# *Networking and Internetworking with Microcontrollers*

*By Fred Eady*

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Newnes is an imprint of Elsevier

# *Networking and Internetworking with Microcontrollers*

Newnes is an imprint of Elsevier
200 Wheeler Road, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK



Recognizing the importance of preserving what has been written, Elsevier prints its books on acid-free paper whenever possible.

**Library of Congress Cataloging-in-Publication Data**

(Application submitted.)

**British Library Cataloguing-in-Publication Data**
A catalogue record for this book is available from the British Library.

ISBN: 0-7506-7698-1

For information on all Newnes publications
visit our website at www.newnespress.com

03 04 05 06 07 08 10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

# *Contents*

# *Preface*

There are lots of philosophical things I could say here. However, I don't claim to be a philosopher or a poet. My days are spent designing microcontroller hardware, writing code to drive that hardware and then writing about my adventures.

This book is a mean-business document designed to give you the knowledge needed to network microcontroller-based devices successfully. Before you turn the last page of this book, you'll know how to integrate RS-232, I²C and Ethernet into a network device that can be used to communicate via LAN, WAN or Internet. In addition to the knowledge you will gain building the network devices, you'll also walk away with in-depth knowledge of how the code within those network devices works.

Our microcontroller-based network devices will be fabricated using microcontrollers from Atmel and Microchip. To maintain consistency at the coding level, I'll use ImageCraft's ICCAVR Pro C Compiler for the Atmel parts and Custom Computer Service's CCS PIC C Compiler for the PIC parts. Both of these C compilers are moderately priced and easily obtainable via the Internet. You should be able to easily port the C source code from any project in this book to other variants of C.

Our networking adventure will begin with RS-232. We'll build on what we learn in the RS-232 sections and ultimately implement both an I²C-bus and an Ethernet interface. No bit will be left unturned. What you don't see in the pages of this book can be found on the companion CD-ROM. There's also a support web site (http://www.edtp.com) where you can get technical support and purchase parts, kits and assembled units that are discussed in this book.

Both Atmel and Microchip provide a free IDE that you can get for a download from their respective web sites. I'll use Atmel's AVR Studio and Microchip's MPLAB exclusively when working with these microcontrollers. To provide an extra layer of visibility into the microcontrollers, I'll employ the services of a Microchip MPLAB ICE 2000, a Microchip MPLAB ICD 2 and an Atmel AVR JTAG ICE. On the networking side, I'll use a Network Associates Sniffer to show you what's inside the Ethernet packets.

OK…now that you know what this book is about, let's go build some microcontroller-based network devices.

## A Quick Look at the Microcontrollers

### Atmel's AVR

The Atmel AVR is a very capable and highly networkable microcontroller. There are a number of AVR families, which include the standard AVR line, a low-power AVR microcontroller set, the tinyAVR family and the ATmega AVR microcontrollers. I've chosen to concentrate on networking the ATmega AVR microcontrollers for a number of reasons. Many of the legacy AVRs are being replaced by faster and more powerful ATmega AVRs. For instance, the ATmega16 has replaced the ATmega163 and the ATmega32 has shoved the ATmega323 out. In addition to added functionality, the AVR upgrades fix bugs found in the older silicon they are replacing.

I'm not going to get into internal differences found in the AVR versus other microcontrollers that could be called AVR peers. That's what datasheets are for. However, I will give you my reasons for employing the ATmega AVRs as microcontrollers in network devices. The ATmega AVRs that I will network all have a maximum clock speed of 16 MHz. That may not sound "fast," but the ATmega AVRs execute most instructions in a single cycle and thus are capable of producing 16 MIPS with a 16MHz clock. Basically, the number in the name of a ATmega AVR represents the amount of program Flash in kilobytes. The ATmega16 contains 16K of program Flash while an ATmega32 has 32K words of program flash. The largest ATmega AVR, the ATmega128, contains 128K of program Flash memory. The large portions of program memory are supplemented by big slices of SRAM. The ATmega16 is loaded with 1K of SRAM and the ATmega32 SRAM area doubles the ATmega16 SRAM capacity. Even the ATmega8, the smallest of the ATmega AVRs, has 1K of SRAM. Applying the logic to the rest of the ATmega AVR family reveals the ATmega64, ATmega128 and ATmega8 with 64K, 128K and 8K of program Flash memory, respectively. I think you can see why I've decided to go with Atmel's ATmega AVR line as far as networking is concerned. The high speed and large program Flash memory areas coupled with ample SRAM and EEPROM memory make the ATmega AVR microcontroller a good choice for networking projects.

Atmel's AVR can be obtained from many of the mail order electronic part distributors. AVRs are reasonably priced and come with a tub full of goodies just right for networking. Two 8-bit timers and a 16-bit timer allow the creation of precision delays while an on-chip USART (Universal Synchronous Asynchronous Receiver Transmitter) takes care of the housekeeping chores needed to effect the RS-232 serial protocol. Twenty-one interrupt vectors cover all of the AVR's networking components including the two-wire interface (Atmel's name for $I^2C$), the SPI subsystem and the USART.

### *The AVR USART*

The ATmega AVR USART is capable of full duplex operation. Like most every other USART in existence, the ATmega AVRs USART supports 5, 6, 7, 8 or 9 data bits plus the standard 1 or 2 stop bits. The USART baud rate generator for the Atmel ATmega AVR is an integral part of the USART hardware. A typical Atmel USART is depicted in the block diagram you see inside Figure 1. All ATmega AVRs contain a USART and the ATmega128 is equipped with a pair of USARTs.

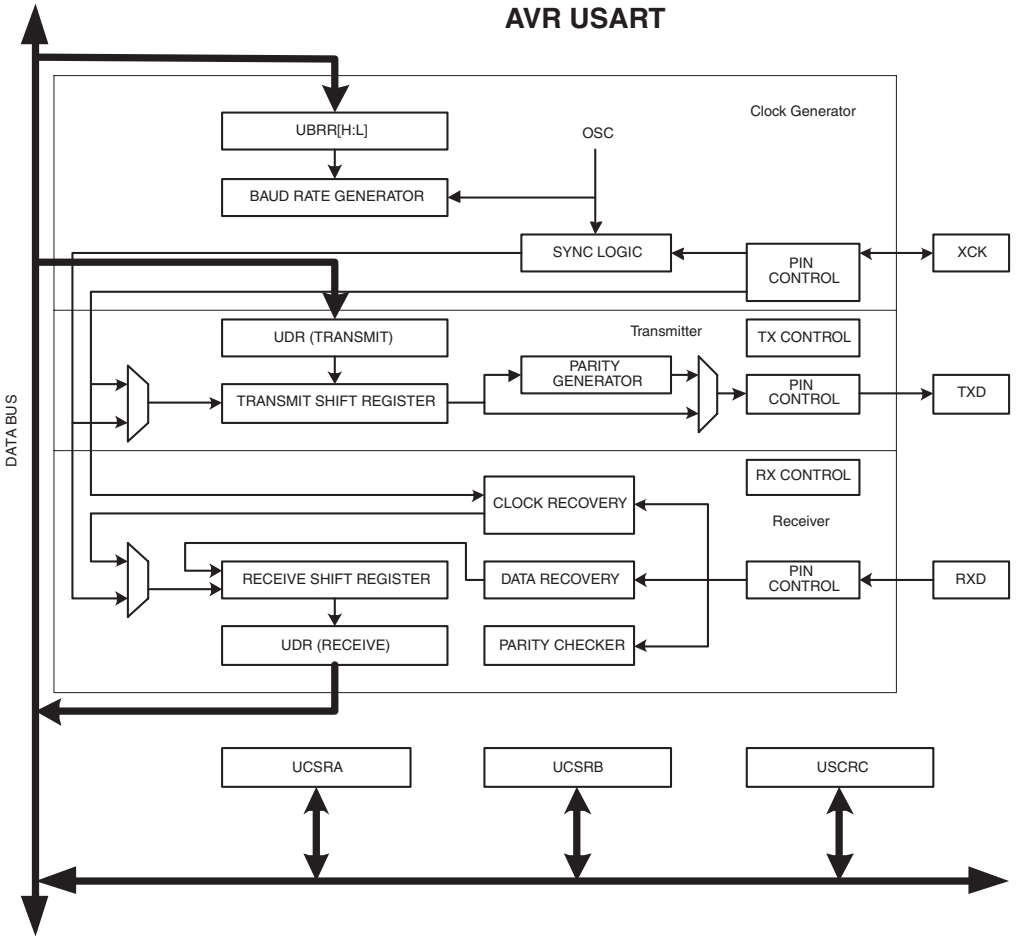


*Figure 1*

Atmel USARTs allow the ATmega AVRs to enter MPCM (Multi-processor Communication Mode). This mode of operation uses addressing to allow multiple processors to communicate over the same serial bus. MPCM uses the 9-bit character frame format in conjunction with the master/slave paradigm.

### *The Two-wire Serial Interface*

In the Atmel world, I²C is known as TWI, or Two-wire Interface. Other than a name change, TWI looks like and smells like I²C. 128 devices can hang on the two-wire bus and are addressed using the standard I²C 7-bit addressing scheme. Master and slave operation is supported at speeds of up to 400 kHz. To help fight false triggering due to noise, the Atmel TWI module includes noise suppression circuitry. You can even wake up the AVR from sleep with a TWI.

### *Programming the ATmega AVR*

Loading code into an ATmega AVR device is a breeze. There are many ways to accomplish this. There's the AVR ISP (In-System Programmer) programming module that costs less than $40 and hooks up to a personal computer's serial port. Or, AVR programming can be done with the STK500 development board. ATmega AVRs with 16K or more of program memory also support a JTAG interface, which can be used for programming the ATmega AVR program Flash. No matter how you decide to load the code into your ATmega AVR, AVR Studio supports all of the programming devices I've mentioned. AVR Studio is Atmel's front-end IDE software that runs on a personal computer.

I'll complement AVR Studio with ImageCraft's ICCAVR Pro C Compiler. ICCAVR Pro is a true ANSI-based C compiler for the ATmega AVR. I particularly like the code generator and the AVR calculator features of ICCAVR Pro.

### *Emulating the ATmega AVR*

At the helm of emulation for AVRs is AVR Studio. AVR Studio interfaces to the many AVR emulation devices. In this text, the emulation device of choice is the AVR JTAG ICE. The AVR JTAG ICE communicates with an on-chip debug module embedded within the target AVR. The OCD (On-Chip Debugger) module in the ATmega AVRs eliminates the need for a special bondout emulation device.

## Microchip's PIC

Most PIC microcontrollers have everything one would need to effect a network application. Larger PICs have on-chip UARTS and USARTS for synchronous and asynchronous communications using the RS-232 protocol. A software UART function can also be implemented for the smaller PICs that don't have the sophistication of a built-in UART or USART module.

In networking, timing is everything. Up to three internal timers can be had on larger PIC devices. Even the tiny 8-pin PIC12F675 has an 8-bit and a 16-bit timer. The timers can be used for generating precision millisecond and microsecond delays or the time of day.

Fortunately, the CCS C Compiler for PIC and its native PIC peripheral routines makes it very easy to assemble a working RS-232 PIC application. In fact, CCS C also has hooks for I²C. The Microchip PIC family complements the CCS C peripheral routines by providing ample Flash memory for program code, scratch pad SRAM and user data storage. The more SRAM the better when it comes to creating buffer areas for interrupt-driven communications applications.

All of our network coding and hardware design and fabrication time will be spent dealing with the Flash-based series of the Microchip PIC family. I've chosen to work with the Flash-based parts because they're inexpensive and easily obtained and don't require the support hardware a standard windowed PIC needs. For instance, using Flash devices eliminates the need for an ultraviolet EPROM eraser. And, since Flash parts can be programmed and reprogrammed in-circuit using ICSP (In-Circuit Serial Programming), fewer microcontroller parts are needed in the development cycle since there is no need to rotate a number of parts through the ultraviolet eraser while you're debugging your code.

For the purposes of networking, I've selected the largest part in the PIC16F87X crew, the PIC16F877. The PIC16F877 can operate with a 20 MHz clock, which gives an instruction cycle of 200 nsec. There are 8K words of program Flash and 368 bytes of SRAM or data memory inside a PIC16F877. Should we decide it's necessary, there is also a block of 256 bytes of EEPROM available for storing constants or whatever else we decide is important to keep even after the power is removed from the part. As we move into putting an RS-232 serial port together on a PIC, you'll see how important interrupts are when it comes to microcontrollers like the PIC. The PIC16F877 can be interrupted in 15 different ways.

I/O pins are also very important in a networking application. Not only do we need enough I/O to perform tasks like monitoring a voltage or turning an external device on or off, there have to be some I/O pins dedicated to the networking task. For instance, a simple micro- controller Ethernet driver application requires at least 16 I/O pins alone. The PIC16F877 has 33 I/O lines we can put to work, which leaves some I/O for things that microcontroller do best—control.

The PIC16F877 offers quite a bit of functionality for things other than effecting networking. However, I'm primarily concerned with giving you the ability to network the PIC16F877. With that, let's start with a look at one of my favorite networking modules, the PIC16F877 USART.

### *The PIC16F877 USART*

USART is short for Universal Synchronous/Asynchronous Receiver/Transmitter. On the PIC, the USART is also called the SCI or Serial Communications Interface. You probably have heard the word UART (Universal Asynchronous Receiver/Transmitter) as for many years that was the only IC used by serial ports in personal computers. Some of today's microcontrollers sport UARTs instead of USARTs.

The PIC16F877 USART takes much of the pain away when it's required to communicate with other serial-based devices. Instead of writing timing routines to produce a specific baud rate, the PIC16F877 USART baud rate is generated by an internal baud rate generator. With a USART or UART, it's not necessary to code routines to look for incoming start bits or time the inter-bit distances to pick up the incoming data. All of that work is done within the USART itself. A USART makes it possible to communicate with other serial devices in full-duplex or half-duplex mode. Full-duplex mode allows communications to flow in both receive and transmit directions simultaneously between two serial devices. Half-duplex mode only allows one device to transmit at a time while the other device listens.

### The PIC16F877 MSSP Module

MSSP, or Master Synchronous Serial Port, is yet another PIC16F877 communications subsystem. The MSSP is a serial interface used to bring I²C applications to life. Like the USART, the MSSP is a register and status bit-oriented module.

I²C uses six MSSP registers for control, status and buffering. Two PIC16F877 I/O pins are dedicated to I²C, RC3 for SCL (clock) and RC4 for SDA (data). Like the USART's synchronous function, I²C is a master/slave communications configuration. Figure 4 is a graphic example of how the MSSP allocates the registers, I/O pins and buffers for I²C operation.

I²C is a Philips invention that was designed as a clever way to allow integrated circuits in television sets and stereo rigs to talk to each other. We'll cover I²C as it pertains to PIC microcontrollers thoroughly in this book. Thanks to Philips, there are hundreds of I²C-capable devices for us to play with from various manufacturers.

The PIC16F877 lends itself to oddball networking solutions. Using the PIC16F877 precision timers, we can put together a homebrew protocol and bit bang between devices. For instance, in the past I once coded a PIC application that required the PIC to clock data to and from a personal computer's parallel port pin. In addition, in the early days of PIC there were no UARTs or USARTs on the 18-pin PIC16C5X microcontrollers. Therefore, I had to code a "software" UART to emulate the task that today's hardware USARTs perform. You'll find that the software UART is still a good thing to have in your coding toolbox when designing networking and communications applications with the tiny USART-less 8-pin PICs.

### The PIC18F452

Another PIC device I'll base networking code on in this book is the PIC18F452. The PIC18F452 is pin-compatible with the PIC16F877. The PIC18F452 is loaded with 16K of on-chip program memory backed up by 1.5K of SRAM. This makes the PIC18F452 a candidate for Ethernet LAN applications. In addition to the increased internal memory area, the PIC18F452 can run twice as fast as the PIC16F877 (40 MHz). All of the PIC16F877 communications peripherals we talked about earlier operate in the same manner on the PIC18F452 and the CCS PIC C Compiler has the capability to generate code for them as well.

### The PIC12F675

Sometimes it's more fun to push an economy car to its limits and not drive that performance hot rod with all of the bells and whistles. That's how I feel about the little 8-pin PIC12F675. In comparison, it's as tiny physically as it is logically. The PIC12F675 only has 1K words of program Flash and 64 bytes of SRAM. There are only six I/O pins but inside the PIC12F675 you'll find a couple of timers, an A/D (Analog to Digital) converter and a comparator. Like the big guys, there is on-chip EEPROM but only 128 bytes of it. With some tricky coding, we'll make the tiny PIC do RS-232 with the best of them.

### *Programming the PIC*

The Flash-based PICs that will be featured in this book are all programmed using the ICSP (In-Circuit Serial Programming) method. As this book is focused on microcontroller communications and networking, I won't offer up any made-in-the-garage PIC programming hardware or software. I'm going to stick to the Microchip factory programmers and software. You can use the Microchip MPLAB ICD 2 (In-Circuit Debugger) or the Microchip PRO MATE II for programming the PIC Flash parts.

### *Emulating the PIC*

The Microchip MPLAB ICD 2 and the MPLAB ICE 2000 will be used to debug and display the inner-workings of the PIC code that will be presented in this book. I'll be able to show you all of the code, internal registers and memory areas using the Microchip MPLAB ICE 2000 PIC emulator system. Like the CCS C Compiler and the Microchip PRO MATE II device programmer, the MPLAB ICE 2000 and Microchip MPLAB ICD 2 are natively supported by Microchip's MPLAB. The merger of the Microchip PRO MATE II, the Microchip MPLAB ICE 2000, the Microchip MPLAB ICD 2 and the CCS PIC C Compiler will allow me to show you how things are done inside and outside the PIC using only a single MPLAB IDE screen.

# *What's on the CD-ROM?*

All of the source code and the executable code discussed in this book are on the companion CD-ROM. In addition, all of the Easy Ethernet device schematics are provided in PDF format. Printed circuit board layouts are also part of the CD-ROM package and are included for those readers who wish to build the Easy Ethernet devices from scratch.

CHAPTER 1

# The Essence of Microcontroller Networking—RS-232

Let's begin by exploring the RS-232 protocol. Knowing how to manipulate data with RS-232 will help you master more complex communications protocols. You'll also find RS-232 techniques to be invaluable in the development phase of your projects.

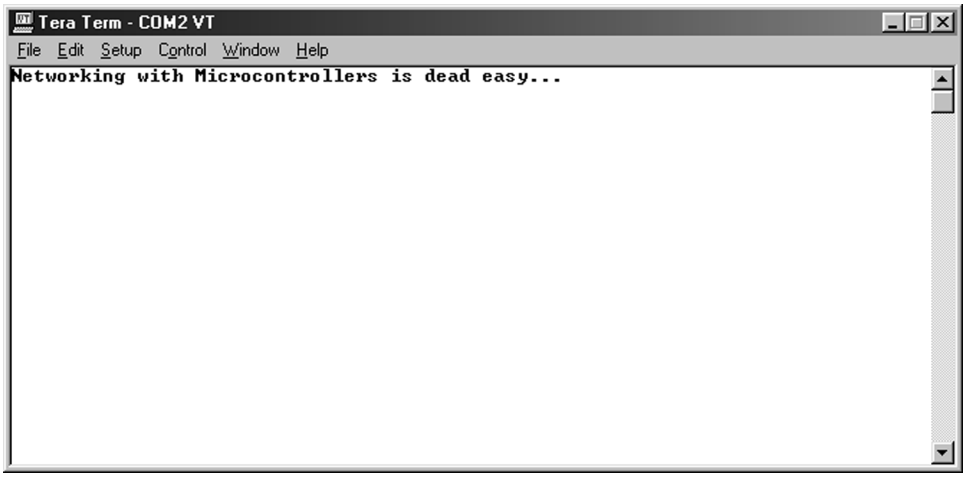


*Figure 1.1: Effecting RS-232 communications with a microcontroller is a snap. As you continue reading this book, you will find that knowing how to implement simple RS-232 with a microcontroller can assist you in building and debugging more complex microcontroller projects.*

The information you see in the terminal emulator window in Figure 1.1 was generated by some very simple firmware and a not-so-complicated off-the-shelf, two-buck microcontroller. I used a tiny 8-bit microcontroller that does not contain a built-in hardware USART (Universal Synchronous/Asynchronous Receiver/Transmitter), to transfer the ASCII characters you see in Figure 1.1 from one of its I/O pins to an RS-232 converter IC. A serial cable connected between the microcontroller/RS-232 converter IC circuitry and my personal computer's serial port allowed the ASCII characters to flow from the little microcontroller's firmware out of the microcontroller's I/O pin, through the RS-232 converter IC, across the serial cable to the personal computer's USART/RS-232 circuitry and finally end up in the terminal emulator window you see in Figure 1.1.

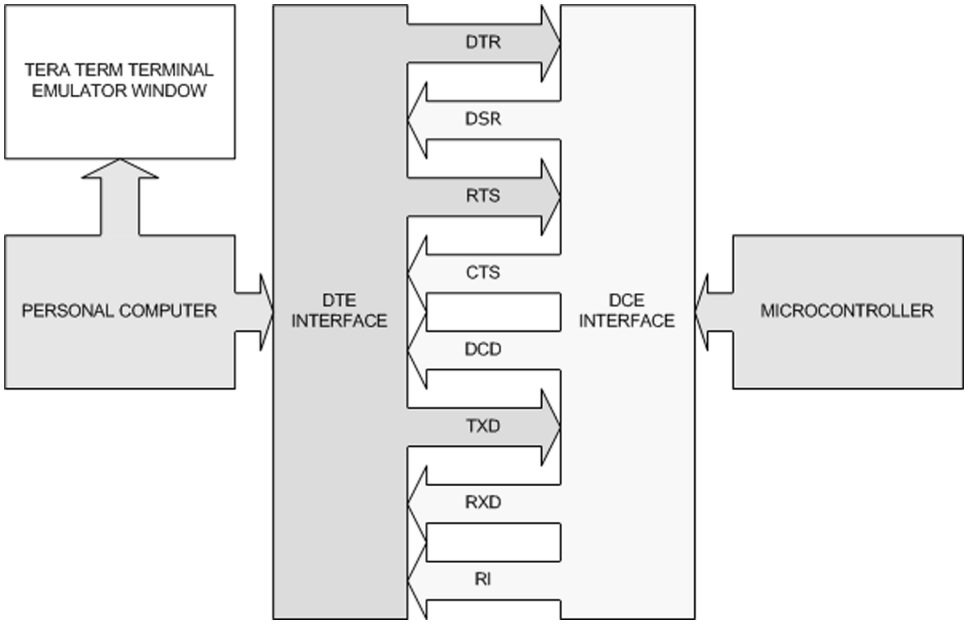


*Figure 1.2: The DTE and DCE interfaces usually consist of some sort of voltage-conversion circuitry to translate RS-232 voltage levels to voltage levels that are compatible with the computing equipment on each end of the communications link. The simplest form of an RS-232 link uses only the TXD and RXD signals with a common ground.*

What I've just described is one of the simplest forms of microcontroller networking. It is commonly known as serial or RS-232 communications. As you can see in Figure 1.2, RS-232 was designed to tie DTE (Data Terminal Equipment) and DCE (Data Communications Equipment) devices together electronically to effect bidirectional data communications between the devices.

An example of a DTE device is the serial port on your personal computer. Under normal conditions, the DTE interface on your personal computer asserts DTR (Data Terminal Ready) and RTS (Request To Send). DTR and RTS are called modem control signals. A typical DCE device interface responds to the assertion of DTR by activating a signal called DSR (Data Set Ready). The DTE RTS signal is answered by CTS (Clear To Send) from the DCE device. A standard external modem that you would connect to your personal computer serial port is a perfect example of a DCE device.

## Some History

In May of 1960, it was evident that a standard was needed to identify the electrical interface between computers and modems. It was decided to establish a standard voltage with standard signal parameters and a standard nomenclature to identify the conductors in the cable that connected computers and data sets. Even today, you will sometimes hear the term data set applied to modems and DCE equipment.

To compete as well as exist in the current communications environment, telecommunications vendors needed common ground to assure that each vendor's equipment set could talk to any other vendor's telecommunications equipment set. In other words, the industry needed a working standard. Without a standard, the whole teleprocessing industry could come to a grinding, nonstandardized halt.

To help establish some harmony, a committee named the Electronic Industries Association was formed. The EIA drafted a standard known as EIA RS-232(X). Though it was a great idea, the original specification was broad in meaning and didn't guarantee compatibility. The new RS-232 specification also had a competitor outside the United States, known as the CCITT, or Consultative Committee on International Telegraphy and Telephony, recommendation V.24.

The RS-232 proposal defined a logical and physical interface between DTE equipment and DCE equipment. The computer's DTE serial port presents both a physical and a logical interface to a modem or data set's DCE port and consists of several conductors for controlling, transmitting and receiving data. Timing and clocking signals are also intermixed within the RS-232 interface. The logical and physical attributes of the RS-232 proposal eventually became a set of standards known today as the EIA RS-232 interface.

Once the signals reach the DCE device, a second interface provides a physical path to the communication channel (RF link, telephone line, fiber-optic link, satellite link, and so forth). For most of you, that second interface is a standard two-conductor analog telephone line, which is terminated inside your modem.

The EIA standard originally identified seven interface conductors and no specific connector. Signal voltages were defined as at least 3 volts but not greater than 20 volts with respect to ground.

In October 1963, RS-232 became RS-232-A and was modified to include a 25-pin connector with a maximum cable length of 50 feet. This revision established fixed relationships between a circuit and specific pin numbers on the 25-pin connector. Also, an alphabetic coding system for each type of interface circuit was presented. The first character of the coding system designated A for ground, B for data, C for control and D for clocking. Table 1.1 lays out the pinout and various names for each RS-232 signal.