



Column #28, June 1997 by Scott Edwards:

Nifty Networking Chips Link Stamps Far and Wide

Use an RS-485 transceiver for reliable network comms

STAMPS ARE GREAT for bridging the gap between PCs and hardware sensors or controls. The Stamps easily communicate with the PC serial port, and just as easily interface with primitive hardware on its own terms.

The obvious next step is to network Stamps together so that a single PC can gather data from Stamps scattered around the house, garden, building, or factory.

While Stamps offer some built-in networking capabilities, you're probably pushing your luck if you build a network more than a few dozen feet long. If you want real networking, you want RS-485.

This month's column will show how to interface Stamps to an RS-485 transceiver chip for easy and robust network communication.

What's RS-485?

You are probably familiar with RS-232, the electronics industry specification that describes a garden-variety computer serial port used to communicate with modems and other accessories. In RS-232 signaling, a positive voltage (+5 to +15V) represents a digital 0, and a negative voltage (-5 to -15V) a 1. Those voltages are specified relative to a signal ground, which must be common to both RS-232 devices. I say "both" because RS-232 is meant to connect only two devices at a time. According to the specs, maximum cable length is 50 feet.

RS-232 calls for separate transmit and receive lines, so both ends of an RS-232 connection may send and receive data simultaneously, a capability called *full duplex*.

RS-485 is an alternative standard designed to allow multiple devices (up to 32) to communicate over a single pair of wires up to 4000 feet long. RS-485 signals are not referenced to ground *per se*, but to the voltage difference between the two wires of the signaling pair. The two wires of the pair are designated A and B; when the voltage on A is greater than the voltage on B, that's a digital 1. When B is greater than A, that's a 0.

This differential signaling is nearly immune to noise pickup over long wire runs. The only catch is that the local ground potentials of the RS-485 devices cannot differ by more than ± 7 volts. If they do, the system won't work.

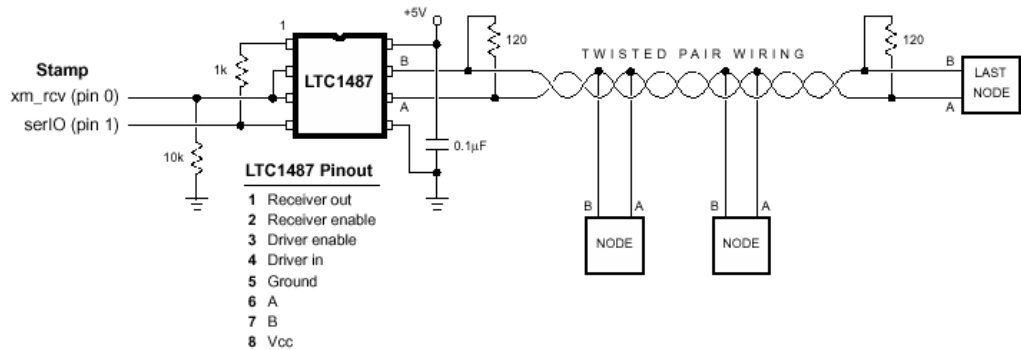
Since only one pair is used for receive and transmit, the RS-485 devices have to take turns using it. This is called *half duplex*. If two RS-485 units try to transmit at the same time, the error condition is called *bus contention*.

Making RS-485 work

An RS-485 network is like a group of people using walkie-talkies on a single frequency. There are two major limitations:

- When a unit is listening, it cannot talk; when talking, it can't listen.
- If two units talk at the same time, neither will be heard correctly.

Figure 28.1: Networking demo setup.



Given those limitations, it's vital to coordinate units' behavior. One of the simplest ways to do this is to appoint one unit *master* and all other units *slaves*. No slave unit is permitted to transmit until told to do so by the master unit. This solves the problem of preventing units from transmitting at the same time. It does, however, require the master to check in with all of the slave units periodically for updates. If a slave unit possesses urgent information, it must still wait until called upon by the master before transmitting.

An RS-485 demo

Figure 28.1 illustrates how I setup a demonstration network using four BS1 controllers connected to LTC1487 RS-485 transceivers. I chose the LTC1487 because it has some virtues beyond the normal RS-485 specs, including capability for up to 256 units on a line, low electromagnetic interference (EMI) from the signaling wires, and various sorts of fool-proofing against static electricity, open inputs and bus contention. It also draws far less current than most comparable devices, just 120mA in receive mode.

Hardware. The LTC1487 has separate driver-enable and receiver-enable pins. The driver is enabled by a 1 on pin 3; the receiver by a 0 on pin 2. To conserve Stamp pins, I tied these together so that a 0 means *send* and a 1 means *receive*.

Why doesn't the 1487 have just a single send/receive pin? It supports two other modes: one in which both the driver and receiver are enabled, allowing a controller to hear its own serial output. The Stamp can't send and receive simultaneously, so this mode is no use to it.

Column #28: Nifty Networking Chips Link Stamps Far and Wide

In the other unused mode, both the driver and receiver are disabled. This amounts to shutting off the LTC1487, reducing its current draw to 1mA. However, the LTC1487's current draw in receive mode is only 120mA. Unless your application uses battery power and software-controlled shutdown, the pin saved by combining the enable inputs is more valuable than the possible savings in current draw.

The 10k resistor to ground prevents the LTC1487 from accidentally getting into transmit mode when the Stamp is disconnected or reset.

I also tied the receiver output and driver input lines together. Stamps can send and receive through the same pin, so there's no reason to do otherwise. The 1k resistor in series with the receiver-output prevents damage in the event that the receiver is enabled while the Stamp is trying to transmit data. This would occur only in the event of a bonehead program bug, but a resistor is cheap insurance.

A Stamp plus the LTC1487 circuit make up what I'm calling a *node* in the figure. Per the LTC1487 specs, you can have up to 256 nodes along a 2000-foot stretch of twisted-pair wire. The manufacturer's application diagram shows the use of shielded cable, with the shield connected to ground at one end. For shorter runs of wire in electrically quiet surroundings, you may find that shielding is not strictly necessary. You also may find that you can run the full 4000-foot RS-485 cable length if you comply with the 32-node limit. Networking is very much a your-mileage-may-vary kind of business.

Figure 28.1 shows the first and last nodes on the wire with 120Ω *terminating resistors* across A and B. These help prevent reflections and ringing on the network. I'm going to duck the difficult job of explaining the theory behind that—it is enough to understand this: A properly terminated RS-485 network preserves the nice, crisp on/off waveforms of the serial data. An unterminated or incorrectly terminated network can distort the signals, sometimes to the point of causing data errors.

No matter how many nodes are connected to the net, only two of them—first and last—get terminating resistors.

One of the LTC1487's fool-proofing features (controlled slew rate) makes it less fussy about net termination than other RS-485 devices, but do it right anyway.

As I said before, it is not necessary for the nodes to share a common ground. For my demo, I powered the nodes with individual batteries so that they were floating relative to ground.

If you're setting up a net in which nodes will be connected to separate grounds, make sure to check for differences in ground potential with an AC voltmeter. If the difference is greater than ± 7 volts, you have a problem; the net won't work. A discussion of building electrical wiring and grounding practices is waaay beyond the scope of this column (and beyond my experience as a non-electrician). Suffice to say that you must check and if necessary correct grounding problems before you continue with your net.

Software. Listings 28.1 and 28.2 show how I programmed my demo network. Listing 28.1 runs the master; listing 28. runs on each of the slaves, which are identified A, B, and C.

The master program puts the LTC1487 into transmit mode and sends the ID of one of the slaves, followed by a number for the slave unit to write to its unused six output pins. The master then waits for the slave to acknowledge receipt of the data by sending back its ID.

Because of the way BS1 serial input instructions work, this network is fragile. If a slave unit is turned off, the master will get stuck waiting for acknowledgment. There are two remedies: use a BS2 master and set a serial timeout (Program Listing 28.3), or dedicate an additional Stamp to the job of monitoring the network and resetting the master or providing a substitute response.

Note that the listings send data to the slaves in numeric text format (by preceding values with #), rather than as single bytes. If data were sent as bytes, it could accidentally match a different slave's ID, causing that slave to expect further data. By limiting slave IDs to letters and data to numbers, we prevent such conflicts. If you're not clear on how numbers are represented as text, you might review *Stamp Applications* no. 16.

One more subtlety—the Serout instruction that talks to a particular slave unit appends a period “.” after the numeric data. This has no significance other than to cause the receiving program to recognize that it has received a complete, valid number. I could have used any non-numeric character, but the period seemed appropriate.

The slave program in Listing 28.2 is just the opposite of the master program. A slave waits to receive its ID, then grabs the number that follows and writes it to pins 2 through 7. It then enables its transmitter and sends its ID back to the master as an acknowledgment.

Listing 28.3 is a BS2 master-unit program. It operates in exactly the way as the BS1 version, but uses the BS2's serial timeout capability to recover from cases in which the slaves fail to respond.

Refinements

I have only scratched the surface of RS-485 networking, a pretty complicated subject. If you decide to put some of the principles shown here into action, be prepared for some challenges. For example, if you make changes to the *protocol*, the rules governing the way the master and slaves communicate, you will be fighting a debugging war on two fronts.

If the net doesn't work, is the bug in the master code, the slave code, or both? What if the code is fine, but you reset the master before the slaves so that the slaves missed the message that would have started the net? Or what about a loose or incorrect connection? Or... You get the idea.

One obvious refinement might be to include a PC in your developing network. RS-232 to RS-485 adapters are available from better computer and electronics suppliers like Jameco (sources). But you should be comfortable with serial-port programming before you add this new complication to your life.

Column #28: Nifty Networking Chips Link Stamps Far and Wide

```
' Program Listing 28.1. BS1 RS-485 Master
' Program: MASTR485.BAS (RS-485 net master)
' This program demonstrates some basic principles of using an RS-485
' transceiver chip (LTC1487 used in our setup). The program sends
' a byte to each of three 'slave' units, which write that bit
' pattern to their 6 output pins not used for RS-485 communication
' and control. To confirm receipt of the message, slaves reply with
' their node id, in this case "A", "B", or "C".

SYMBOL id = b11          ' ID number of net node.
SYMBOL reply = b10      ' Response from node.
SYMBOL pinSet = b9      ' Pin setting for node.
SYMBOL xm_rcv = 0       ' Pin 0 sets transmit (1) or receive (0).
SYMBOL serIO = 1        ' Pin 1 is used for serial input/output.

' For the purpose of the demo, the pinSet value that the master will
' tell the slaves to write to their outputs will be an easy-to-
' recognize sequence: 000001 000010 000100 001000 010000 100000 000001...
' You can connect LEDs to the slaves' pins 2-7 to watch the sequence,
' or take my word for it. Since the lowest two bits are reserved for
' use by the RS-485 transceiver, the starting pattern is 00000100.
begin:
  pinSet = %00000100     ' Starting bit pattern.
again:
  pause 1000            ' Run slowly for demo purposes.
  for id = "A" to "C"   ' Cycle through ids A, B, C.
    high xm_rcv: pause 1 ' Turn on 485 transmitter; wait briefly.
    serout serIO,T2400,(id,#pinSet, ".") 'Send id, bit pattern, and ".".
    low xm_rcv          ' Switch to receive mode.
    serin serIO,t2400,reply ' Receive the slave's response.
    debug "Unit ", #@reply," OK.",cr ' Display on debug screen.
  next                  ' Next unit.
  debug cr,cr          ' Scroll the screen.
  pinSet = pinSet * 2  ' Shift bit pattern left.
  if pinSet = 0 then begin ' If bitpattern is 0, reload %00000100.
  goto again           ' Else continue.
```

Column #28: Nifty Networking Chips Link Stamps Far and Wide

```
' Program Listing 28.2. BS1 RS-485 Slave
' Program: SLAVE485.BAS (RS-485 net slave)
' This program demonstrates some basic principles of using an RS-485
' transceiver chip (LTC1487 in our setup). The program waits for
' an ID letter matching the value myID set below. When that letter
' is received, the data that follows--a number in text format like
' "192"--is saved to the byte variable and then written to the upper
' six output pins. The program acknowledges receipt of the data
' by sending its ID back to the master.
' >>The master program is set up to expect three slaves with ids "A",
' "B" and "C". Use this program for all slaves, but change the symbol
' myId below to "B" and "C" for the other slave units.

SYMBOL myId = "A"           ' ID letter of this node.
SYMBOL pinSet = b10        ' Pin setting for node.
SYMBOL xm_rcv = 0         ' Pin 0 sets transmit (1) or receive (0).
SYMBOL serIO = 1          ' Pin 1 is used for serial input/output.
again:
  dirs = %11111100        ' Set upper 6 pins to output.
  low xm_rcv              ' Set 485 transceiver to receive mode.
  serin serIO,t2400,(myId),#pinSet ' Wait for id, then receive number.
  pinSet = pinSet & %11111100 ' Strip off two lowest bits of number.
  let pins = pinSet       ' Write that value to outputs.
  high xm_rcv: pause 10   ' Set 485 transceiver for transmit.
  serout serIO,T2400,(myID) ' Send back my ID letter.
goto again                ' Do it all over.

' Program Listing 28.3. BS2 RS-485 Master
' Program: MASTR485.BS2 (RS-485 net master for BS2)
' This program demonstrates some basic principles of using an RS-485
' transceiver chip (LTC1487 used in our setup). The program sends
' a byte to each of three 'slave' units, which write that bit
' pattern to their 6 output pins not used for RS-485 communication
' and control. To confirm receipt of the message, slaves reply with
' their node id, in this case "A", "B", or "C". If a slave does not
' respond within 100 milliseconds, this BS2 version of the master
' program displays an error message and carries on.

id      var byte          ' ID number of net node.
reply   var byte          ' Response from node.
pinSet  var byte          ' Pin setting for node.
xm_rcv  con 0             ' Pin 0 sets transmit (1) or receive (0).
serIO   con 1             ' Pin 1 is used for serial input/output.
T2400   con 396           ' Noninverted 2400-baud serial baudmode.

' For the purpose of the demo, the pinSet value that the master will
' tell the slaves to write to their outputs will be an easy-to-
' recognize sequence: 000001 000010 000100 001000 010000 100000 000001...
' You can connect LEDs to the slaves' pins 2-7 to watch the sequence,
' or take my word for it. Since the lowest two bits are reserved for
' use by the RS-485 transceiver, the starting pattern is 00000100.

begin:
  pinSet = %00000100      ' Starting bit pattern.
again:
```

Column #28: Nifty Networking Chips Link Stamps Far and Wide

```
pause 1000          ' Run slowly for demo purposes.
for id = "A" to "C" ' Cycle through ids A, B, C.
high xm_rcv: pause 1 ' Turn on 485 transmitter; wait briefly.
serout serIO,T2400,[id,DEC pinSet,"."] 'Id, bit pattern, and ".".
low xm_rcv          ' Switch to receive mode.
serin serIO,T2400,100,netErr,[reply] ' Get response.
debug "Unit ", reply," OK.",cr ' Display on debug screen.
errReturn:
  next              ' Next unit.
  debug cr,cr       ' Scroll the screen.
  pinSet = pinSet * 2 ' Shift bit pattern left.
  if pinSet = 0 then begin ' If bitpattern is 0, reload %00000100.
goto again          ' Else continue.

' If a slave unit does not repond within 100ms, the program comes here
' to display an error message on the screen, then continues with the
' next unit.
netErr:
  debug "Unit ", id," : NO RESPONSE.",cr ' Display on debug screen.
goto errReturn
```

