Serial Port Complete

Programming and Circuits for RS-232 and RS-485 Links and Networks

by Jan Axelson

ISBN 0-9650819-2-3

Published by Lakeview Research
2209 Winnebago St.
Madison, WI 53704
USA
Phone: 608-241-5824
Fax: 608-241-5848
Email: jan@lvr.com
WWW: http://www.lvr.com

1

Options and Choices

This book explores one corner of the computer universe: computers that are linked together to monitor and control the world outside themselves. Each computer can exchange information with the others, and each can also calculate, decide, and take action on its own. This type of link requires three things: computers to do the work, programming that tells the computers what to do, and a link to connect them. This chapter introduces options for each of these.

The Computers

Some projects need only a simple link between two computers, while others require three or more computers that connect along a common path. In this book, I use the term *link* broadly, to refer to a connection between two or more computers, while a *network* is a link of at least three computers. Each computer in a link is a *node*, or junction. Usually, each node can both send and receive, though in a simple link some nodes may communicate one-way only.

In the types of link described in this book, the computers may read sensors, switches, or other inputs. They may control motors, relays, displays, or other outputs. Because the computers can communicate with each other, the result is an integrated, intelligent system that enables one computer to react to or control events at another.

The computers may be of any type, and they may be all the same, or a combination. This book focuses on two categories: personal computers and embedded controllers.

A personal computer (PC) may be a desktop machine or a laptop, notebook, or subnotebook. The examples in this book use the family of computers that has evolved from the IBM PC, including the models XT, AT, ’386, ’486, Pentium, and their many clones and compatibles. But you can use any personal computer that has an appropriate serial interface.

An embedded controller is a computer that’s dedicated to performing a single task or a set of related tasks. Embedded controllers tend to be smaller and less complex than PCs. Many are built into, or embedded in, the devices they control. An embedded controller may have no keyboard or display and may be invisible to its users. For example, PC peripherals such as printers and modems contain embedded controllers that enable the peripherals to handle much of the work of printing or communicating over the phone lines on their own.

Many embedded controllers have nothing at all to do with PCs. Cars, video-cassette recorders, and microwave ovens are a few everyday items that contain embedded controllers. Embedded controllers are also popular for one-of-a-kind or small-scale, custom projects that involve simple control or monitoring tasks.

The CPU, or computer chip, in an embedded controller may be the same microprocessor found in PCs, or it may be a microcontroller, which is a computer chip designed specifically for use in control tasks.

Microcontroller chips come in many varieties: 8-bit chips have an 8-bit data path and are popular for use in monitoring and control links, but 4-, 16-, and 32-bit chips are also available. Different chips have different features and abilities, including serial ports of various types, varying amounts of memory for storing programs and data, and low-power modes for battery-powered circuits. A monitoring and control link can use any microcontroller that can connect to the desired interface.

The examples in this book use two microcontrollers: Parallax's Basic Stamp and Intel/Micromint’s 8052-Basic. Both are inexpensive and have on-chip Basic interpreters for easy programming and debugging.

One category of embedded controllers straddles both worlds. The embedded PC has the architecture of a PC, but in a stripped-down form that may lack a full-screen display, keyboard, or disk drives. Embedded PCs are popular because they can use many of the PC’s familiar programming tools.

The Programming

Each computer in a link must do each of the following:

- Detect communications intended for it.
- Send communications at appropriate times.
- Ignore communications intended for other nodes (if any).
- Carry out any other tasks the computer is responsible for on its own.

The computer's programming is responsible for each of these, with some assistance from the hardware.

Languages and Operating Systems

The program code may vary from one node to another, because the computer type and programming language may vary, and also because different nodes may have different functions.

On a PC, the program is software stored on disk. To run the program, the operating system loads it into the system's memory (RAM). In all but some embedded PCs, the user interface includes a keyboard and display.

On a microcontroller, the program is in *firmware*, which is program code stored in an EPROM or other nonvolatile memory chip. The microcontroller may run the program directly from where it is stored, without requiring an operating system to load the program into RAM or manage other operations.

The computers may use any programming language. The only requirement for communications is that all must agree on a format for data on the link.

Message Properties

Although there are many types of monitoring and control links, the communications in a link tend to have the following in common:

Messages are short, ranging from a byte or two in a very simple system to hundreds of bytes in others. A computer in this type of link isn't likely to send Megabytes of data at once.

Messages may require a quick response. In some links, a message may carry emergency information (*The motor is stuck! The door is open!*) and the receiving computer will need to respond quickly, either by taking direct action or by instructing another computer to handle the problem.

The frequency of messages may vary. In some links, a computer may send or receive many messages within a second. In others, a computer may go a day or a week without sending or receiving anything.

The communications protocol and message format are two ways that the programming ensures that each node recognizes and understands the messages directed to it.

Protocols

A protocol is a set of rules that defines how the computers will manage their communications. The protocol may specify how data is formatted for transmitting and when and how each node may transmit.

PCs and many microcontrollers have built-in components (UARTs) that handle many of the details automatically, or with limited program assistance.

When there are just two devices, the rules need to specify whether both ends can transmit at once, or whether they need to take turns. With three or more devices, things become more complicated. Because all nodes usually share the same path, each device has to know when it may transmit, as well as whether a received communication is meant for itself or another node.

Besides the data path, a link may use additional lines to indicate when a transmitter has data to send, when a receiver is able to accept new data, or other control or status information. The process of exchanging status information about a transmission is called *handshaking*. The control and status signals are handshaking signals. Hardware handshaking uses dedicated lines for the signals. Some links use software handshaking, which accomplishes the same thing by sending special codes in the data path.

Message Format

A message is a block of data intended for one or more receivers. The message format defines what type of data the message contains and how the data is arranged within the message. All nodes have to agree on a format.

When there is more than one receiver, the receivers need a way to detect which node is the intended receiver. For this reason, network messages usually include the receiver's address. In a very simple network, a message may consist of just two bytes: one to identify the receiver and another containing data.

Messages may include other information as well. To enable receiving nodes to detect the start and end of a message, the message may include codes to indicate these, or bytes specifying the length of the message. A message may also include one or more bytes that the receiving node uses in error-checking.

The Link

The physical link between computers consists of the wires or other medium that carries information from one computer to another, and the interface that connects the medium to the computers.

The requirements of a link help to determine which interface to use and what medium to use to connect the nodes. In the types of systems described in this book, the distance between computers may range from a few feet to a few thousand feet. The time between communications may be shorter than a second, or longer than a week. The number of nodes may range from two to over two hundred.

Most links use copper wire to connect computers, often inexpensive twisted-pair cable. The path may be a single data wire and a ground return, or a pair of wires that carry differential signals. Other options include fiber-optic cable, which encodes data as the presence or absence of light, and wireless links, which send data as electromagnetic (radio) or infrared signals in the air.

For most projects, there is a standard interface that can do the job. Most of the links described in this book use one of two popular interfaces: RS-232 for shorter, slower links between two computers, or RS-485 for longer or faster links with two or more computers.

An interface may use existing ports on the computers, or it may require added ports or adapters. Most PCs have at least one RS-232 interface, and an RS-232 or RS-485 interface is easily added to a PC or microcontroller.

Table 1-1 compares RS-232 and RS-485 to other interfaces that a monitoring or control system might use.

RS-232 is popular because it’s widely available, inexpensive, and can use longer cables than many other options. RS-485 is also inexpensive, easy to add to a system, and supports even longer distances, higher speeds, and more nodes than RS-232.

The IrDA (Infrared Data Association) interface can use the same UARTs and data formats as RS-232 (with added encoding), but the data transmits as infrared energy over a wireless link. IrDA is useful for short, line-of-sight links between two devices where cabling is inconvenient.

MIDI (Musical Instrument Digital Interface) is used for transferring signals used by musical instruments, theatrical control equipment, and other machine controllers. It uses an optically isolated 5-milliampere current loop at 31.5 kbps.

Microwire, SPI, and I^2C are synchronous serial interfaces that are useful for short links. Many microcontrollers have one or more of these interfaces built-in.

Table 1-1: Comparison of popular computer interfaces. Where a standard doesn't specify a maximum, typical maximums are listed.

Interface	Format	Number of Devices (maximum)	Length (maximum, feet)	Speed (maximum, bits/sec.)
RS-232 (EIA/TIA-232)	asynchronous serial	2	50-100	20k (115k with some drivers)
RS-485 (TIA/EIA-485)	asynchronous serial	32 unit loads	4000	10M
IrDA	asynchronous serial infrared	2	6	115k
Microwire	synchronous serial	8	10	2M
SPI	synchronous serial	8	10	2.1M
I^2C	synchronous serial	40	18	400k
USB	asynchronous serial	127	16	12M
Firewire	serial	64	15	400M
IEEE-488 (GPIB)	parallel	15	60	1M
Ethernet	serial	1024	1600	10M
MIDI	serial current loop	2	15	31.5k
Parallel Printer Port	parallel	2, or 8 with daisy-chain support	10-30	1M

USB (Universal Serial Bus) and Firewire (IEEE-1384) are new, high-speed, intelligent interfaces for connecting PCs and other computers to various peripherals. USB is intended to replace the standard RS-232 and Centronics printer ports as the interface of choice for modems and other standard peripherals. Firewire is faster and designed for quick transferring of video, audio, and other large blocks of data.

Ethernet is the familiar network interface used in many PC networks. It's fast and capable, but the hardware and software required are complex and expensive compared to other interfaces.

The alternative to serial interfaces is parallel interfaces, which have multiple data lines. Because parallel interfaces transfer multiple bits at once, they can be fast. Usually there is just one set of data lines, so data travels in one direction at a time.

Over long distances or with more than two computers in a link, the cabling for parallel interfaces becomes too expensive to be practical.

The Centronics parallel printer interface predates the PC and just about every PC has included a Centronics-compatible interface. The IEEE-1284 standard defines new connectors, cables, and high-speed protocols for the port's 17 lines. Because the interface has been standard on all PCs, it's been pressed into service as an interface for scanners, external disk drives, data-acquisition devices, and many other special-purpose peripherals.

IEEE-488, which began life as Hewlett-Packard's GPIB (General-purpose Interface Bus) is another parallel interface popular in instrumentation and control applications.

Applications

This book focuses on what you need to design and program serial links. It doesn't get into application-specific details such as how to interface and access sensors, motors, and other devices that connect may to a computer in a monitoring or control link; these are topics for another time, and another book. But to give an idea of the possibilities, this section is an overview of the kinds of things you can do with these links.

One way to categorize links is by direction of data flow. In some systems all computers send and receive more or less equally. In others, most of the data flows to or from a central computer. For example, most of the activity in a link may relate a computer's collecting data from remote locations.

An everyday example of a system that collects data is a weather-watching network. A desktop PC may serve as a master that controls the activities of a variety of remote computers, which may simple microcontrollers. The master sends commands to the remote computers to tell them how often to collect data, what data to send to the master, and when to do it. The data collected may include temperature, air pressure, rainfall, and other variables. At intervals, each site sends its collected data to a master computer, which stores the data and makes it available for further viewing and processing.

This basic setup is adaptable to many other types of data-gathering systems. You can find a sensor to measure just about any property. Table 1-2 lists a variety of sensor types.

Other systems are mainly concerned with controlling external devices, rather than gathering data from them. A store-window display may include a set of mini-robots, each with switches and signals that control motors, lights, and other

Table 1-2: Types of Sensors

Acceleration	Flow	Moisture	Temperature
Chemical content	Force	Position	Thickness
Color	Level	Pressure	Velocity
Density	Light	Radiation	Vibration
Distance	Magnetic properties	Sound	Weight
Electrical properties	Mass	Strain	Wind

mechanical or electrical devices. Again, each device may have its own computer, with a master computer controlling the show by sending commands to each of the robot's computers. The robots may also return information about their current state to the master computer, but the main job of this type of system is to control the devices, rather than to collect information from them. This arrangement is typical of many other control systems.

An example of a system involved equally with monitoring and controlling is a home-control system, which may watch temperature, humidity, motion, switch states, and other conditions throughout a house. Control circuits hook into the house’s heating, cooling, lighting, and alarm systems. When the master computer detects that a room has strayed from the set temperature, it causes more heated or cooled air to be pumped into the room. When alarm circuits are enabled and motion is detected, the system generates an alarm. The system may also control audio and video systems and outdoor lighting and watering.

In each of the examples above, one computer may act as a master that controls a series of slave computers whose actions are controlled by the master. A slave transmits only after the master contacts it and gives it permission.

It's also possible to have a system with no master. Instead, each computer has equal status with the others, and each can request actions from the others. For example, each computer may take turns transmitting to the others. Or one computer may send a message to another, which in turn can pass the same message, or a different message, to another computer. In some links, any computer may try to transmit at any time, and a protocol determines what happens if two try to transmit at once.

A simple link may use just two computers. One may gather data from or send commands to another. Or two computers may each be responsible for various monitoring and control functions, sharing information as equals.

These are just a few examples. By choosing components and writing programs to control them, you can put together a system to serve whatever purpose you have in

mind. The rest of this book is devoted to presenting what you need to make this happen.

Serial Port Complete. Introduction. I developed the PC examples with Visual Basic 5. Because they're intended as design tools, and not as finished applications, I provide the complete source code but not compiled, executable programs. To compile the programs, you must have a copy of Vis ual Basic. I tested the code on a system running Windows 95.Â Serial Port Complete. Chapter 1. The computers may be of any type, and they may be all the same, or a cornbination. Preview â€" Serial Port Complete by Jan Axelson. Serial Port Complete: Com Ports, USB Virtual Com Ports, and Ports for Embedded Systems. by. Jan Axelson. really liked it 4.00 Â· Rating details. Â· 12 ratings Â· 0 reviews.Â Topics include using .NET's SerialPort class for COM-port communications on PCs; upgrading existing RS-232 designs to USB or wireless networks; and creating serial networks of embedded systems and PCs.

Access serial ports with JavaScript. Linux, OSX and Windows. Welcome your robotic JavaScript overlords.Â README.md. Node Serialport. Access serial ports with JavaScript. Linux, OSX and Windows. Welcome your robotic JavaScript overlords. In computing, a serial port is a serial communication interface through which information transfers in or out sequentially one bit at a time. This is in contrast to a parallel port, which communicates multiple bits simultaneously in parallel. Throughout most of the history of personal computers, data has been transferred through serial ports to devices such as modems, terminals, various peripherals, and directly between computers. Serial Port Complete. Cables 146. Length Limits How Many Wires? Isolated Links 148. Ways to Achieve Isolation About Grounds. Power Supply Grounds Optoisolating. Surge Protection.Â Serial Port Complete. A Simple Stamp Network 259 Debugging Tips 275. Appendices. A Resources 287. B RS~232 Signals 291. C Number Systems 293. Index 2.97. Serial Port Complete. Introduction Acknowledgments 1 Options and Choices 2 Formats and Protocols 3 COM Ports on PCs 4 Inside RS-232 5 Designing RS-232 Links 6 Inside RS-485 7 Designing RS-485 Links and Networks 8 Going Wireless 9 Using .NETâ€™s SerialPort Class 10 Managing Ports and Transfers in .NET 11 Ports for Embedded Systems 12 Network Programming 13 An RS-485.Â Janâ€™s book is about as complete a reference as youâ€™ll find on serial communications using RS-232 and RS-485. So I read from the serial port like so. while (running !=0) {. int n = read (fd, input_buffer, sizeof input_buffer)Â The problem I am having is that I need to get a complete message. Half of a message could be in the buffer one iteration. And the other half could come into the message the next iteration. Somebody suggested that free the buffer up from the complete message. And then I move the rest of data in the buffer to the beginning of the buffer.