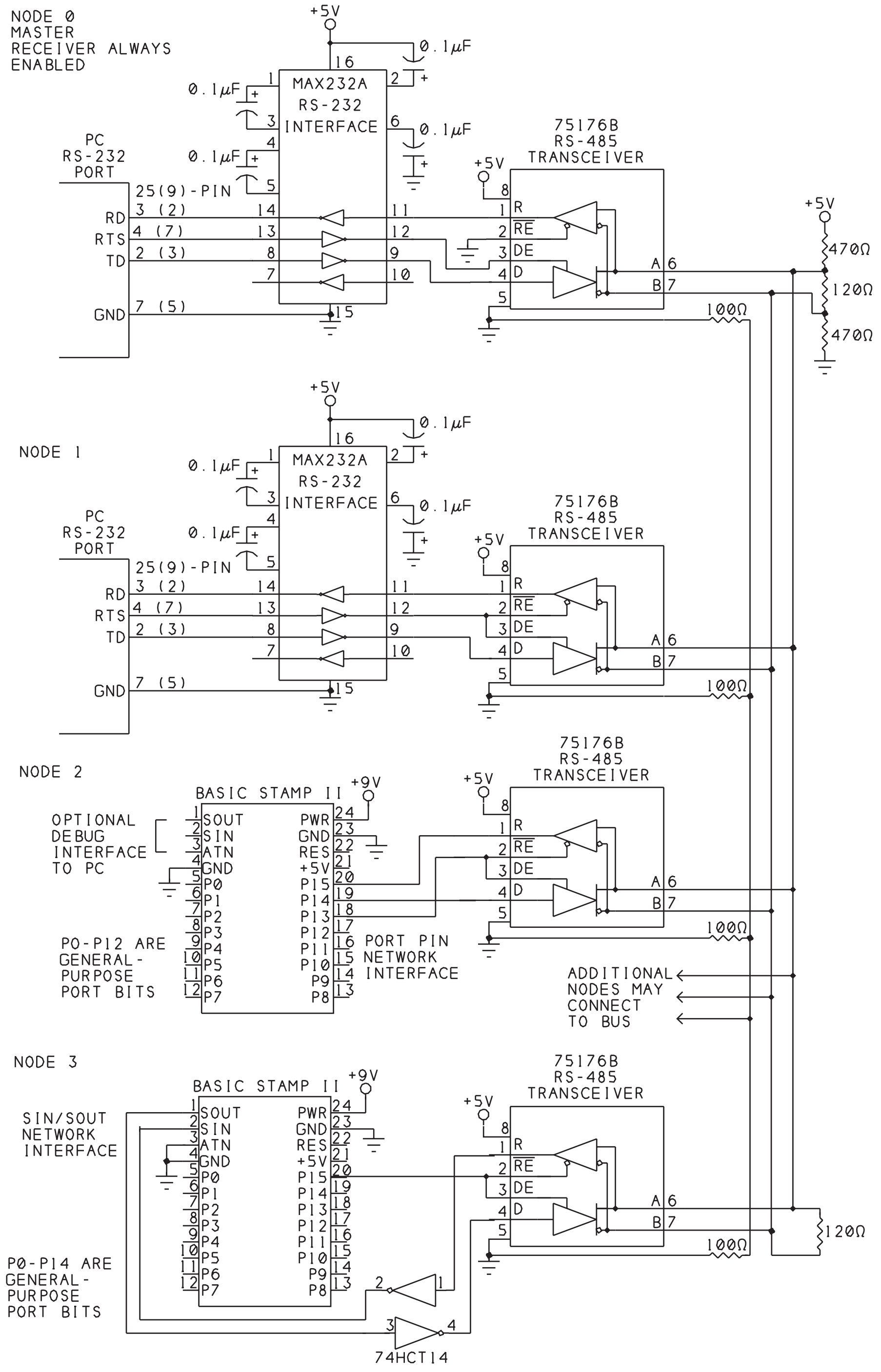


Figure 10: Example wiring you can use for the RS-485 network.

At the master node, a MAX232A converts the PC's RS-232 voltages to TTL voltages. The MAX232A also re-inverts the signals: at its TTL inputs and outputs, a high voltage represents a logic 1 and a low voltage represents a logic 0. The TTL voltages interface to the 75176 transceiver.

At the PC, the *RTS* line determines the direction of the 75176. For the microcontrollers, any output port pin can perform this function.

The first and last nodes on the bus each have a 120-ohm termination resistor. Two 470-ohm biasing resistors at one end hold the inputs high when no drivers are enabled on the network. If your link is short line as defined in Chapter 10, you don't need the 120-ohm terminations or the 470-ohm biasing resistors.

The serial cable to the PC has to have at least four wires. A standard 9-wire cable will have the needed connections. Master and slave PC nodes use identical circuits except that the slave's driver and receiver enable lines are tied together. In the example slave software, the slaves don't read back their transmissions.

For a microcontroller link, I chose Basic Stamps, mainly for their ease of use for debugging, because they can run the network program while maintaining communications with another host PC.

The Basic Stamp connects directly to its 75176 transceiver. You can use any of the Stamp's 16 port bits for the serial link. Node 2 in the schematic shows bit 15 as the receive input, bit 14 as the transmit output, and bit 13 as the direction-control output. This leaves 13 bits for other uses.

If you want to reserve two more bits for other uses, use Node 4's wiring for the Stamps. This configuration uses *Sin* and *Sout* for the network interface, and uses only one of the 16 port bits, for direction control. This wiring prevents the use of *Sin* and *Sout* for debugging with the Stamp's editor software on a PC. The simplest approach is to use port pins for developing and debugging. When all is working, if desired, change the pin assignments for *Sin* and *Sout*, load the program into the Stamp, then power down, disconnect the PC, connect the RS-485 transceiver, and power back up.

Sin and *Sout* use external inverters to reinvert the signals so that A > B on the RS-485 line corresponds to the idle state. This is necessary because *Sout* gets its logic-low (or negative) voltage from *Sin*. *Sin* must be low when the link is idle. But on the RS-485 link, the idle state is A > B, which results in a logic high at pin 1 of the '176. The inverter converts this to the required logic low. The second inverter just ensures that both signals use the same polarity.

For the network wires, unshielded twisted pair works well. Include the ground wire to each node unless you're sure that all nodes already have a common signal-ground connection.

The Master's Programming

Listing 3 is Visual-Basic program code for the master node. Many parts of the code are similar or identical to Chapter 8's 2-device link. Both applications exchange data with remote computers, but this time there are multiple nodes to deal with.

```
Option Explicit
'A master node communicates with up to 7 slave nodes
'over a half-duplex RS-485 interface.
'Each node has an address.
'Each message consists of the receiver's address, followed by
'4 ASCII Hex bytes representing 2 binary values.
'Each reply consists of the sender's address,
'followed by 4 ASCII Hex bytes representing 2 binary values.
Option Base 0
'Delay (milliseconds) to ensure RTS has toggled (Windows delay):
Const RTSDelay = 200
'Delay (milliseconds) before enabling transmitter,
'to allow the slave to disable its transmitter.
Const EnableDelay = 500
'Delay (milliseconds) to wait for a reply from a slave.
Const ReplyDelay = 3000
'Node 0 is the master; other nodes are slaves.
Const HighestNodeNumber = 7
'With each message, the master sends and receives
'4 ASCII Hex bytes.
Const NumberOfDataBytesOut = 4
Const NumberOfDataBytesIn = 4

Private Type typNodes
    Address(0 To HighestNodeNumber) As Byte
    DataOut1(0 To HighestNodeNumber) As Byte
    DataOut2(0 To HighestNodeNumber) As Byte
    DataIn1(0 To HighestNodeNumber) As Byte
    DataIn2(0 To HighestNodeNumber) As Byte
    Status(0 To HighestNodeNumber) As String
    Cpu(0 To HighestNodeNumber) As String
    Active(0 To HighestNodeNumber) As Integer
    LastAccess(0 To HighestNodeNumber) As String
End Type
Private Type typDataTransferFormat
    SingleOrContinuous As String
    IntervalUnits As String
    IntervalValue As Single
End Type
```

```
Dim SelectedNode As Integer
Dim PollInterval As Integer
Dim DataOut(NumberOfDataBytesOut - 1) As Byte
Dim DataIn(NumberOfDataBytesIn - 1) As Byte
Dim DataTransferFormat As typDataTransferFormat
Dim PreviousTime As Date
Dim TimeOfTransfer As String
Dim TransferInProgress As Boolean

Dim Nodes As typNodes
```

Listing 3: Code for the RS-485 network’s main window. (Sheet 1 of 13)

```
Private Function fncConfirmTransmittedData _
    (Buffer As Variant) _
    As Integer
'Ensure that all data has transmitted by reading it back.
'Receiver must be enabled!
'Returned values:
'-1 = Data read back successfully
'0 = Data didn't match
'1 = Timeout
Dim DataReadBack As String
'Estimate the time to transmit the data:
tmrTimeout.Interval = OneByteDelay * LenB(Buffer) + 500
tmrTimeout.Enabled = True
TimedOut = False
Do
    DoEvents
Loop Until MSComm1.InBufferCount >= Len(Buffer) Or TimedOut = True
DataReadBack = MSComm1.Input
If StrComp(DataReadBack, Buffer, vbBinaryCompare) = 0 Then
    fncConfirmTransmittedData = -1
Else
    If TimedOut = False Then
        fncConfirmTransmittedData = 0
    Else
        fncConfirmTransmittedData = 1
    End If
End If
tmrTimeout.Enabled = False
TimedOut = False
End Function
```

```
Private Function fncCreateMessage _
    (NodeNumber As Integer) _
    As String
'A message consists of four bytes in ASCII Hex format.
'Each ASCII Hex pair represents the value of a byte.
Dim MessageLength As Integer
Dim MessageToSend As String
    MessageLength = NumberOfDataBytesOut - 1
    Call GetDataToSend(NodeNumber)
    'Create the message, consisting of
    '4 bytes that contain the 2 data bytes in ASCII Hex format.
    'Each byte represents 1 hex digit (4 bits).
    'Convert the 2 data bytes to ASCII Hex
    'and store in the Message string.
    MessageToSend = _
        fncByteToAsciiHex(Nodes.DataOut1(NodeNumber)) & _
        fncByteToAsciiHex(Nodes.DataOut2(NodeNumber))
    fncCreateMessage = MessageToSend
End Function
```

Listing 3: Code for the RS-485 network's main window. (Sheet 2 of 13)

```
Private Function fncDisplayDateAndTime() As String
'Date and time formatting.
fncDisplayDateAndTime = _
    CStr(Format(Date, "General Date")) & ", " & _
        (Format(Time, "Long Time"))
End Function
```

```
Private Function fncWaitForAck(NodeNumber As Integer) As Boolean
'End on receiving Acknowledge from the slave or timeout.
Dim Ack As Boolean
Dim NodeAddress As String
Dim ReceivedData As String
'The Acknowledge is the node address.
NodeAddress = Chr(Nodes.Address(NodeNumber))
Ack = False
tmrTimeout.Interval = ReplyDelay
'Disable the transmitter until Ack is received or timeout.
Call DisableTransmitter
'Wait for Acknowledge.
Do
    tmrTimeout.Enabled = True
    TimedOut = False
    Do
        DoEvents
    Loop Until (MSComm1.InBufferCount >= 1) Or (TimedOut = True)
    If TimedOut = False Then
        tmrTimeout.Enabled = False
        'Read the byte & compare to what was sent.
        ReceivedData = MSComm1.Input
        If StrComp _
            (ReceivedData, NodeAddress, vbBinaryCompare) = 0 Then
            Ack = True
            Nodes.DataIn1(NodeNumber) = Asc(ReceivedData)
        Else
            'if the Ack doesn't match the node address:
            Ack = False
            Call SaveResults(NodeNumber, 0, 0, "Ack Error")
         End If
    Else
        Ack = False
        Call SaveResults(NodeNumber, 0, 0, "No Ack")
    End If
Loop Until Ack = True Or TimedOut = True
tmrTimeout.Enabled = False
fncWaitForAck = Ack
TimedOut = False
Call EnableTransmitter(EnableDelay)
End Function
```

Listing 3: Code for the RS-485 network's main window. (Sheet 3 of 13)

```
Private Function fncWaitForReply _
    (NodeNumber As Integer) _
    As Boolean
'From the slave, read the node address & 4 ASCII Hex bytes.
Dim Ack As Boolean
Dim Reply As Boolean
Dim ReceivedData As String
Ack = False
Reply = False
TimedOut = False
tmrTimeout.Interval = ReplyDelay
'Disable the transmitter until bytes are received or timeout.
Call DisableTransmitter
tmrTimeout.Enabled = True
Do
    'Wait for reply
    TimedOut = False
    Do
        DoEvents
    Loop Until (MSComm1.InBufferCount > 4) Or (TimedOut = True)
    If TimedOut = False Then
        tmrTimeout.Enabled = False
        ReceivedData = MSComm1.Input
        Reply = True
        If StrComp(Asc(Left(ReceivedData, 1)), _
            Nodes.Address(NodeNumber), vbBinaryCompare) = 0 Then
            'If the first byte equals the slave's address,
           'get the numeric value of each pair of ASCII Hex bytes.
            Call SaveResults _
                (NodeNumber, _
                Val("&h" & Mid(ReceivedData, 2, 2)), _
                 Val("&h" & Mid(ReceivedData, 4, 2)), _
                 "OK")
        Else
            'If the first byte doesn't equal the node address:
             Call SaveResults(NodeNumber, 0, 0, "Data Error")
        End If
     Else
        'If the wait for a reply times out:
        Call SaveResults(NodeNumber, 0, 0, "Reply Timeout")
     End If
Loop Until Reply = True Or TimedOut = True
tmrTimeout.Enabled = False
Call EnableTransmitter(EnableDelay)
fncWaitForReply = Reply
End Function
```

Listing 3: Code for the RS-485 network's main window. (Sheet 4 of 13)

```
Private Sub cboIntervalValue_Click()
'Store the selected interval for data transfers.
DataTransferFormat.IntervalValue = Val(cboIntervalValue.Text)
'With shorter intervals, check elapsed time more often.
Select Case DataTransferFormat.IntervalUnits
    Case "seconds"
        tmrTransferInterval.Interval = 100
    Case "minutes", "hours"
        tmrTransferInterval.Interval = 1000
End Select
End Sub
```

```
Private Sub cmdStart_Click()
'Initiate data transfer.
Select Case DataTransferFormat.SingleOrContinuous
    Case "single"
        'Transfer data once.
        'Disable the Start button until polling is finished.
        cmdStart.Enabled = False
        Call PollSlave
        cmdStart.Enabled = True
    Case "continuous"
        'Do one transfer immediately, then let the timer take over.
        cmdStart.Enabled = False
        cmdStop.Enabled = True
        cmdStop.SetFocus
        PreviousTime = Now
        tmrTransferInterval.Enabled = True
        Call PollSlave
    Case Else
End Select
End Sub
```

```
Private Sub cmdStop_Click()
'Stop transferring data.
tmrTransferInterval.Enabled = False
cmdStop.Enabled = False
cmdStart.Enabled = True
Call DisableTransmitter
End Sub
```

```
Private Sub DisableTransmitter()
'Set RTS true (high) to disable the RS485 transmitter
'by bringing its chip-enable low.
'Assumes that a second RS-232 receiver inverts RTS.
MSComm1.RTSEnable = True
End Sub
```

Listing 3: Code for the RS-485 network’s main window. (Sheet 5 of 13)

```
Private Sub EnableTransmitter(EnableDelay As Single)
'Set RTS false (low) to enable the RS485 transmitter.
'Assumes that a second RS-232 receiver has inverted RTS.
'Delay in milliseconds allows remote node
'to disable its transmitter.
Call Delay(EnableDelay)
MSComm1.RTSEnable = False
'Windows delay:
Call Delay(RTSDelay)
End Sub
```

```
Private Sub Form_Load()
Show
Call GetSettings
Call Startup
Load frmPortSettings
Load frmNodes
TransferInProgress = False
tmrTimeout.Interval = ReplyDelay
tmrTransferInterval.Enabled = False
tmrTimeout.Enabled = False
TimedOut = False
Call InitializeDisplayElements
SaveDataInFile = False
Call InitializeNodes
Call GetNewNodeSettings
'The master's transmitter is enabled,
'except when receiving replies.
Call EnableTransmitter(0)
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
Call ShutDown
Unload frmNodes
Unload frmDataFile
Unload frmPortSettings
Close #2
End
End Sub
```

```
Private Sub GetDataToSend(NodeNumber As Integer)
'Dummy data for testing: the current hour and minute.
Dim CurrentTime As String
CurrentTime = CStr(Format(Time, "nnss"))
Nodes.DataOut1(NodeNumber) = Val(Left(CurrentTime, 2))
Nodes.DataOut2(NodeNumber) = Val(Right(CurrentTime, 2))
End Sub
```

Listing 3: Code for the RS-485 network's main window. (Sheet 6 of 13)

```
Public Sub GetNewNodeSettings()
'Store user changes made on the Nodes form.
Dim Count As Integer
    Nodes.Address(0) = CInt("&h" & frmNodes.cboAddress(0).Text)
For Count = 1 To 7
    Nodes.Cpu(Count) = frmNodes.cboCPU(Count).Text
    Nodes.Address(Count) = _
        CInt("&h" & frmNodes.cboAddress(Count).Text)
    Nodes.Active(Count) = frmNodes.chkNodeActive(Count).Value
Next Count
End Sub
```

```
Private Sub InitializeDisplayElements()
optSingleOrContinuous(0).Value = True
optIntervalUnits(0).Value = True
cboIntervalValue.ListIndex = 0
rtxStatus.Locked = True
rtxStatus.Text = ""
DataTransferFormat.IntervalValue = 1
cmdStop.Enabled = False
End Sub
```

```
Private Sub InitializeNodes()
Dim Count As Integer
For Count = 0 To HighestNodeNumber
    Nodes.DataIn1(Count) = 0
    Nodes.DataIn2(Count) = 0
    Nodes.Status(Count) = ""
    Nodes.LastAccess(Count) = ""
    Nodes.Cpu(Count) = ""
Next Count
Call UpdateDisplay
End Sub
```

```
Private Sub mnuDataFile_Click(Index As Integer)
frmDataFile.Show
End Sub
```

```
Private Sub mnuNodes_Click(Index As Integer)
frmNodes.Show
End Sub
```

```
Private Sub mnuPortSettings_Click(Index As Integer)
frmPortSettings.Show
End Sub
```

Listing 3: Code for the RS-485 network's main window. (Sheet 7 of 13)

```
Private Sub optIntervalUnits_Click(Index As Integer)
'Set the interval combo box to match the units selected.
Dim Maximum As Integer
Dim Count As Integer
Select Case Index
    Case 0
        Maximum = 59
        DataTransferFormat.IntervalUnits = "seconds"
    Case 1
        Maximum = 59
        DataTransferFormat.IntervalUnits = "minutes"
    Case 2
        Maximum = 24
        DataTransferFormat.IntervalUnits = "hours"
End Select
cboIntervalValue.Clear
For Count = 1 To Maximum
    cboIntervalValue.AddItem CStr(Count)
Next Count
cboIntervalValue.ListIndex = 0
End Sub
```

```
Private Sub optPollUnits_Click(Index As Integer)
'Set the combo box items to match the units selected.
Dim Maximum As Integer
Dim Count as Integer
Select Case Index
    Case 0, 1
        'seconds, minutes
        Maximum = 59
    Case 2
        'hours
        Maximum = 24
End Select
End Sub
```

```
Private Sub optSingleOrContinuous_Click(Index As Integer)
Select Case Index
    Case 0
        DataTransferFormat.SingleOrContinuous = "single"
        'Disable interval selection:
        optIntervalUnits(0).Enabled = False
        optIntervalUnits(1).Enabled = False
        optIntervalUnits(2).Enabled = False
    Case 1
        DataTransferFormat.SingleOrContinuous = "continuous"
        'Enable interval selection:
        optIntervalUnits(0).Enabled = True
        optIntervalUnits(1).Enabled = True
        optIntervalUnits(2).Enabled = True
End Select
End Sub
```

Listing 3: Code for the RS-485 network's main window. (Sheet 8 of 13)

```
Private Sub PollSlave()
'Send the node address & wait for Acknowledge.
'If Ack received, send data, wait for reply.
'Store the results.
Dim AckReceived As Boolean
Dim AttemptNumber As Integer
Dim Buffer As Variant
Dim Count As Integer
Dim LastNode As Integer
Dim MessageToSend As Variant
Dim NumberOfTries As Integer
Dim ReplyReceived As Boolean
Dim TransmitFinished As Boolean
TransferInProgress = True
For Count = 1 To HighestNodeNumber
'Skip the node if it isn't selected (Active) on the Nodes form.
    If Nodes.Active(Count) = 1 Then
        'Clear the transmit and receive buffers
        MSComm1.OutBufferCount = 0
        If MSComm1.InBufferCount > 0 Then
            Buffer = MSComm1.Input
        EndIf
        'Create the message from the stored values.
        MessageToSend = fncCreateMessage(Count)
        'Store the time of the poll.
        Nodes.LastAccess(Count) = fncDisplayDateAndTime
        'Send the node address as a text character.
        Buffer = Chr(Nodes.Address(Count))
        'For Stamp and other slaves without input buffers,
        'poll more than once if needed.
        Select Case Nodes.Cpu(Count)
            Case "PC"
                NumberOfTries = 1
            Case "Stamp"
                NumberOfTries = 2
        End Select
        AttemptNumber = 0
```

Listing 3: Code for the RS-485 network's main window. (Sheet 9 of 13)

```
        Do
            MSComm1.Output = Buffer
            'Wait for the data to transmit
            Select Case fncConfirmTransmittedData(Buffer)
                Case -1
                    'If success, wait for Acknowledge.
                    AckReceived = fncWaitForAck(Count)
                Case 0
                    Nodes.Status(Count) = "Transmit error"
                Case 1
                    Nodes.Status(Count) = "Ack Timeout"
            End Select
            AttemptNumber = AttemptNumber + 1
        Loop Until AckReceived = True Or _
            AttemptNumber = NumberOfTries
        If AckReceived = True Then
            MSComm1.Output = MessageToSend
            'Delay to let the data transmit
            Select Case fncConfirmTransmittedData(MessageToSend)
                Case -1
                    'Data has transmitted.
                    'Wait for the slave's reply.
                    ReplyReceived = fncWaitForReply(Count)
                Case Else
                    Nodes.Status(Count) = "Transmit error"
            End Select
        End If
        Call UpdateDisplay
    End If
Next Count
If SaveDataInFile = True Then
    Call WriteResultsToFile
End If
TransferInProgress = False
End Sub
```

```
Private Sub SaveResults _
    (NodeNumber As Integer, _
    Data1 As Byte, _
    Data2 As Byte, _
    ResultStatus As String)
    Nodes.DataIn1(NodeNumber) = Data1
    Nodes.DataIn2(NodeNumber) = Data2
    Nodes.Status(NodeNumber) = ResultStatus
End Sub
```

Listing 3: Code for the RS-485 network's main window. (Sheet 10 of 13)

```
Private Sub WriteResultsToFile()
'Save received data and time in a file.
Dim Count As Integer
For Count = 1 To HighestNodeNumber
    'Skip if the node isn't selected (active) on the Nodes form.
    If Nodes.Active(Count) = 1 Then
        Write #2, _
            Count, _
            Nodes.LastAccess(Count), _
            Nodes.DataOut1(Count), _
            Nodes.DataOut2(Count), _
            Nodes.DataIn1(Count), _
            Nodes.DataIn2(Count), _
            Nodes.Status(Count)
    End If
Next Count
End Sub
```

```
Private Sub tmrTransferInterval_Timer()
'See if it's time to do a transfer.
Dim CurrentTime As Date
Dim Units As String
CurrentTime = Now
Select Case DataTransferFormat.IntervalUnits
    Case "seconds"
        Units = "s"
    Case "minutes"
        Units = "n"
    Case "hours"
        Units = "h"
End Select
'If elapsed time since the last transfer is more than
'the selected interval, do a data transfer.
If DateDiff(Units, PreviousTime, CurrentTime) >= _
        DataTransferFormat.IntervalValue Then
    PreviousTime = CurrentTime
    'But don't start a new transfer if one is in progress.
    If TransferInProgress = False Then
        Call PollSlave
    End If
End If
End Sub
```

```
Private Sub tmrTimeout_Timer()
tmrTimeout.Enabled = False
TimedOut = True
End Sub
```

Listing 3: Code for the RS-485 network’s main window. (Sheet 11 of 13)

```
Private Sub UpdateDisplay()
'Show the latest information for all nodes
Dim Column As Integer
Dim DataIn1Display As String
Dim DataIn2Display As String
Dim Count As Integer
'Set up 5 columns
With rtxStatus
        .SelTabCount = 5
        For Column = 0 To .SelTabCount - 1
            .SelTabs(Column) = 1000 * Column
        Next Column
End With
rtxStatus.Text = "Node #" & Chr(vbKeyTab) _
    & "Data out" & Chr(vbKeyTab) _
    & "Data in" & Chr(vbKeyTab) _
    & "Status" & Chr(vbKeyTab) _
    & "Last Access" & vbCrLf
For Count = 1 To HighestNodeNumber
    'Skip if the node isn't selected (active) on the Nodes form.
    If Nodes.Active(Count) = 1 Then
    Select Case Nodes.Status(Count)
           Case "OK"
                DataIn1Display = _
                    fncByteToAsciiHex(Nodes.DataIn1(Count))
                DataIn2Display = _
                    fncByteToAsciiHex(Nodes.DataIn2(Count))
           Case Else
                DataIn1Display = ""
                DataIn2Display = ""
        End Select
        rtxStatus.SelStart = Len(rtxStatus.Text)
        rtxStatus.SelText = _
             Hex$(Count) & Chr(vbKeyTab) _
            & fncByteToAsciiHex(Nodes.DataOut1(Count)) & "    " _
            & fncByteToAsciiHex(Nodes.DataOut2(Count)) _
            & Chr(vbKeyTab) _
            & DataIn1Display & "   " & DataIn2Display _
            & Chr(vbKeyTab) _
            & Nodes.Status(Count) & Chr(vbKeyTab) _
            & Nodes.LastAccess(Count) & vbCrLf
    End If
Next Count
End Sub
```

Listing 3: Code for the RS-485 network's main window. (Sheet 12 of 13)

```
Public Function fncInitializeComPort _
    (BitRate As Long, PortNumber As Integer) As Boolean
'BitRate and PortNumber are passed to this routine.
'All other properties are set explicitly in the code.
Dim ComSettings As String
If MSComm1.PortOpen = True Then
    MSComm1.PortOpen = False
End If
ComSettings = CStr(BitRate) & ",N,8,1"
MSComm1.CommPort = PortNumber
' bit rate, no parity, 8 data, and 1 stop bit.
MSComm1.Settings = ComSettings
'Set to 0 to read entire buffer on Input
MSComm1.InputLen = 0
MSComm1.InBufferSize = 256
'Input and output data are text.
MSComm1.InputMode = comInputModeText
'MSComm does no handshaking.
MSComm1.Handshaking = comNone
MSComm1.OutBufferSize = 256
MSComm1.EOFEnable = False
'No OnComm event on received data.
MSComm1.RThreshold = 0
'No OnComm transmit event.
MSComm1.SThreshold = 0
MSComm1.PortOpen = True
OneByteDelay = fncOneByteDelay(BitRate)
End Function
```

Listing 3: Code for the RS-485 network's main window. (Sheet 13 of 13)

The master reads back all data it sends. When the transmitted data has been read back, the master knows that it's safe to disable its transmitter. Reading back the data also provides error-checking. If another node has mistakenly enabled its transmitter, or if Windows' timeout delay expires, the master's data won't transmit and won't be read back. After receiving data, the master delays before re-enabling its transmitter, to give the sending node time to disable its transmitter.

Selecting Nodes

Listing 4 is the code for the Nodes window that enables users to select which nodes are active, and what type of CPU the node contains.

Slave Programming

Listing 5 is the program stored in each Basic Stamp when using port bits for serial communications. Each node must have a unique address. The program disk also includes Stamp code for using *Sin* and *Sout* for communications and QuickBasic code for a slave node on a DOS PC.

```
Option Explicit
'Enables the user to specify the type & address of remote nodes.

Private Sub cmdOK_Click()
Call frmMain.GetNewNodeSettings
Hide
End Sub

Private Sub Form_Load()
Call InitializeNodeCpuComboBoxes
Call InitializeNodeAddressComboBoxes
Call InitializeNodeActiveCheckBoxes
Call GetSettings
Call frmMain.GetNewNodeSettings
End Sub

Private Sub Form_Unload(Cancel As Integer)
Call SaveSettings
End Sub

Private Sub GetSettings()
Dim Count As Integer
    cboAddress(Count).ListIndex = GetSetting _
        (ProjectName, "Startup", "NodeAddress" & CStr(0), 0)
For Count = 1 To 7
    cboCPU(Count).ListIndex = GetSetting _
        (ProjectName, "Startup", "NodeCPU" & CStr(Count), 0)
    cboAddress(Count).ListIndex = GetSetting _
        (ProjectName, "Startup", "NodeAddress" & CStr(Count), 0)
    chkNodeActive(Count).Value = GetSetting _
        (ProjectName, "Startup", "NodeActive" & CStr(Count), 1)
Next Count
End Sub

Private Sub InitializeNodeActiveCheckBoxes()
Dim Count As Integer
For Count = 1 To 7
    chkNodeActive(Count).Value = 1
Next Count
End Sub
```

Listing 4: Code for the Nodes window, which enables users to select and configure nodes in the network. (Sheet 1 of 2)

```
Private Sub InitializeNodeAddressComboBoxes()
Dim Count As Integer
Dim Address As Integer
'Address range is 67h to 6Eh
For Count = 0 To 7
    For Address = &H67 To &H6E
        cboAddress(Count).AddItem Hex$(Address)
    Next Address
Next Count
For Count = 0 To 7
    cboAddress(Count).ListIndex = Count
Next Count
End Sub
```

```
Private Sub InitializeNodeCpuComboBoxes()
Dim Count As Integer
For Count = 1 To 7
    cboCPU(Count).AddItem "PC"
    cboCPU(Count).AddItem "Stamp"
Next Count
End Sub
```

```
Private Sub SaveSettings()
Dim Count As Integer
SaveSetting ProjectName, "Startup", _
    "NodeAddress" & CStr(0), cboAddress(0).ListIndex
For Count = 1 To 7
    SaveSetting ProjectName, "Startup", _
        "NodeCPU" & CStr(Count), cboCPU(Count).ListIndex
    SaveSetting ProjectName, "Startup", _
        "NodeAddress" & CStr(Count), cboAddress(Count).ListIndex
    SaveSetting ProjectName, "Startup", _
        "NodeActive" & CStr(Count), chkNodeActive(Count).Value
Next Count
End Sub
```

Listing 4: Code for the Nodes window, which enables users to select and configure nodes in the network. (Sheet 2 of 2)

A Simple Stamp Network

The second network is example is a very simple one. It connects multiple Basic Stamp II's using open-baud mode. The programming uses non-inverted communications, so the data line uses a pull-up resistor to +5V.

The programming is again a master/slave network. Listing 6 is the master's program, and Listing 7 is the slave's. For use in debugging, the code reads and sets I/O bits on each Stamp.

This network is suitable for as many as eight Stamps, over a total distance of about 15 feet. With more nodes and over longer distances, proceed at your own risk!

The network traffic is half-duplex and uses just one port bit.

```
'Basic Stamp II RS-485 network node.
'The Stamp periodically waits for serial input.
'If it receives a byte that matches its node number,
'it sends an Acknowledge, waits for a message, and
'sends a message in reply.
'If the received byte doesn't match, or if the wait times out,
'the Stamp takes no action at the serial port
'and continues on with its other activities.
'All debug statements are for testing & may be removed.

'The RS-485 transmit and receive uses 3 port bits.
'One port bit is direction control.

'Each node must have a unique address!
'Allowed addresses are 68h through 6Eh.
NodeAddress con $6A

Constants:
'Serial-port settings: 2400 bps, non-inverted, 8-N-1
BaudMode con 396

'Serial transmit output
SerialOutput con 14
'Serial receive input
SerialInput con 15
'Direction-control output:
TRControl con 13

'Delay (milliseconds) before enabling transmitter.
'Allows previous node to disable its transmitter.
EnableTransmitterDelay con 500
'Time to wait for incoming data before giving up.
timeout con 2000

'Variables:

DataIn1 var byte
DataIn2 var byte
DataOut1 var byte
DataOut2 var byte
```

Listing 5: Code for a Basic Stamp II node. (Sheet 1 of 3)

```
'Configure I/O bits as input or output.
dir13=0
dir14=1
'Direction control:
dir15=1
'For debugging, bits 0-3 are outputs, 4-7 are inputs.
dira=%1111
dirb=%0000
'Bits 8-12 are unused.

'Initialize the output high.
high SerialOutput

'main loop:
  debug "RS-485 network",cr

Begin:
'Enable the receiver, disable the transmitter (default state):
low TRControl
gosub NodeActivities
'Wait until a byte is received or timeout.
'If a byte is received, store it in DataIn1.
'If timeout, give up.
serin serialinput,baudmode,timeout,NoData,[DataIn1]
  debug "received: ",hex2 DataIn1,cr
if DataIn1=NodeAddress then GetMessage
NoData:
goto begin
```

Listing 5: Code for a Basic Stamp II node. (Sheet 2 of 3)

```
GetMessage:
pause EnableTransmitterDelay
'Enable the driver, disable the receiver
high TRControl
'Send NodeAddress
serout SerialOutput,Baudmode,[NodeAddress]
  debug "sending: ",hex2 NodeAddress,cr
'Enable the receiver, disable the driver
low TRControl
'Read 4 ASCII Hex bytes & store as binary values.
Serin SerialInput,baudmode,timeout,NoMessage,[hex2 DataIn1,hex2
 DataIn2]
'Reply with the node address and 4 ASCII Hex bytes
'representing 2 binary values.
gosub ProcessReceivedData
'Wait to be sure the master's transmitter is disabled.
pause EnableTransmitterDelay
'Enable the driver, disable the receiver
high TRControl
Serout SerialOutput,baudmode,[NodeAddress,hex2 DataOut1,hex2
 DataOut2]
goto Begin

NodeActivities:
'Use for any activities the node is responsible
'for on its own.
return

NoMessage:
  debug "no data received",cr
goto Begin

ProcessReceivedData:
'Some test values and actions for debugging.
  debug "received: ",hex2 DataIn1, " ", hex2 DataIn2,cr
'Set output bits 0-3 to match bits 0-3 of DataIn2:
outa=DataIn2.lownib
'First byte to send is received byte + 1:
DataOut1=DataIn1+1
'Second byte to send contains the values of input bits 4-7:
DataOut2=inb
  debug "sending: ",hex2 NodeAddress, " ", hex2 DataOut1, " ",
 hex2 DataOut2, cr
return
```

Listing 5: Code for a Basic Stamp II node. (Sheet 3 of 3)

```
'Stamp II network example using open-baud mode.
'Master node
'The master sends a byte to each of up to 7 Stamps.
'In response, the receiving Stamp sends a byte to the master.
'Bits 5-7 identify the node the byte is addressed to.
'Bits 0-4 may contain data, commands, or other information.

'In this example, the receiving Stamp sets its port bit 0
'to match the received bit 4. Bits 0-3 are unused.
'When the master receives a reply, it sets its port bit
'corresponding to the node number to match bit 4 of the
'received byte. Bits 5-7 of the reply identify the master
'as the recipient (node 8). Bits 0-3 are undefined.

Node var nib
DataOut var byte(7)
DataIn var byte(7)
InputBits var byte
OutputBits var byte
NodeNumber var nib
temp var byte
'2400 bps, not inverted, open-baud mode
baudmode con 36081

'Pin used for serial I/O:
ComBit con 7

inputs:
dir0= 0
dir1= 0
dir2= 0
dir3= 0
dir4= 0
dir5= 0
dir6= 0
'This pin is input except during Serout
dir7 = 0
```

Listing 6: Code for an open-baudmode network’s master node. (Sheet 1 of 3)

```
'outputs:
dir8= 1
dir9= 1
dir10= 1
dir11= 1
dir12= 1
dir13= 1
dir14= 1

'Bit 15 is unused.

begin:
GoSub DoPortIO
GoSub PollTheStamps
goto begin

PollTheStamps:
'Send and receive a byte from each Stamp.
For Node = 0 to 6
  'Send the appropriate DataOut byte.
  serout ComBit, baudmode,[DataOut(Node)]
  debug  "Node=",hex node, " Output=",hex Dataout(node),cr
  'Wait for reply; skip if no response.
  serin ComBit, baudmode, 500,NextNode,[DataIn(Node)]
  debug " Input=",hex DataIn(Node),cr
  NextNode:
  'Pause for Stamps to catch up.
  pause 500
Next
return

DoPortIO:
'Read and write to port bits.
'Read port bits 0-7.
InputBits=InL
'Store the bit values as bit 4
'in the appropriate bytes in the DataOut array.
'(Bit 7 is unused.)
outputbits=0
```

Listing 6: Code for an open-baudmode network’s master node. (Sheet 2 of 3)

```
For Node=0 to 6
  'Store bit 0 of InputBits as bit 4 of DataOut(Node)
  DataOut(Node)=(Node * $20) + (InputBits.Bit0 * $10)
  'Shift Inputs right to position the next bit as bit 0.
  InputBits=InputBits  >> 1
  'Write to port bits 8-14.
  'Store bit 4 of the received byte as bit 7 of PortOutputs.
  'Use Temp variable to get value of bit4 from array byte
     Temp=DataIn(Node)
     OutputBits.bit7 = Temp.bit4
  'Shift PortOutputs right to position a new bit as bit 7.
  OutputBits=OutputBits >> 1
'debug dec nodenumber, " rec: ",hex datain.bit4(nodenumber*8),
 "out: ",bin outputbits,cr
'debug hex dataout(nodenumber),cr
Next
'debug "output: ",hex outputbits,cr
'Set port bits 8-14 to match PortOutputs.
OutH=OutputBits
return
```

Listing 6: Code for an open-baudmode network's master node. (Sheet 3 of 3)

```
’Stamp II network example using open-baud mode.
'Slave node
'When the slave receives a byte, it sets its bit 0
'to match bit 4 of the received byte.
'Bits 5-7 identify the node. Bits 0-3 are unused.

'2400 bps, not inverted, open-baud mode
baudmode con 36081

'Pin used for serial I/O:
ComBit con 7

'Each slave node has a unique number, 0-6.
'The master is node 7.
Node con 0

DataOut var byte
DataIn var byte
Address var nib

inputs:
dir0= 0
'Bit 7 is input except during Serout
dir7= 0

'outputs:
dir1= 1

'Bits 2-6, 8-15 are unused.

'Set bits 5-7 to the master’s address.
DataOut = $E0

begin:
gosub NodeActivities
'Watch for incoming data; quit if no activity.
'debug "watching"
serin ComBit, baudmode, 1000, begin,[DataIn]
'debug hex datain,cr
'If bits 5-7 match the node number, take action.
Address=DataIn >> 5
debug "Address: ",dec address,cr
if Address = Node then Respond
goto begin
```

Listing 7: Code for an open-baudmode network’s slave node. (Sheet 1 of 2)

```
Respond:
'Set bit 1 to match bit 4 of DataIn
Out1=DataIn.bit4
'Read bit0 and set bit 4 of DataOut equal to it.
DataOut=$E0 + ($10 * in0)
debug hex dataout,cr
'Send the reply
serout ComBit, baudmode, [dataout]
goto begin

NodeActivities:
'Place other node activities here.
return
```

Listing 7: Code for an open-baudmode network's slave node. (Sheet 2 of 2)