

Serial Network Hub (Part 1)

Network Topology and Design Planning

If you're building a custom multipart animatronics or robotics system, consider adding a serial network hub to run the show. Here you learn why building a six-port RS-485 hub for a custom network packet system can simplify node wiring and more.

What can I say? I really like Halloween. I also enjoy designing hardware and writing software. What's a geek to do? Combine both interests into a fully automated Halloween display system each year, of course!

This is the third article I've written about the technology behind my ever-evolving Halloween display (www.socalhalloween.com). What began in 1995 as a couple of Styrofoam tombstones, a pumpkin, and a sheet shaped like a ghost has mutated into a large-scale yard display featuring custom animatronics, smoke machines, lights, six channels of digital audio, and a dedicated RS-485 control network to run it all. And the latest addition? An RS-485 hub designed to handle my custom RS-485 serial network packets (see [Photo 1](#)).

The basic design philosophy behind my control system is to have programs running on a laptop inside my house that talk to and control hardware nodes located outside. An example of a hardware node is shown in [Photo 2](#). My generic I/O node can control or poll anything that can be interfaced to its I/O bits. I created a data packet format for my system to allow me to create software and hardware that can communicate without writing custom protocols for each new device. My Halloween network uses standard CAT-5 Ethernet cables to carry both full-duplex RS-485 communications as well as 12 V of DC power. This lets me plug hardware nodes onto cables

and provide both data and power with a single cable. I chose CAT-5 cables because they are affordable, readily available, and provide four twisted pairs of conductors in them.

The RS-485 network turned out to be a solid design choice. Through the use of conventional serial communications and an inexpensive level converter IC, I have a robust network that works flawlessly. All the dedicated hardware can interact, and the laptop controls everything.

RS-485 DETAILS

If you're familiar with RS-485, you're probably thinking, "Why does he need a hub for RS-485?" If you're unfamiliar, let me explain a bit about why everyone else is perplexed.

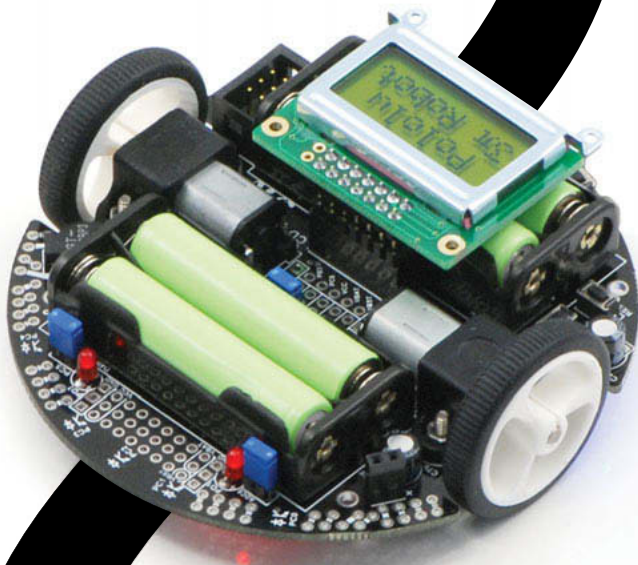
RS-485 is an electrical spec that sends data using a differential balanced line over twisted pair wiring. Simply put, the data is sent via two transmitters simultaneously, but



Photo 1—This is the finished network hub design.

3pi Robot

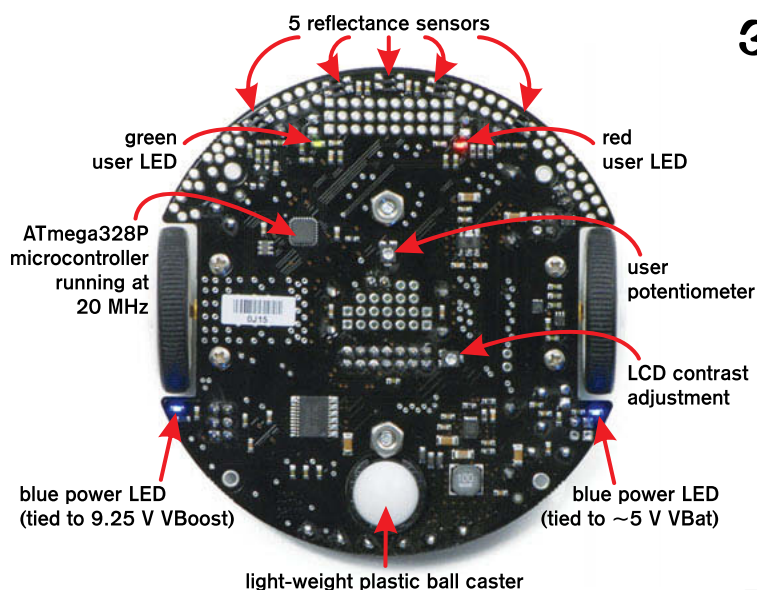
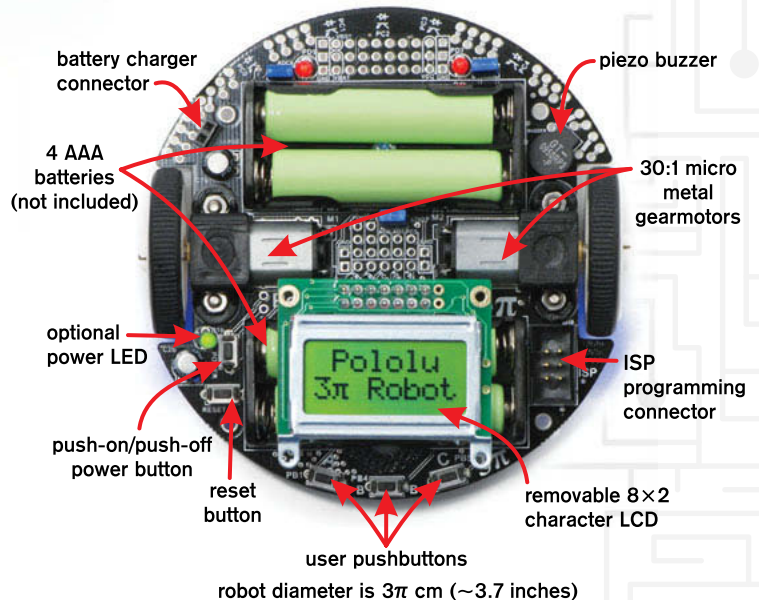
The Pololu 3pi robot is a complete, high-performance mobile platform featuring two micro metal gearmotors, five reflectance sensors, an 8×2 character LCD, a buzzer, and three user pushbuttons, all connected to a C-programmable ATmega328P microcontroller. Capable of speeds exceeding 3 feet per second, 3pi is a great first robot for ambitious beginners and a perfect second robot for those looking to move up from non-programmable or slower beginner robots.



Item #975: **\$99.95**
Available at www.pololu.com/3pi

3pi Line-Following and Maze-Solving

The 3pi robot is designed to excel in line-following and maze-solving competitions. It has a small size (3.7" diameter, 2.9 oz without batteries) and takes just four AAA cells (not included), while a unique power system runs the motors at a constant 9.25 V independent of the battery charge level. The regulated voltage allows the 3pi to reach speeds up to 100 cm/second while making precise turns and spins that don't vary with the battery voltage.



3pi Features

- Two metal gearmotors
- 8×2 character LCD
- Five reflectance sensors
- Buzzer, LEDs, three user pushbuttons
- High-traction silicone tires
- Speeds exceeding 3 ft/sec using innovative constant-voltage motor supply
- On-board C/C++-programmable Atmel AVR ATmega328P microcontroller
- Works great with free GNU C/C++ compiler and free AVR Studio development environment.
- Extensive C/C++ libraries and example programs make it a snap to use all hardware features.

the signal is inverted on one of the transmitters. On the receiving side is a pair of receivers, one of which takes the inverted signal and inverts it again, thus restoring the polarity back to normal. This reinverted signal is summed with the signal that was sent unchanged, and the result is the final output signal. Figure 1 shows the basic concept. There is a reason for going through all this trouble. If any electrical noise is induced on the line during transmission, it will be induced the same amount on both wires. However, since the receiver inverts one line and then sums them, the noise will now be inverted on one of the lines. When the two signals are added together, the noise cancels itself out, leaving just the pristine signal. Unlike RS-232, which requires the use of a negative voltage supply, RS-485 is single ended, which simplifies power issues. The RS-485 spec can reliably send data up to 1,200 m.

There are two standard network topologies used with RS-485: half duplex and full duplex. In a half-duplex

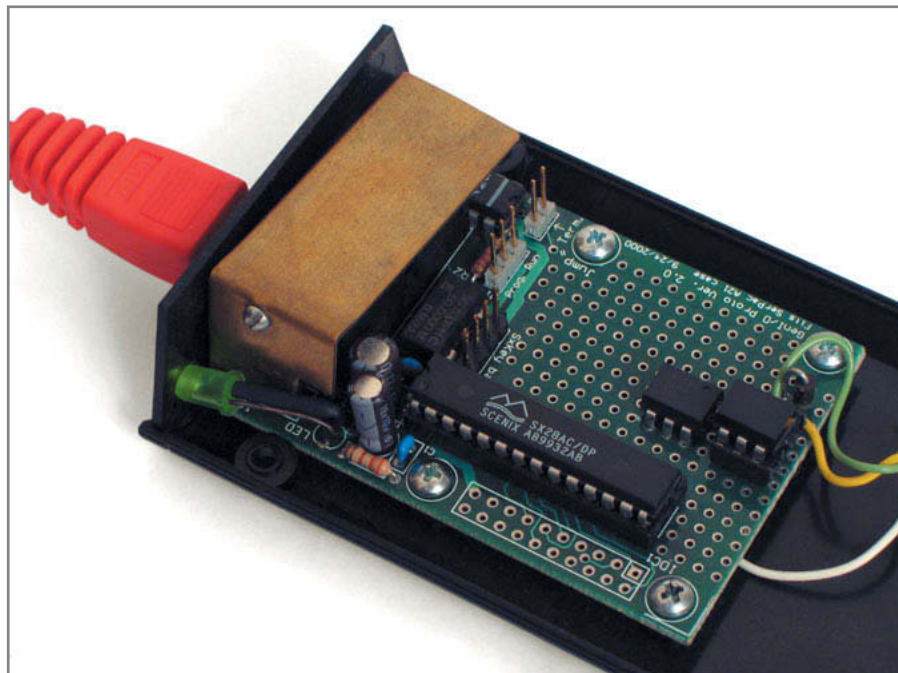


Photo 2—This is a generic I/O node from my RS-485 networked show system.

system, as shown in Figure 2, two wires are used to connect all nodes. Each node's transmitter pair and receiver pair are connected to these two wires,

and any node can communicate with any other node. The advantage is that you only need two wires and a common ground to enable communication—thus

Easy Embedded Linux

OMNI-EP



- 200 Mhz ARM9 CPU
- 10/100 Mb Ethernet
- 32 MB RAM
- 16 MB Flash
- 16 Digital I/O Lines
- 2 Ports of USB 2.0
- SPI Bus
- AC97 Amplified Audio
- Battery Backed Clock
- 2 Serial Ports
- Low Power Consumption
- RoHS Compliant

Our newest ARM9 Linux controller, the OmniEP doesn't cost an arm and a leg. It delivers removable storage, amplified audio, ethernet and serial RS232 communication ports in a rugged and attractive enclosure. Models without enclosure and LCD available.

The OmniEP comes preloaded with Linux to jumpstart your development process, with LCD and pushbutton drivers supplied. Large capacity USB drives can be easily mounted in the USB port.


Call 530-297-6073 Email sales@jkmicro.com
www.jkmicro.com

JK microsystems, Inc.

International Orders Welcome



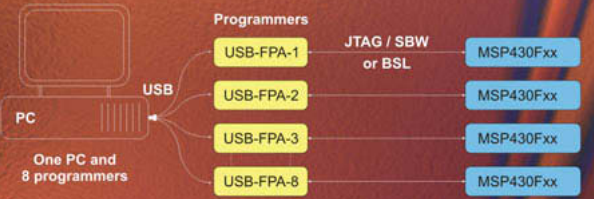
FlashPro430 FlashPro-CC FlashPro2000 GangPro430 GangPro-CC



USB Flash Programmers for Texas Instruments' MCUs
MSP430, Chipcon CCxx, C2000 DSPs

**Reliable and the fastest programmer on the market.
 Perfect for production usage.**

- * can assign unique serial number
- * up to eight programmers can be connected to one PC and program target devices simultaneously



One PC and 8 programmers

Elprotronic

Incorporated

www.elprotronic.com

simplifying wiring. The disadvantage is that it complicates the actual communication since only one node can transmit on the wire pair at a time. If two or more nodes try and transmit simultaneously, the resulting collision will simply create digital noise and no data will get through.

The full-duplex system uses two wire pairs and requires the use of a master/slave system. The node designated as the master can communicate with every slave, and vice versa, but slaves can't talk to each other. The reason is that one wire pair connects the transmitters of the master to all the slaves, and the other wire pair connects all the transmitters of the slaves to the master receiver pair (see Figure 3).

AN RS-485 HUB

If a single pair of wires can have multiple receivers or transmitters, why make a hub? The reason is that even with a full-duplex setup with multiple slave transmitters on a wire pair, only one of them can talk at a time. The issue of multiple devices trying to use a shared resource like a network is at the heart of all network design. Ethernet enables multiple transmitters and receivers to hang on the wires, but the Ethernet spec also dictates a collision detection and retransmission scheme. After checking that the line is not in use, an Ethernet device transmits while monitoring the line to make sure the data on the wire is the same as what it is sending. If it isn't, the device assumes some other device is transmitting simultaneously. All Ethernet devices do this, and when they detect collisions, they both stop and wait a randomized amount of time before retransmitting. In theory, the randomization means one will start transmitting first and the other will wait for the line to be clear before it transmits.

This works quite well in practice. Witness the billions of Ethernet devices operating worldwide right now. However, the RS-485 spec is only an electrical spec, meaning that issues such as collision detection are left up to the individual designer to take care of. As a result, most folks

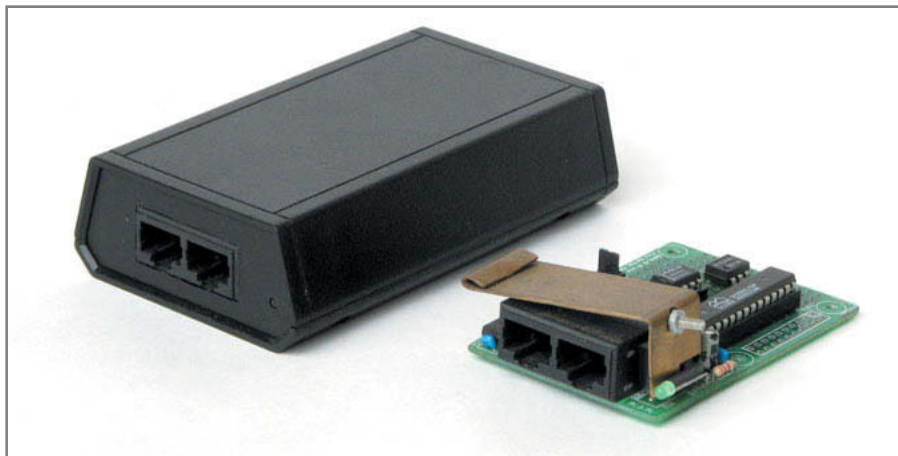


Photo 3—This hardware node has RS-485 networking. The board on the right is the same as what's in the case on the left.

bypass collision detection and use the much simpler approach of letting only one transmitter talk at a time.

There are numerous common approaches to handling this rule. My system uses the full-duplex master/slave concept with polling. In this approach, the master is the only node that can initiate communication. Slaves never send a packet without

the master node first requesting information from that slave node. Further simplifying communication, slave nodes do not send any packets back to acknowledge having received a packet from the master node. The master assumes that the packets it sends are being received. Having each slave acknowledge packets would not only complicate the network software, but

Visit us in Booth 1142 at the Embedded Systems Conference in San Jose April 27-29, 2010

See It & Solve It
with
USBees Test Pods

- ✓ Logic Analyzers
- ✓ Oscilloscopes
- ✓ Signal Generators
- ✓ Protocol Analyzers
 - I2C, SPI, Async, CAN
 - 1-Wire, USB, PS/2
 - I2S, SMBus, Serial
- ✓ Configurable
- ✓ Programmable
- ✓ High Speed USB
- ✓ PC-Based
- ✓ Affordable

USBees™
USB based Electrical Engineer
www.USBees.com

A new era, a new ESC

New tracks. New courses. New focus.

ESC Silicon Valley brings together systems architects, design engineers, suppliers, analysts, and media from across the globe. With cutting edge product demonstrations, visionary keynotes, and hundreds of essential training classes, ESC is the ideal conference for the embedded design community to learn, collaborate, and recognize excellence.

ESC Silicon Valley 2010 features:

- Two "Build Your Own Embedded System" programs featuring the new Intel Multicore Atom and the Freescale Tower development system
- Multicore Expo and Conference co-located with ESC Silicon Valley
- ARM Partner Pavilion
- Display and Connected Devices Emerging Technology Zones
- Keynotes from Dr. Michio Kaku, theoretical physicist and bestselling author and Richard Templeton, CEO & President, Texas Instruments

Start your own personal development at ESC 2010. You can't afford to miss it.

Silicon Valley

McEnery Convention Center, San Jose, April 26 - 29, 2010

Conference: April 26 - 29, 2010

Expo: April 27 - 29, 2010

Register Today.

www.embedded.com/sv

Learn today. Design tomorrow.



it would greatly slow down the system. Imagine that the master wants to send packets to nodes 1 and 2. If the system had each slave acknowledge every packet sent, the master could not send a packet to node 2 until it received an acknowledgement from node 1. Otherwise, there is the chance that both nodes 1 and 2 could overlap their reply packets, thus destroying the replies from both.

Requiring acknowledgement of packets received also requires a timeout period to prevent the master node from waiting forever for a slave acknowledgement packet. Timeout periods are trickier than they seem. Too long, and the system response is slowed to a crawl when a slave node has a problem. Too short, and the slave could send the acknowledgement after the timeout period. This new packet would also require an acknowledgment, and the problem could keep repeating. Suppose we create a timeout period of 0.25 s. While this certainly sounds like a nice, short duration, the potential for network slowdown due to a timeout is massive. The minimum packet size in my Halloween Network is 4 bytes and the network runs at 9,600 bps. Thus, in 0.25 s, the system could theoretically send 60 packets. If a node was not

responding for some reason, it would send only one packet in that same 0.25 s, thus making the network run 60 times slower. Okay, let's cut the timeout in half and have a 0.125-s timeout. It sounds good until you realize that a timed out node will still cause the network to run 30 times slower than it should. You can see the problem is not easily solved simply by shrinking the timeout period. This is why I do not have slave nodes send acknowledge packets.

I have a laptop, which is the master,

and all the hardware nodes that do the real work of running the show are the slaves. Most of the nodes are output-only devices and simply need to be told what to do. For example, the smoke machine node simply triggers an output bit connected to an opto-relay that then triggers the actual smoke machine. If the smoke machine node misses a packet, the worst-case scenario is that a puff of smoke doesn't get created. However, it will be getting another packet in a few seconds to generate another smoke puff, and the end result doesn't impact the show. Input nodes are a different case. For example, I have a node that is connected to an infrared beam system that can tell when a person has passed by. This is

used to trigger events in the show. That node clearly has to tell the laptop when the beam has been broken. The preferred approach would allow the node to transmit whenever the beam got broken; but instead, I have a program on the PC polling the node four times a second to check the status. Like most polling schemes, it is an inefficient use of network bandwidth and processor overhead. I would love to be able to have that node simply send a message whenever the event happens without risking packet collisions.

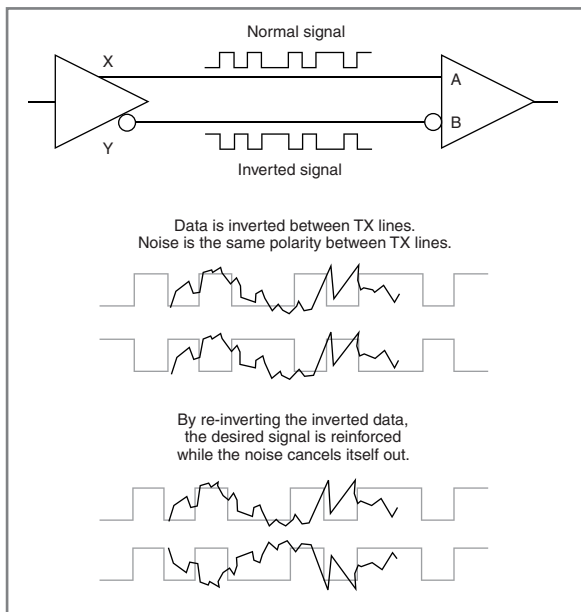


Figure 1—RS-485 uses differential signaling to eliminate noise. Any noise induced on the twisted pair transmission lines will be canceled out when the “B” channel is inverted at the receiver and summed with the “A” channel.

NETWORK TOPOLOGY

To help allow this, I redesigned my hardware nodes to use a “store and forward” scheme, where the node receives messages from downstream nodes on one port, and then forwards those upstream to the laptop using another physically separate port. If the node has to send a message at the same time, it will send its own message first and then send the stored message. This is a modified full-duplex topology, which is shown in [Figure 4](#). The physical implementation of a hardware node is shown in [Photo 3](#). Note the use of a two-port RJ-45 jack. One jack is designated as the upstream (closer to the master)

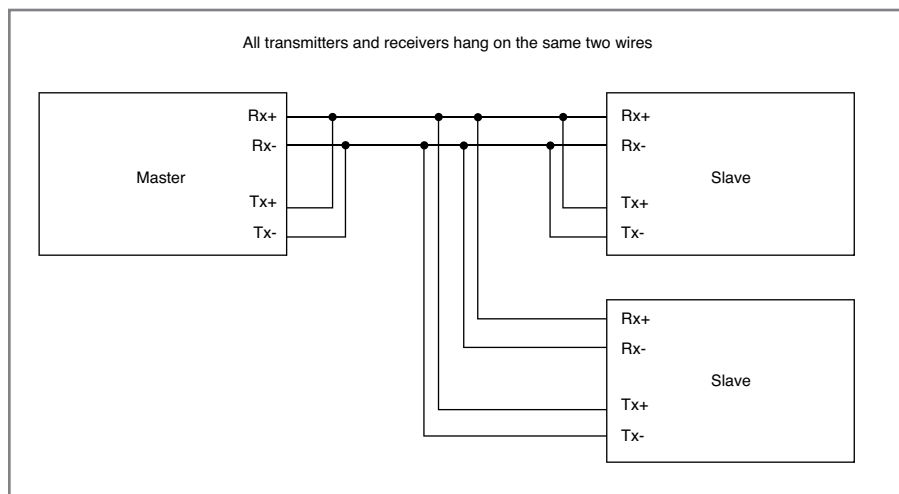


Figure 2—Many RS-485 systems use a simple half-duplex topology where all receivers and transmitters hang on the same wire pair. Software prevents more than one transmitter from operating at the same time.

port, and the other the downstream (farther from the master) port. **Photo 4** shows a pair of nodes daisy-chained together. Each hardware node has two receiver UARTs and RS-485 chips and one transmitter UART and RS-485 chip. By having in essence a dedicated link between each node heading back upstream to the master, any slave can transmit at any time without fear of collision since it is the only node on that dedicated link.

While it's a good system, there are numerous drawbacks. First, each node needs an extra RS-485 receiver chip—one for receiving upstream messages from the master, and one for receiving downstream messages from the node below it. This adds an extra chip to the design. It also requires more complicated software on each node, because each node must handle storing and forwarding messages from other nodes, along with interleaving messages from itself as needed. There is also a time delay introduced by each and every node in the system because a node has to completely receive a packet before sending it on. Finally, most small embedded processors are incredibly light on RAM, meaning that there isn't much space to buffer messages.

The time delay issue is subtle. It doesn't sound like it would be such a big deal until you do the math. With the Halloween network running at 9,600 bps, and the smallest network packet being 4 bytes long, receiving a packet takes about 4.2 ms. If I had 16 nodes daisy-chained together, sending a message from the farthest node to the laptop would take 67 ms. While that may not sound like much, it means that the farthest node can only send a maximum of 15 messages per second instead of 240. It also means that the delay time varies depending on where a node is located in the chain.

Up until last year, I'd been able to work around this issue satisfactorily. However, that changed when I created the Halloween Remote ("Dynamic Animatronic Remote," Parts 1 and 2, *Circuit Cellar* 218 and 219, 2008). Unlike most of the nodes in my system, the remote needs to send lots of

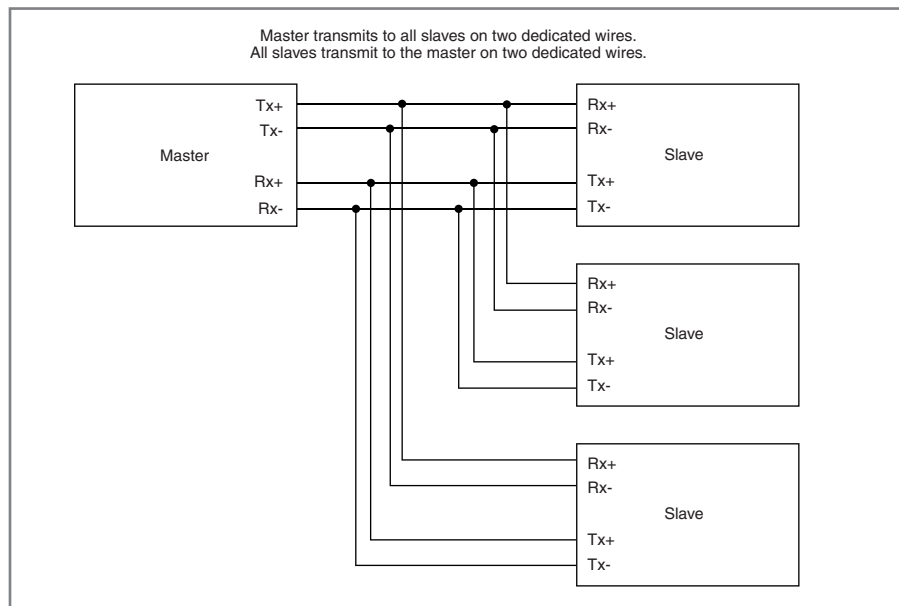


Figure 3—Full-duplex networks typically have a master/slave arrangement: one wire pair lets the master transmit to all slaves and the other pair lets the slaves report back to the master. Software ensures only a single slave transmits at a time.

packets back to the host since it is controlling programs on the laptop. Further, the messages aren't all tiny 4-byte messages. They can get much bigger. All of this puts a strain on the limited buffering space of the nodes. In addition, the remote itself doesn't store and forward

since that would require dragging two cables (one upstream and one downstream) around as I used it. Since the remote has only a single cable, I would be forced to plug the remote in as the system's last node. This is the node farthest downstream from the master and

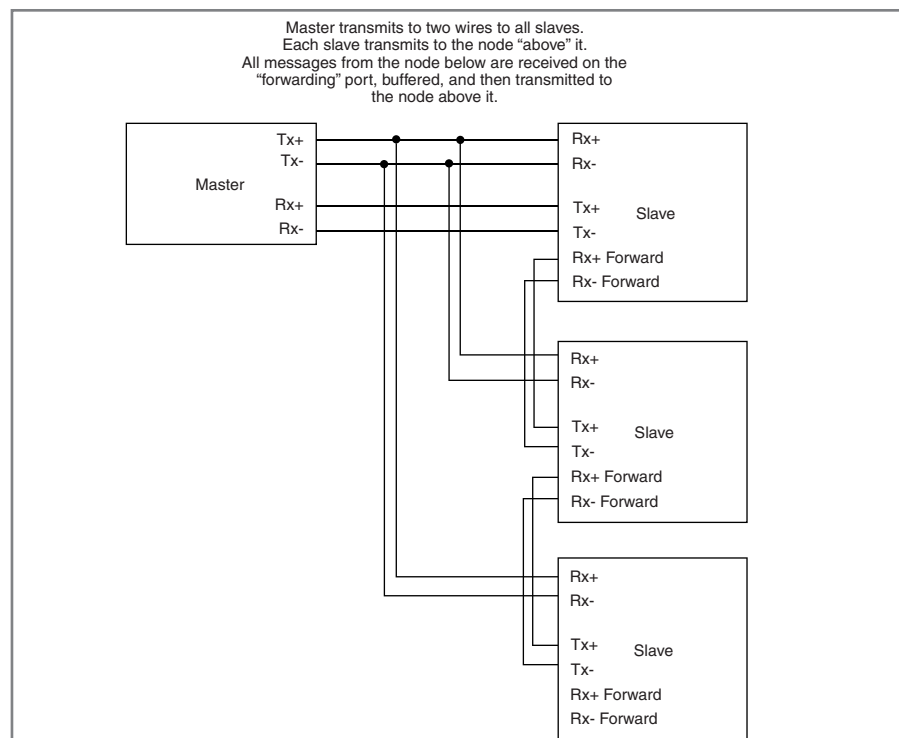


Figure 4—The network "store and forward" system is also master/slave. The extra hardware allows each slave to transmit at any time as all slaves buffer messages heading back to the master node.

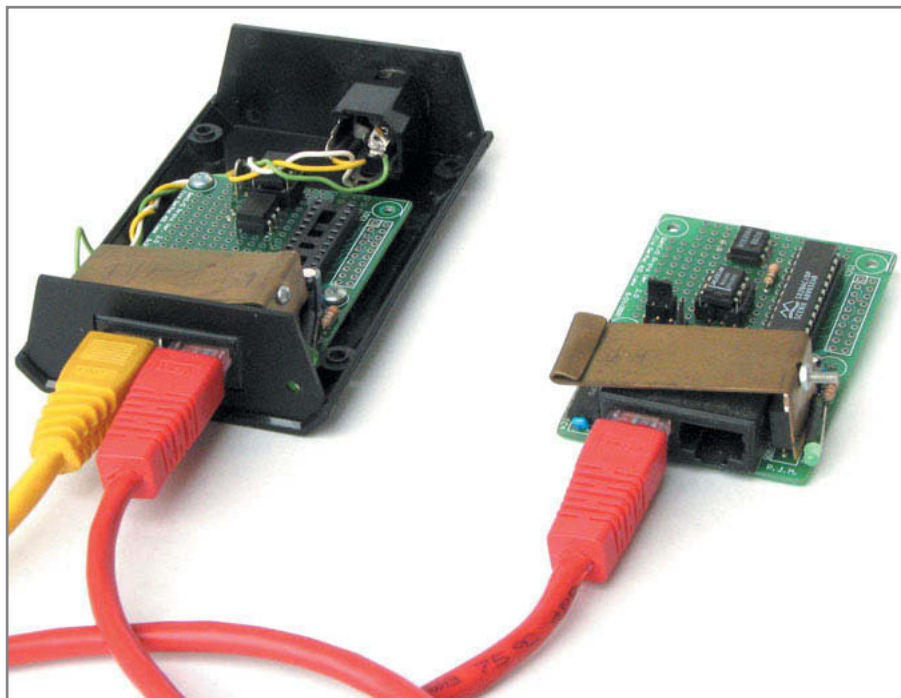


Photo 4—A pair of nodes is daisy-chained together. The left node is in the case with the top removed. The right one shows the “raw” board.

having the greatest delay time sending packets to the master. Forcing the remote to have the slowest response time and having to physically plug it in at the end of the node chain made using it problematic. I wanted to be able to plug it into some fast central location that made walking and using it easier.

HUB DESIGN SPECS

For all of these reasons, I decided the time was right to create the Halloween Serial Network Hub. This is a six-port hub that enables me to plug any node or string of nodes into a port, and it takes all incoming packets, buffers them as needed, serializes them, and then forwards them to the laptop. While there is a delay added, it doesn't accumulate with each node in the chain. My nodes have a jumper so they can operate in either Store and Forward mode, or simply daisy-chain them using the standard full-duplex approach. If I have a group of nodes that only need to transmit when polled, I can set them up in plain daisy-chain mode. This means that there is no additional delay no matter how many nodes are strung together. The sole delay is created by the network hub, and that delay will be just the length of a single packet if no other node is transmitting at that time.

In addition, I made the link between the laptop and the hub run at 38,400 bps. This means that when a packet is fully received by the hub, it then forwards it to the laptop four times faster than it was received. This helps minimize the added delay while letting the main network run at a slower, and more robust, 9,600 bps.

I had a certain criteria in mind when

I set out to design the hub. I wanted to write as much of the code as possible in C language to make my life easier. Also, I didn't want to have a pile of UART chips on the board. However, finding a six-port UART chip is somewhat tricky.

The hub also needs the UART connected to the laptop to run at RS-232 signal levels, since normal PCs and laptops don't come with RS-485 ports. The Halloween network uses standard CAT-5 cables to not only send and receive data, but also to provide 12-V DC power to run each node. I was sick of devices powered by wall warts and wanted to keep the entire unit self-contained. This meant fitting a small power supply into the case as well.

Finally, I wanted to create a small device that looked as professional as possible. This last requirement was more a personal challenge than an actual requirement; but since I use this equipment year after year, I found the extra time spent on making the packaging solid has paid off in equipment that hasn't been flaky in the field. My work was cut out for me.

In the second part of this article series, I'll explain exactly how I created my RS-485 hub. I'll include details about the software and hardware, as well as some hidden “gotchas” present in RS-485 chips. 📌

Peter Montgomery (PJMonty@csi.com) spent 12 years working as a visual effects supervisor on films such as Mortal Kombat and Ace Ventura: When Nature Calls before becoming a director. He has directed dozens of commercials and made the transition to episodic television with The Disney Channel's Lizzie McGuire. Peter is self-taught in both programming and digital hardware design.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2010/236.

SOURCES

NFM-20-12 Power supply

MEAN WELL Enterprises Co. | www.meanwell.com

SX-28 Proto Board

Parallax, Inc. | www.parallax.com

TLV1117 Linear power regulator

Texas Instruments, Inc. | www.ti.com

Z8F4802 Microcontroller

Zilog, Inc. | www.zilog.com