THE COMPUTER HACKER®

Volume I, Number 1 September 1983 $2.50 U.S.

EDITOR'S PAGE

Welcome to the first issue of THE COMPUTER HACKER, a magazine for those interested in building, interfacing, programming, and using microprocessor based devices.

When I first got my microcomputer, I was thrilled about all the things that I could do with it. I used a word processor, a spread sheet, and a data base program for my print shop, and learned to love working with the computer. But then I wanted to use the computer to control things in the real world and needed to use something besides prepared programs. I learned to program in Basic, and found that I needed assembly language. I took a course in assembly language, and found that I needed to learn how to interface to and control devices.

Up to this point I had been able to find the information I needed. There is a lot of information on Basic programming. Not all of it is good, but I could pick and choose to find what I wanted. There is some information on assembly language programming. Not much of it is very useful, but at least I could find enough to help. When I started searching for down to earth, hands-on, information on interfacing and control I hit a brick wall.

I looked at the shelves full of computer books, and the racks full of computer magazines, and said "With all this literature, what I need must be here somewhere." So I started searching for books and magazines which had what I needed. I found a few books which touched lightly on the subject, and an occasional magazine article which shed some light, but I couldn't find a periodical with ongoing current information.

We live in a rather isolated area, and I figured that what I wanted certainly must be available in "The Big City." Whenever I traveled, I asked anyone who had anything to do with computers for leads to the information. I got a lot of blank stares because they didn't understand what I was talking about, and the few who understood said "When you find it, let me know because I need it too." Finally someone said, "Why don't you publish a magazine for the Hacker?" I answered, "What? Another computer magazine? You gotta be kidding."

I kept on searching, but what I needed didn't exist. The magazines had nice four-color pictures of the latest $10,000 computers, stories on how to buy your first computer, and reports on the newest elegant, high cost, peripherals. But they didn't have what I wanted.

What is it about my needs that make it so unusual? I want to know how to program and control Stepper Motors. Not just buy someone's controller board and plug it in, but understand exactly what makes them tick. I want a directory of stepper motor controller chips with applications. I want a directory of Hacker priced sensors with how-to hands-on bench level application information. I want to hear from the people who are making things happen–what they're doing, what works, what doesn't work, and most important, WHY.

And Stepper Motors is just an example. I want to know HOW to set up a Timex 1000 or a microprocessor as a dedicated device to "sit on a process" so that I don't have to tie up my main microcomputer for control or monitoring purposes. This means that I have to understand various series and parallel interfacing methods, and how to apply them for data transfer and control.

Data transfer involves AD and DA converters, UARTs, and various interface adapter chips. I want to know how to apply them. Not just some prepared circuit to follow, but know how to apply them to my needs.

The initial response to The Computer Hacker has been very good, so apparently I am not the only one with these strange needs. We have some very interesting articles coming in, but we also need feedback from YOU. This is your magazine, and it will only be as good as YOU make it by sending articles and information. ■

Editor/Publisher........................Art Carlson
Art Director........................Joan Thompson
Technical Editor........................Lance Rose
Technical Editor........................Phil Wells

The Computer Hacker® is published 12 times a year. Annual subscription is $24 in the U.S., $30 in Canada, and $39 in other countries.

Entire contents copyright © 1983 by The Computer Hacker.

Postmaster: Send address changes to: The Computer Hacker, P.O. Box 1697, Kalispell, MT 59903-1697.

Address all editorial, advertising and subscription inquiries to: The Computer Hacker, P.O. Box 1697, Kalispell, MT 59903-1697.

THE RS-232-C SERIAL INTERFACE

Part One

by Phil Wells—Technical Editor

INTRODUCTION

Much myth, misinformation and misapplication surrounds the use of the RS-232C serial communications interface standard in the microcomputer industry. The standard is too general, was not written with micros in mind, and unnecessarily complex for most micro applications. Designers implement simplified versions of the standard and usually don't choose the same path to simplification.

The purpose of this tutorial is to explain what the RS-232C standard is and is not, to give examples of real micro-world implementation of the standard, and to propose a simplified version for use in Computer Hacker projects.

The RS-232C Standard

The RS-232C standard is a publication of the Electronic Industries Association (EIA). The name means "Recommended Standard" number 232, revision C. The full title is "Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange." The most recent revision was adopted in 1969. You can obtain a copy of the standard by writing to the address given in Table 2.

To discuss the standard, we need to agree on definitions of some of the terms used in the standard.

- **Serial Binary Data** means that the ones and zeroes (bits) are sent one at a time —in serial fashion—rather than eight bits at a time as with a parallel (e.g. Centronics) interface. This standard says nothing about data formats or codes. It only defines a way to send one bit at a time.
- **DTE or Data Terminal Equipment** is the hardware that produces or uses the information being communicated. This means a host or remote computer or terminal, a printer, a remotely controlled device, etc.
- **DCE or Data Communication Equipment** is the hardware that conveys the information between the pieces of Data Terminal Equipment. This usually means a MODEM. When two DTE's (Terminal Equipments) are connected by an RS-232C interface, one must play the role of a DCE or MODEM even if two DTE's are being directly connected. This arrangement can be simulated using a "Modem eliminator" described later.
- **Interchange or Interchange Circuit.** The standard uses this term to define each current path— a signal wire and its return wire. Simpler than it sounds.
- **Interface.** Simply the point of connection between two pieces of equipment.

The standard defines several things:

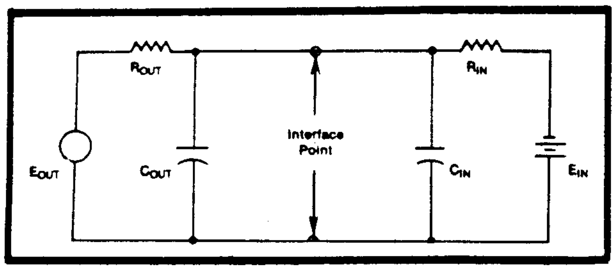
- Electrical characteristics.
- Mechanical characteristics.
- A functional description of data, timing and control circuits.
- Standard subsets of the interchange circuits for specific applications.

One reason the RS-232C standard is applied in such diverse ways when micros are connected to printers is that the entire document refers to a DTE-DCE-telephone line DCE-DTE communications link. That is, two computers talking to each other over telephone lines, with MODEMS coupling both machines to the phone lines. The standard defines the electrical connections between each computer and its MODEM. Many if not most micro applications of RS-232C are really misapplications since no DCE is involved; in these cases most of the standard makes no sense at all.

RS-232C Electrical Characteristics

The standard references the equivalent circuit shown in figure 1. It shows a driver side and a terminator side, connected at an "interface point." For the driver, we usually use an MC1488—or similar—integrated circuit, and for the receiver side, an MC1489A—or similar—integrated circuit. The driver generates an open-circuit voltage, E_{OUT}. The driver side of

Figure 1—Interchange Circuit.



the interface (including cable) has an output impedance, R_{OUT} and an output capacitance, C_{OUT}. The receiver or terminator side has an open-circuit voltage, E_{IN}, an input impedance and capacitance (again including cable) of R_{IN} and C_{IN}.

While it may not be obvious, an "Interchange Circuit" is one of the signal, timing or control wires, while the "Circuit AB, Signal Ground" is the one and only return wire for all three groups of signals. The RS-232C standard defines an "unbalanced" circuit; the newer RS422 standard defines a "balanced" type of interface, with a separate return wire for each signal.

The voltages, resistances, capacitances and unbalanced configuration of this equivalent circuit determines the electrical requirements and the limitations of an RS-232C implementation. For our use, the important electrical requirements and limitations are:

1. The driver (sending chip) must not be damaged by an open circuit or by a short to ground or to any other wire in the interface cable.
2. The terminator (receiving chip) must be able to tolerate 25 volts above ground and 25 volts below ground. Driver chips such as the 1488 are designed to meet these requirements; normal TTL circuits will not meet the spec.
3. The open-circuit voltage at the terminator must not exceed plus or minus two volts.
4. The terminator load impedance must be between 3,000 and 7,000 ohms.
5. The terminator load impedance must be non-inductive and the terminator's shunt capacitance must not exceed 2,500 picofarads.
6. The driver's open-circuit voltage must be limited to plus or minus 25 volts.
7. The driver's output impedance is not directly specified, but must be selected to limit the short-circuit output current to one-half ampere, and the power-off output impedance must not be less than 300 ohms.
8. The driver's output impedance should be selected such that the "one" and "zero" levels at the output of the driver are between 5 and 15 volts in magnitude.
9. A logical "1", or MARK or OFF condition exists when the voltage at the interface point is between −3 and −15 volts.
10. A logical "0", or SPACE or ON condition exists when the voltage at the interface point is between +3 and +15 volts. Since the driver must output at least ± five volts and the receiver must detect a "0" at +3 volts and a "1" at −3 volts, we have a two volt safety or "noise" margin on each side.
11. On some specific interchange circuits, an open connection or a power-off condition must be decoded by the receiving device as an "off" or "logic 1" condition. This is important because it determines what the software does if the connectors are unplugged or the power is off at one end.

The effects of the voltages specified above are summarized in Figure 2. Notice that 25 volts is the maximum allowable magnitude and that a legal "1" or "0" is between 3 and 15 volts in magnitude.

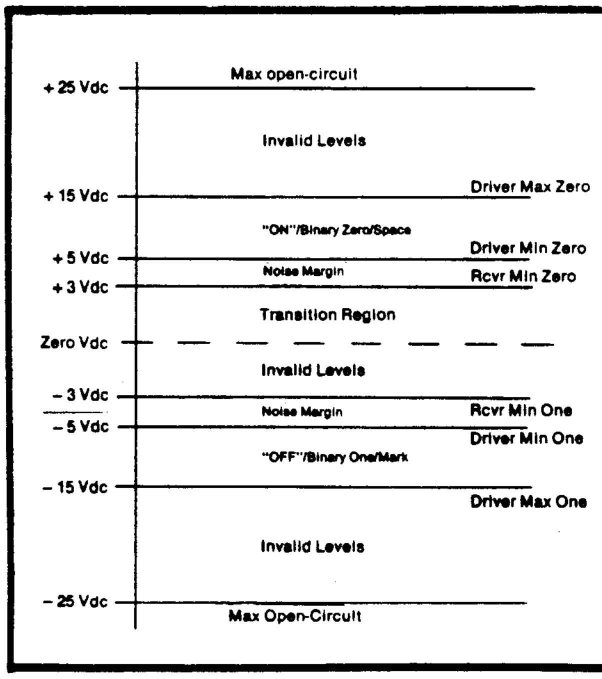


Figure 2—RS-232-C Voltage Levels.

The transition region, between ±3 volts, has some additional requirements: a signal must go cleanly through this region, without stopping or changing directions; a Control circuit signal must completely pass through this region within one millisecond; a Data or Timing signal must pass through in the lesser of one millisecond or four per cent of the nominal duration of a signal element on that interchange circuit; and the maximum instantaneous rate of change of voltage must not exceed 30 volts per microsecond.

Hacker's View of the Electrical Requirements

If you want to build a RS-232C compatible interface, the electrical requirements dictate at least the following:

- A 5 volt power supply doesn't cut it and standard TTL drivers and receivers can't be used. The voltages required are "bipolar"; they go both sides of ground. You must be able to swing the voltages at least ±5 volts from ground. Maximum open-circuit voltage allowed is 25 volts, and your drivers must be able to stand a short to ground or to another output. Maximum signal voltage is 15, each side of ground.
- Driver and receiver chips usually invert the signal since the standard defines a logical "1" as a negative voltage while in most popular microprocessor systems a logical "1" is the more positive of the two allowable states.
- The voltage and impedance requirements, as well as the A.C. characteristics, can most easily and cheaply be met using ±12 volt power supplies—which don't have to be regulated if they are well filtered, and by using the inexpensive and easily obtained MC1488 line driver and MC1489A line receiver IC's.
- The capacitance and (to a lesser extent) the resistance requirements mean that the cable length is restricted—although low-capacitance cables can extend the distance over which the interface can be used. The standard recommends that the combined lengths of cables be restricted to 100 feet (50 feet for each of the two "extension" cables on each side of the interface point). Hackers use longer cables at your own risk.
- "0" = "ON" = "SPACE" = +3 to +15 volts.
- "1" = "OFF" = "MARK" = −3 to −15 volts.

Mechanical Characteristics

The standard's mechanical specifications are vague and brief. Here are my interpretations. Everything is given in terms of a DTE connected to a DCE; again, in practice, we cannot connect two computers or a computer and a printer without having one of them simulate a MODEM.

1. The interface is not "hard-wired.". There must be a "pluggable connector signal interface point between the two equipments."
2. The DCE (MODEM) has a female connector, which must be "mounted in a fixed position near the DTE." This connector can be mounted on the end of a cable from the DCE; it doesn't have to be attached directly to the DCE.
3. The standard seems to require that the DTE provide a panel-mounted connector (any sex) and an "extension cable" ending in a male connector.
4. The "extension" cables on either DTE or DCE (or both) are recommended to be shorter than 50 feet, although longer cables are permitted provided the total load capacitance including terminator "does not exceed 2500 picofarads."
5. If there is a separate unit between the DCE and DTE to provide additional functions, there must be a female connector on the side connecting with the DTE and a male connector on the side connecting to the DCE.
6. The standard defines 25 pins, three of which are unassigned and two of which are reserved for testing. That leaves 20 pins actually defined for interchange circuits.

Hackers View of the Mechanical Requirements.

If you want to build devices using the RS-232C interface, the mechanical requirements dictate at least the following:

- The DTE must provide a female connector on the end of an extension cable.
- The DCE must provide a female connector mounted in a fixed position.
- In practice, these two requirements are met by putting female connectors on the

Pin No.	Circuit Name		Description	Type	Direction
	RS-232-C	Mnemonic			
1	AA	GND	Protective Ground	Ground	X
7	AB	GND	Signal Ground		X
2	BA	TXD	Transmitted Data	Data	From DTE
3	BB	RXD	Received Data		To DTE
14	SBA	(S)TD	Secondary Transmitted Data		From DTE
16	SBB	(S)RD	Secondary Received Data		To DTE
4	CA	RTS	Request to Send	Control	From DTE
5	CB	CTS	Clear to Send		To DTE
6	CC	DSR	Data Set Ready		ToDTE
8	CF	DCD	Data Carrier Detected		To DTE
20	CD	DTR	Data Terminal Ready		From DTE
21	CG	SQ	Signal Quality Dector		To DTE
22	CE	RI	Ring Indicator		To DTE
23	CH		Data Signal Rate Selector (DTE)		From DTE
23	CI		Data Signal Rate Selector (DCE)		To DTE
12	SCF	(S)DCD	Secondary Data Carrier Det.		To DTE
13	SCB	(S)CTS	Secondary Clear To Send		To DTE
19	SCA	(S)RTS	Secondary Request to Send		From DTE
15	DB	(TC)	Trans. Signal Element Timing (DCE)	Timing	To DTE
17	DD	RC	Rec. Signal Element Timing (DCE)		To DTE
24	DA	(TC)	Trans. Signal Element Timing (DTE)		From DTE
9			Reserved for Data Set Testing		
10			Reserved for Data Set Testing		
11			Unassigned		
18			Unassigned		
25			Unassigned		

Figure 3—RS-232-C Circuit Groups

rear panels of both the DCE and the DTE, using a cable with male connectors on both ends to connect the two "equipments."

- A 25-pin connector is specified, but no details are given about its size and shape. In practice, the DB-25S (female) and DB-25P (male) connectors are the de facto standards.
- Since all but three of the 25 pins are explicitly defined, it is asking for trouble to add functions not included in the standard. But if you do, use the three unassigned pins. Some dumb practices I've run into that cause problems are (1) putting both a serial and a parallel interface on a single DB-25 connector, (2) adding a current loop interface to an RS-232C serial DB-25 connector and (3) using a DB-25 connector for a "Centronics compatible" interface (the newest micros from both Apple and IBM do this).

Definitions of Circuit Functions

The standard defines four groups of circuits, as listed in Figure 3:

Ground
Data
Control
Timing

Ground Circuits

Two ground wires are specified. Pin seven, the "signal ground/common return," is always used because it is the single return path for all the currents in the interchange circuits.

Pin 1: Protective ground.

Pin 7: Frame ground.

Pin one, the "protective ground," is optional and is often omitted, but shouldn't be. This wire is designed to protect operators by ensuring that the chassis or frames of the DCE and DTE are both at the same electrical potential. A shock hazard can exist if this wire is not used. Pin one of the connectors on each piece of gear should be "electrically bonded to the machine or equipment frame."

Carefully designed analog equipment avoids internal "ground loops" by maintaining a "signal common" within the chassis, connected to the chassis at one and only one point. In this case, Pin 1 connects to the frame and Pin 7 connects to the signal common (often called "logic ground" or "logic common" on digital equipment.)

Data

Four data leads are defined; two for transmit and two for receive.

The standard defines a "primary" or "forward" channel and a "secondary" or "backwards" channel, so in a sense there are two complete interfaces. In micro systems we rarely see any use of the secondary or backwards circuits.

Pin 2:Transmitted Data (to the MODEM)

Pin 3: Received Data (from the MODEM)

For the primary channel, the DTE transmits on pin two and receives on pin three. This means that the DCE transmits on pin three and receives on pin two. For the simplest interface, all that is needed is three wires; on pins two, three and seven. Note that despite what is often done, pin one should not be the ground wire selected for a "three-wire serial interface," since it is optional while pin seven (signal ground) is mandatory.

To connect two DTE's with a "standard" male-to-male cable does not work, since they both transmit on pin two—neither would receive any data. This is solved with a "MODEM eliminator," a short cable with a female connector on one end and a male connector on the other end. Pin two on each end is connected to pin three on the other end (see Figure 4).

The transmitted data circuit must meet several other requirements. It must put out a "marking" voltage (−5 to −15 volts) when not transmitting data and between characters or words. Also, it must not transmit unless an "on" voltage (+5 to +15 volts) is present on pins 4, 5, 6 and 20, when these pins are connected as specified.

The secondary channel DTE transmits on pin 14 and receives on pin 16.

Control

The control circuits are supposed to ensure that neither end tries to transmit data unless the other is ready to listen. Since the standard is concerned with a MODEM telephone connection, several of the control circuits signal the DTE whether or not the telephone connection to the remote MODEM has been properly made. These lines serve no purpose when a terminal or printer is connected directly to a computer, so they are very often omitted.

More commonly, at least with micros, the control leads perform some kind of simplified "handshaking." For all but the simplest of applications, we need some way for each device to tell the other to stop sending or to not start sending. The most common case is where a computer is driving a printer which can't physically print as fast as the computer sends data. When the printer's input buffer fills, it must be able to signal the computer to suspend transmission until the printing catches up. When the printer is ready for more data, it must be able to signal the computer to resume transmission. This is a simple one-way handshake.

A two-way handshake is needed if both ends can transmit. Each must be sure the other end is ready to listen. For example, say a terminal is connected to a computer, and the computer plays the MODEM role. The computer needs to check that the terminal is ready to receive before it sends a block of data and the terminal needs to be sure the computer is ready to listen before it sends keyboard data to the computer. There are several methods of implementing a two-way handshake by sending special characters on the transmit data lines; the control lines provide a hardware handshake scheme. For maximum reliability, both should be used.

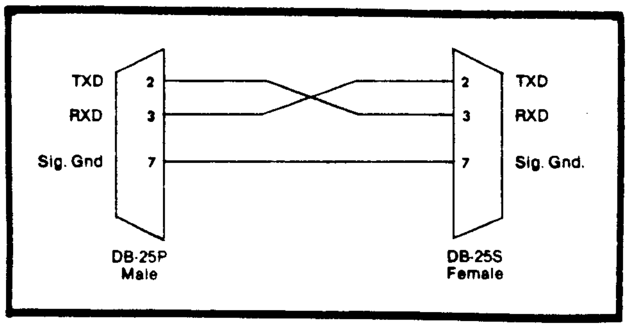


Figure 4— 3-Wire Modem-Eliminator.

There are nine control leads defined for the primary channel:

Pin 4: Request to send (from DTE to the MODEM).

The standard spends a full page enumerating all the things a MODEM must do in response to an ON or OFF condition (or a transition). Simply, the DTE tells the DCE that it is ready to transmit data by setting pin 4 ON (positive voltage).

Pin 5: Clear to send (from the MODEM to the DTE).

The MODEM sets this ON to tell the DTE that Transmitted Data (pin2) will be passed on to the telephone line. In other words, "Go ahead and send, I am listening and am ready to relay your message." This line is set ON only after Data Set Ready and Request to Send have both been set ON.

Pin 6: Data Set Ready
(from the MODEM to DTE).

The MODEM sets this ON to tell the DTE that it (the MODEM only—not necessarily the complete channel to the receiving end) is in a "Ready to listen" condition. This means (1) it is connected to a communication channel, (2) it is not in test, voice, or dial mode, and where these apply, (3) has completed any timing functions related to establishing the call, and (4) has completed transmission of its answering tone. Most of these conditions don't apply in a direct computer-to-printer interface.

Pin 20: Data Terminal Ready
(From DTE to MODEM).

Strictly speaking, this line switches the MODEM to and from the communication line. More loosely, the ON condition of DTR indicates that the DTE is ready to send or receive data.

Pin 8: Received Line Signal Detect (from MODEM to DTE). Commonly called DCD for Date Carrier Detect.

The MODEM sets this line ON when it is receiving a signal on the telephone line which meets its "suitability criteria"; that is, the signal is of high enough quality that the MODEM (which stands for MOdulater/DEModulator) can actually demodulate it. An OFF (negative) level indicates that the MODEM is not receiving a signal or that the received signal is of too low quality to ensure good reception.

Pin 21: Signal Quality Detector
(from MODEM to DTE)

The MODEM sets this line ON whenever "there is no reason to believe an error has occurred." An OFF says an error probably has occurred, due for example to a high noise level on the telephone line.

Pin 23: Data Signal Rate Selector (from DTE or from DCE).

Selects one of two data signalling rates or ranges of rates; an ON level selects the higher rate.

The secondary channel is allotted three handshake (control) lines:

Pin 19: Secondary Request to Send.

Pin 13: Secondary Clear to Send.

Pin 12: Secondary Received Line Signal Detector.

The uses of these lines are analagous to the similar primary channel lines.

Timing Circuits

Pin 24: Transmitter Signal Element Timing (from DTE to MODEM).

This is a clock signal. The DTE effects an ON to OFF transition on this line to indicate the center of each signal element (usually means each bit cell) on the Transmitted Data line. This can be implemented to reduce errors, but is not often used.

Pin 15: Transmitter Signal Element Timing (from MODEM to DTE).

Clock signal from the MODEM to the DTE. If the MODEM implements this signal, the DTE is required to adjust its sending timing such that the transitions (not the center) of the signal elements occur nominally at the same time as the OFF to ON transition of this clock. In other words, where these two timing signals are used, the MODEM controls the DTE's transmitted data transitions, and the DTE tells the MODEM where the element centers are.

Pin 17: Receiver Signal Element Timing (from MODEM to DTE).

A timing element similar to pin 24, but from the MODEM to tell the DTE where to expect the center of the signal elements transmitted by the MODEM to the DTE.

There are no secondary channel timing signals. Where a secondary channel is implemented, timing is controlled by the primary channel.

Reserved Circuits

Pins 9 and 10 are reserved for "Data Set Testing."

Unassigned Circuits

Pins 11, 18 and 25 are "unassigned"; these are for circuits not specified in the standard, "made by mutual agreement."

An example is in the Semiconductor Equipment Communication Standard (SECS) published by the Semiconductor Equipment and Materials Institute, Inc. (SEMI), which uses pins 18 and 25 to supply plus and minus 15 volts power to an isolated interface. While we're on the subject, the SECS overcomes one of the problems with the RS-232C standard interface—isolating the two equipments. You should be conscious of the fact that when you connect two equipments with an RS-232C cable you are connecting their signal and/or frame grounds. This has caused sparks to fly on more than one installation where poor power wiring or large separations resulted in a difference of chassis potential between the two ends.

Table 1: What the RS-232C standard does not specify.

1) The ASCII character code set. ANSI X3.4 (American National Standard Code for Information Interchange) specifies this.

2.) How to control message transfer using ASCII. ANSI X3.28 and X3.57 cover this.

3.) Communication at data rates higher than 20,000 bits per second.

4.) The DB-25 connector. The standard specifies only that the connectors used shall have 25 pins. No physical dimensions (size or shape) are given.

5.) Limitation to asynchronous communications. The standard references RS-334, "Signal Quality at Interface Between Data Processing Terminal Equipment and Synchronous Data Communication Equipment for Serial Data Transmission" for use with synchronous systems.

6.) The "current loop" serial interface. There should be no such thing as a "RS-232 Current Loop," but at least one manufacturer call its interface this. The current loop is an unofficial standard design dating from early teletype days.

Table 2: Where to get the standards.

1.) Electronics Industries Association standards such as the RS-232-C ($5.10 plus $2.50 for RS-232-C application notes):
IEC Standards
2001 "I" Street, NW
Washington, DC 20006

2.) American National Standards Institute standards such as the ASCII standard (American Standard Code for Information Interchange, X3.4-1977):
ANSI Standards
1430 Broadway
New York, NY 10018

3.) IEEE Standards such as the IEEE-488:
IEEE Standards
345 E. 47th Street
New York, NY 10017

4.) Consultative Committee for International Telephone and Telegraph Standards such as the X.25 (international standards):
CCITT Standards
International Telecom
Places de Nations
1211 Geneva 20
Switzerland

5.) International Standards Organization (ISO):
Order ISO standards from the ANSI Standards address above.

6.) International Electro-Technical Commission(IEC):
IEC Standards
1 Rue de Varembe
1211 Geneva 20
Switzerland

So far we've taken a pretty close look at what the RS-232C standard defines as the electrical and mechanical parts of a serial data interchange scheme. Next month we'll cover the standard's "standard configurations," some real-(micro)-world configurations, and we'll present some recommendations for "hacker" standards. ■

To Be Continued

FEEDBACK NEEDED

This magazine is published for you, and in order to serve you we need your feedback. Send us your questions on the RS-232 interface. We'll answer those we can, and publish the others to get help from our readers. Also tell us about your experiences (both the good ones and the bad ones), the problems you've had, and how you've solved them.

TELECOMPUTING WITH THE APPLE][: TRANSFERRING BINARY FILES

BY Marvin L. De Jong

INTRODUCTION

The programs we will describe allow you to communicate a binary disk file (B file) from one Apple II to another over telephone lines. The equipment required is either a Hayes Micromodem II or a California Computer Systems 7710 Asynchronus Serial Interface connected to an acoustic-coupled modem such as a Novation Cat. We will also show how text files (T files) can be converted to B files, so that the programs will also communicate T files. Since it is easy to convert Applesoft files (A files) to T files, you can also communicate Applesoft programs. All of our work was done with a 48K Apple II. The programs must be modified slightly for systems whose memory capacity is different.

HARDWARE

Both the Micromodem II and the CCS 7710 boards use a 6850 ACIA (Asynchronous Communications Interface Adapter). Consequently, a program written for one board will usually work for the other, with little or no modification. The 6850 ACIA performs the parallel-to -serial and the serial-to-parallel conversions that are required for telephone communications. The asynchronous serial protocol is controlled by the software. The protocol assumed by our software is an eight bit word, no parity, one stop bit, and a bit rate of 300 bits/sec. The 6850 is described in detail in De Jong's *Apple II Applications* (Howard W. Sams & Co., Inc., Indianapolis, IN 46268). If you want to modify the programs for other protocols, consult that reference and the operating manuals that come with the peripheral you are using.

If you are using a Micromodem II then the only connection you need to make is to insert the plug from the Micromodem II into the telephone jack. If you are using a CCS 7710 board, then you will need a modem. For our tests we used a acoustic-coupled Novation Cat. The connections between the CCS 7710 board and the Cat are described in Table 1. Both devices use female DB 25 connectors, so you will need two DB 25 male connectors and a five-conductor cable. A row in Table 1 tells you the pins to be connected.

Table 1

CCS 7710 Board Pin Number	Pin Name	Novation Cat Pin Number
3	TXD	2
2	RXD	3
4	CTS	5
20	DSR	6
7	GND	7

COMMUNICATIONS PROGRAMS

The programs to communicate a binary file from one Apple II to another are given in Listings 1 and 2. Both of the communicating parties have these programs loaded and running when they communicate. Listing 1 contains the BASIC routines that make use of the machine-language routines in Listing 2. In this section we will describe how these programs work.

Refer first to Listing 1. First of all, HIMEM is moved down to provide undisturbed space for the machine language routines. An initilization sequence, starting at line 450 is performed next. These lines are quite easy to understand, and the initilization sequence concludes with the machine language routines being loaded from the disk. Lines 500 to 555 are specifically used by the Micromodem II, not the CCS 7710 board. Their execution stores the correct modem control word in the modem control register, starting the modem transmitter and lifting the telephone off the hook.

Line 600 in Listing 1 calls a machine language routine that resets the 6850 ACIA, sets up the protocol defined above, and waits for carrier to be detected. Line 610 calls a machine language terminal routine. In this routine the two parties can use their keyboards and video monitors to communicate with each other. The person who wishes to send the binary file exits the terminal routine by typing a "CTRL S" character. At this point both programs exit the terminal routine and go back to the BASIC program at line 620.

If a "CTRL S" character was *sent*, then memory location 10 will contain a zero and the program will branch to lines 100 to 200. If a "CTRL S" character was *received*, then memory location 10 will not be zero, and the program will branch to lines 300 to 410. Now each party is executing a different routine. The computer sending the binary file will be executing lines 100 to 200, while the computer receiving the binary file will be executing lines 300 to 410.

Lines 100 to 200 accomplish the following tasks. The file name is input from the keyboard, the file is loaded into memory, and its address and length are obtained from the memory locations where the Apple II DOS stored these parameters. These parameters are now stored in new locations in page zero of memory. The program then calls a machine language program that sends the binary file.

Here is how the file is sent. First the starting address of the binary file is sent in two bytes. Refer to line 147 in the machine language routine, Listing 2. Next, the length of the file is sent in two bytes. The length of the file is added to the starting address to get the ending address. This is used by the sending routine to know when to quit. Each byte is now read from memory and sent to the serial output subroutine. Refer to lines 158 to 183 in Listing 2.

After each byte is sent it is added to a checksum. The checksum adds all the bytes in one page of memory as they are sent. After sending one page of memory, the receiving routine sends back its two checksum bytes, and the transmitting program compares them with its own checksum. If there is disagreement, a flag byte is set, and the parties are notified of a checksum error. If less than one page of binary data is sent, then the checksum is completed when the last byte is sent.

The checksums are accumulated in memory locations that correspond to the text screen. While the data is being sent you can watch the two screen locations change. Obviously, if the two screen locations do not change after a few minutes then something tragic has happened, someone has hung up or lighting has struck the telephone system, and you should return to the terminal mode, or lift up the receiver and communicate normally. On the other hand, if you are sending a HIRES graphics page, perhaps a ground hog day greeting card you have drawn, then there might be numerous bytes that contain zero, so don't be too quick on the draw if the checksum locations do not appear to be changing rapidly. When both parties observe a checksum error you will automatically be returned to the terminal mode.

Let us assume the file is successfully transmitted. Then the BASIC program continues at line 180. It waits until the receiving party has saved the binary file on disk, and then control returns to the terminal mode.

On the receiving end, lines 300 to 410 are executed after a "CTRL S" is received. The machine language receiving routine starts on one 94 in Listing 2. The starting address and the length of the file are received and stored, then the data is stored in the same memory locations as it is found in the computer of the transmitting party. When all the checksum information has been returned, the complete file is in memory and control returns to the BASIC program at line 330. Now the user is obligated to input a file name and the Apple II DOS saves the file. The word "READY" is sent to the other computer, and then both parties are back in the terminal mode.

The extensive remarks and comments should make the programs understandable. We have not included any scheme to interrupt the programs if the carrier is lost. On the Micromodem II this requires a solder connection, and of course it requires an interrupt routine to handle the various situations that can produce an interrupt on the 6850 ACIA. If in fact the carrier is lost, both programs will get hung. This will be obvious in the terminal mode if you no longer get any information. If it happens, lift off the telephone handset and talk. When the file is being communicated, a loss of carrier, or another problem perhaps, is detected by "dead" checksum characters on the screen of the video monitor. Wait a bit to make sure the system really is down and not just transmitting zeros, then lift up the telephone and check for the carrier. Serial communications usually are very reliable, so you should not have to deal with these situations very often.

One last detail. How do you get started? If you are going to transmit a file, make the telephone call and exchange greetings. Then both parties RUN their programs. Enter the correct slot number. If you are calling, select the originate mode: if you are answering, select the answer mode. When the carrier is detected you can begin communicating in the terminal mode. With an acoustic coupled modem this means placing the handset in the muffs. With the Micromodem II this means hanging up the telephone and letting the Micromodem take over since it also has the handset off the hook.

ADDITIONAL PROGRAMS

We include two other utility programs to make the previous programs more useful. The first utility program takes a text file and converts it to a binary file, in which form it can be used by the programs in Listings 1 and 2. This program is given in Listing 3. Assume that you have a text file on your disk. The program asks you to input the file name. Then it GETs the elements in the file, converts them to ASCII, and stores them in memory. Once in memory they are BSAVEd using the file name prefixed with a "B". The file can now be sent using the communications programs in Listings 1 and 2.

The second utility program takes a binary file made by the program in Listing 3 and converts it back to a text file. It also prints the text on the screen. In other words, once a text file is communicated as a binary file, you will need a technique to change it back to its original form. The program in Listing 4 does just that. It BLOADs the file, then takes each ASCII code and saves it in a text file whose name is the same as the binary file but prefixed by a "T".

Suppose you have a text file named LETTER that you wish to send by telephone to another party. The easiest way is to purchase a 20 cent stamp and mail it. A faster more elegant and expensive way to send the letter is to convert it to a binary file using the program in Listing 4. It now bears the name BLETTER. You send it over the telephone using the programs in Listings 1 and 2. Suppose the receiving party gives it the same name, BLETTER, and it stored on that disk with that file name. The receiving party uses the program in Listing 4 to convert it to a text file bearing the name TBLETTER. Now a wordprocessor or another utility can be used to convert the text file into print.

The Apple II DOS manual illustrates how an Applesoft program can be converted to a text file. The program in Listing 3 converts it to a binary file, and finally, after it has been transmitted over the telephone system, the program in Listing 4 is used to convert it back to a text file. Then the EXEC command will load it into your machine in a form that you can RUN it.

I hope you enjoy working with and improving these programs. Bells and whistles include driving all of the programs from a menu, using program input to set the protocol, and, in the case of a direct-connect modem, adding a dialing routine. Several other communications programs are given in the reference mentioned early in the article. Why buy an expensive communications package when you can hack one out yourself?

LISTING 1

```
 15  REM MICROMODEM II OR CCS 7710 BINARY FILE SEND/RECEIVE

 20  HIMEM:37887 : GOTO 450

100  PRINT "INPUT THE FILE NAME."
110  INPUT FILE$
120  PRINT D$; "BLOAD";FILE$
125  REM NEXT POKES AND PEEKS ASSUME 48K APPLE II.
130  POKE 249,PEEK (43634) : POKE 250, PEEK (43635)
135  REM MOVE ADDRESS OF BINARY FILE TO PAGE ZERO LOCATIONS.
140  POKE 251,PEEK (43616) : POKE 252, PEEK (43617)
145  REM MOVE LENGTH OF BINARY FILE TO PAGE ZERO LOCATIONS.
150  CALL 38130 : REM CALL ROUTINE TO TRANSMIT B-FILE.
160  IF PEEK (10) = 0 THEN 180
170  PRINT "CHECKSUM ERROR.  BACK TO TERMINAL MODE." : GOTO 610
180  PRINT "FILE TRANSMITTED SUCCESSFULLY."
190  PRINT "PLEASE WAIT A MOMENT FOR HOUSEKEEPING CHORES"
200  PRINT "TO BE COMPLETED AT THE OTHER TERMINAL." : GO TO 610

300  POKE 10,0 : CALL 38020
310  IF PEEK (10) = 0 THEN 330
320  PRINT "CHECKSUM ERROR.  BACK TO TERMINAL MODE." : GOTO 610
330  PRINT "FILE IS NOW IN MEMORY."
340  PRINT "INPUT A NAME FOF THE FILE."
350  INPUT FILE$
360  LGTH = PEEK(251) + 256*PEEK(252)
370  ADDR = PEEK(249) + 256*PEEK(250) - LGTH
380  PRINT D$;"BSAVE";FILE$;",A";ADDR;",L";LGTH
390  FOR I = 1 TO 5: READ A$ :POKE 69, ASC(A$) : CALL 37985
400  NEXT I : DATA R,E,A,D,Y : RESTORE
410  PRINT "YOU ARE BACK IN THE TERMINAL MODE." : GOTO 610

445  REM  INITILIZATION SEQUENCE
450  CLEAR : D$ = CHR$(4) : HOME
470  PRINT "INPUT THE SLOT NUMBER OF THE SERIAL INTERFACE."
480  INPUT S : POKE 255,16*S
490  PRINT D$;"BLOAD BTERMINAL ROUTINES 1.0.OBJ0"

500  MODEM =  - 16251 + 16 * S
510  PRINT "ANSWER (A) OR ORIGINATE (O)?"
520  INPUT A$
530  IF A$="A" THEN POKE MODEM,139 : GO TO 600
540  IF A$="O" THEN POKE MODEM,143 : GO TO 600
550  GO TO 510
555  REM OMIT LINES 500 - 555 FOR CCS 7710 SERIAL I/O BOARD.

600  CALL 37888 : REM  CALL INITIALIZATION ROUTINE.
610  CALL 37911 : REM CALL TERMINAL ROUTINE.
620  HOME : IF PEEK (10) = 0 THEN 100
630  GO TO 300
```

LISTING 2

```
SOURCE FILE: BTERMINAL ROUTINES 1.0
0000:           ;THIS PROGRAM ASSUMES THE HAYES MICROMODEM OR THE
0000:           ;CCS 7710 CARD IS IN A CARD SLOT.
C000:           KYBD     EQU  $C000        ;APPLE KEYBOARD ADDRESS
C010:           STRB     EQU  $C010        ;KEYBOARD STROBE CLEAR.
C086:           CR       EQU  $C086        ;6850 ACIA CONTROL REGISTER.
C086:           STATUS   EQU  $C086        ;6850 ACIA STATUS REGISTER.
C087:           DATA     EQU  $C087        ;6850 ACIA DATA REGISTER.
FDF0:           COUT1    EQU  $FDF0        ;MONITOR OUTPUT ROUTINE.
000A:           FLAG     EQU  $0A          ;FLAG LOCATION.
----- NEXT OBJECT FILE NAME IS BTERMINAL ROUTINES 1.0.OBJ0
9400:                    ORG  $9400

9400:           ;INITIALIZATION ROUTINE.
9400:A4 FF               LDY  $FF          ;GET PERIPHERAL SLOT INDEX.
9402:A9 03               LDA  #$03         ;RESET 6850 ACIA.
9404:99 86 C0            STA  CR,Y
9407:A9 15               LDA  #$15         ;SET PROTOCOL: 8 BITS,
9409:99 86 C0            STA  CR,Y         ;NO PARITY, ONE STOP BIT.
940C:B9 87 C0   WAIT     LDA  DATA,Y       ;READ AND DISCARD DATA.
940F:B9 86 C0            LDA  STATUS,Y     ;ARE CTS AND DCD SIGNALS
9412:29 0C               AND  #$0C         ;PRESENT?
9414:D0 F6               BNE  WAIT         ;NO, WAIT HERE.
9416:60                  RTS               ;RETURN FROM INITIALIZATION.

9417:           ;TERMINAL ROUTINE.
9417:A9 00      TERM     LDA  #00          ;CLEAR FLAG REGISTER.
9419:85 0A               STA  FLAG
941B:A4 FF               LDY  $FF          ;GET SLOT INDEX.
941D:B9 86 C0   LOOP     LDA  STATUS,Y     ;YES, IS RECEIVER READY?
9420:29 01               AND  #01
9422:F0 0C               BEQ  TRANS        ;NO, TRY THE TRANSMITTER.
9424:B9 87 C0            LDA  DATA,Y       ;YES, GET THE DATA.
9427:09 80               ORA  #$80         ;SET HIGH BIT.
9429:C9 93               CMP  #$93         ;IS IT "CTRL S"?
942B:F0 20               BEQ  SEND         ;BRANCH TO RECEIVE ROUTINE.
942D:20 F0 FD            JSR  COUT1        ;OUTPUT THE CHARACTER.
9430:AD 00 C0   TRANS    LDA  KYBD         ;READ THE KEYBOARD.
9433:10 E8               BPL  LOOP         ;NO DATA, SO LOOP BACK.
9435:48                  PHA               ;SAVE THE CHARACTER.
9436:8D 10 C0            STA  STRB         ;CLEAR THE KEYBOARD STROBE.
9439:20 F0 FD            JSR  COUT1        ;REPLACE WITH THREE "NOP"
943C:           ;INSTRUCTIONS IF OTHER TERMINAL ECHOES.
943C:B9 86 C0   PAUSE    LDA  STATUS,Y     ;IS TRANSMITTER READY?
943F:29 02               AND  #02          ;PAUSE UNTIL IT IS.
9441:F0 F9               BEQ  PAUSE
9443:68                  PLA               ;GET CHARACTER FROM STACK.
9444:99 87 C0            STA  DATA,Y       ;OUTPUT IT TO THE ACIA.
9447:C9 93               CMP  #$93         ;WAS "CTRL S" SENT?
9449:F0 04               BEQ  PAST
944B:D0 D0               BNE  LOOP         ;GO BACK FOR MORE.
944D:C6 0A      SEND     DEC  FLAG         ;SET FLAG.
944F:60         PAST     RTS

9450:           ;SERIAL INPUT ROUTINE
9450:84 47      SERIN    STY  $47          ;SAVE Y HERE.
9452:A4 FF               LDY  $FF          ;GET SLOT INDEX.
9454:A9 01      HOLD     LDA  #$01         ;MAKE A MASK.
9456:39 86 C0            AND  CR,Y         ;RECEIVER FULL?
9459:F0 F9               BEQ  HOLD         ;NO, WAIT FOR IT TO FILL.
945B:B9 87 C0            LDA  DATA,Y       ;READ THE ACIA.
945E:A4 47               LDY  $47          ;RESTORE Y.
9460:60                  RTS
9461:           ;SERIAL OUTPUT ROUTINE
9461:A5 45               LDA  $45          ;USED TO SEND FROM BASIC.
9463:84 47      SEROUT   STY  $47          ;SAVE Y.
9465:A4 FF               LDY  $FF          ;GET SLOT INDEX.
9467:48                  PHA               ;SAVE CHARACTER ON THE STACK.
9468:A9 02      LOAF     LDA  #02          ;MAKE A MASK.
946A:39 86 C0            AND  CR,Y         ;TRANSMITTER EMPTY?
946D:F0 F9               BEQ  LOAF         ;NO, WAIT UNTIL IT IS.
946F:68                  PLA               ;YES, THEN GET CHARACTER
9470:99 87 C0            STA  DATA,Y       ;AND SEND IT OUT.
9473:A4 47               LDY  $47          ;FETCH Y BACK.
9475:60                  RTS

00F9:           ADDRLO   EQU  $F9          ;HOLDS ADDRESS OF BINARY FILE.
00FA:           ADDRHI   EQU  $FA
00FB:           LNGTLO   EQU  $FB          ;HOLDS LENGTH OF BINARY FILE.
00FC:           LNGTHI   EQU  $FC
00FD:           ENDLO    EQU  $FD          ;LAST ADDRESS IN BINARY FILE.
00FE:           ENDHI    EQU  $FE
0536:           SUMLO    EQU  $536         ;CHECKSUM STORAGE.
0535:           SUMHI    EQU  $535

9476:           ;ADDITION ROUTINE.
9476:18         ADD      CLC
9477:A5 F9               LDA  ADDRLO       ;ADD LENGTH TO ADDRESS
9479:65 FB               ADC  LNGTLO       ;TO FIND ENDING ADDRESS.
947B:85 FD               STA  ENDLO
947D:A5 FA               LDA  ADDRHI
947F:65 FC               ADC  LNGTHI
9481:85 FE               STA  ENDHI
9483:60                  RTS

9484:           ;RECEIVE FILE ROUTINE.
9484:20 50 94   FETCH    JSR  SERIN        ;GET LOW BYTE OF ADDRESS
9487:85 F9               STA  ADDRLO       ;STORE IT.
9489:20 50 94            JSR  SERIN        ;GET HIGH BYTE OF ADDRESS.
948C:85 FA               STA  ADDRHI       ;STORE IT.
948E:20 50 94            JSR  SERIN        ;GET LOW BYTE OF LENGTH.
9491:85 FB               STA  LNGTLO       ;STORE IT.
9493:20 50 94            JSR  SERIN        ;GET HIGH BYTE OF LENGTH.
9496:85 FC               STA  LNGTHI       ;STORE IT.
9498:20 76 94            JSR  ADD          ;ADD TO FIND ENDING ADDRESS.
949B:A0 00               LDY  #00          ;PREPARE TO GET BINARY
949D:A2 00               LDX  #00          ;FILE AND STORE IT.
949F:8E 36 05   BACK     STX  SUMLO        ;CLEAR CHECKSUM LOCATIONS.
94A2:8E 35 05            STX  SUMHI
94A5:20 50 94   MORE     JSR  SERIN        ;GET A FILE BYTE.
94A8:91 F9               STA  (ADDRLO),Y ;STORE IT.
94AA:18                  CLC               ;PUT THE BYTE IN
94AB:6D 36 05            ADC  SUMLO        ;THE CHECKSUM.
94AE:8D 36 05            STA  SUMLO
94B1:A9 00               LDA  #00
94B3:6D 35 05            ADC  SUMHI
94B6:8D 35 05            STA  SUMHI
94B9:E6 F9               INC  ADDRLO       ;UPDATE THE LOCATION.
94BB:D0 02               BNE  ARND
94BD:E6 FA               INC  ADDRHI
94BF:A5 F9      ARND     LDA  ADDRLO       ;CHECK TO SEE IF THE
94C1:C5 FD               CMP  ENDLO        ;FILE IS COMPLETE.
94C3:D0 0A               BNE  HERE
94C5:A5 FA               LDA  ADDRHI
94C7:C5 FE               CMP  ENDHI
94C9:D0 04               BNE  HERE
94CB:20 D8 94            JSR  CHKSUM
94CE:60         OUT      RTS
94CF:E8         HERE     INX               ;OUTPUT ONE CHECKSUM FOR
94D0:D0 D3               BNE  MORE         ;EVERY PAGE OF DATA.
94D2:20 D8 94            JSR  CHKSUM
94D5:18                  CLC               ;FORCED BRANCH.
94D6:90 C7               BCC  BACK

94D8:           ;CHECKSUM ROUTINE.
94D8:AD 36 05   CHKSUM   LDA  SUMLO        ;SEND LOW BYTE OF CHECKSUM.
94DB:20 63 94            JSR  SEROUT
94DE:AD 35 05            LDA  SUMHI        ;SEND HIGH BYTE OF CHECKSUM.
94E1:20 63 94            JSR  SEROUT
94E4:20 50 94            JSR  SERIN        ;WAIT FOR REPLY.
94E7:C9 06               CMP  #06          ;IS IT "ACK" CODE?
94E9:D0 01               BNE  ERROR        ;NO, CHECKSUMS DID NOT MATCH.
94EB:60                  RTS
94EC:C6 0A      ERROR    DEC  FLAG         ;SET THE ERROR FLAG.
94EE:D0 DE               BNE  OUT
94F0:F0 DC               BEQ  OUT

94F2:           ;TRANSMIT FILE ROUTINE.
94F2:A5 F9               LDA  ADDRLO       ;GET LOW BYTE OF FILE ADDRESS.
94F4:20 63 94            JSR  SEROUT       ;SEND IT.
94F7:A5 FA               LDA  ADDRHI       ;GET HIGH BYTE.
94F9:20 63 94            JSR  SEROUT       ;SEND IT.
94FC:A5 FB               LDA  LNGTLO       ;GET LOW BYTE OF FILE LENGTH.
94FE:20 63 94            JSR  SEROUT       ;SEND IT.
9501:A5 FC               LDA  LNGTHI       ;GET HIGH BYTE.
9503:20 63 94            JSR  SEROUT       ;SEND IT.
9506:20 76 94            JSR  ADD          ;FIND ENDING ADDRESS.
9509:A0 00               LDY  #00
950B:A2 00               LDX  #00
950D:8E 36 05   BACK1    STX  SUMLO        ;CLEAR CHECKSUM.
9510:8E 35 05            STX  SUMHI
9513:B1 F9      MORE1    LDA  (ADDRLO),Y ;FETCH A BYTE.
9515:20 63 94            JSR  SEROUT       ;SEND IT.
9518:18                  CLC               ;CALCULATE CHECKSUM.
9519:6D 36 05            ADC  SUMLO
951C:8D 36 05            STA  SUMLO
951F:A9 00               LDA  #00
9521:6D 35 05            ADC  SUMHI
9524:8D 35 05            STA  SUMHI
9527:E6 F9               INC  ADDRLO       ;UPDATE ADDRESS.
9529:D0 02               BNE  BRNCH
```

continued on page 13

BEGINNER'S COLUMN

ANYONE FOR A LITTLE KISS ELECTRONICS?

by Phil Wells—Technical Editor

I remember when Byte, Interface Age, Kilobaud and the other "computer hobby" magazines started up, six or seven years ago; they were filled with "how-to" hardware construction articles where engineers, techs and just plain hackers shared their knowledge and experiences with us. What happened to those articles, those magazines? The micro hobby turned into the small-business and home computer craze. A world-wide gold mine was tapped as publishers realized millions of people wanted to know what a micro was, how to make a purchasing decision, what to use a micro for. The hardware hobbyist got lost in the rush. The money was to be made selling new micro add-ons, writing hardware and software reviews; the hardware hobbyist became a very poor relation.

Every week I run into someone else who has been bitten by the micro bug, learned how to run a micro, and now wants to really get into the hardware, but is having trouble getting started. "How do I go about learning just enough electronics to get started modifying and building my own stuff?" they ask. Not to become an engineer, just to be able to play with add-ons, experiment with robotics, have some fun really doing something.

Electronics for the professional is a large and complex subject. Does electronics for the hobbyist really have to be that difficult? I don't think so. Digital electronics (the kind our micros use) is much simpler than analog electronics. A hacker doesn't have to "design for production"—that is, calculate what might happen to every aspect of circuit performance as all possible combinations of component tolerances interact in a hundred thousand production models. All the hacker has to do is get one sample to work acceptably in a particular application.

How about electronics through the KISS (Keep it Simple, Stupid) principle? Certainly, the deeper your understanding of a technical subject, the better your chances of success in designing your own hardware. But I've noticed a little understanding can get a hobbyist a long way, and more importantly, that the hardest part is just getting started. So this column is for the beginning hardware hacker who wants to start burning his fingers with a soldering iron without plowing through textbooks or going to night school.

We'll always try to keep it simple, and we'll cover whatever topics you write in and tell us you want covered. Everything will presented hands-on fashion; not necessarily whole projects, but at least simple circuits you can put together to see the concepts in action. **If you don't get in and mess around with it, you're not going to learn it.** To help locate tools, parts, etc. I'm going to give Radio Shack part numbers, because the Shack is simply the most commonly available get-it-now source.

Getting Started: Tools

Unless you like cutting and forming wire with your teeth, you'll need some small hand tools. I like the smaller versions (4-5 inch handles), since most micro-type projects involve very small parts.

1) 4½" diagonal sidecutters. Also the "flush-cut" style is handy for printed circuit work.

2) 5" needle-nose pliers.

3) Wire strippers.

4) Set of small screwdrivers and a set of small nut drivers.

5) Small pencile-style soldering iron (45 watts is probably too hot—get the 37½ or 25 watt size—with a small chisel tip) and rosin core solder **(never use acid core solder)**.

6) Invest $12 to $19 in a "breadboard socket" such as Radio Shack #276-174, I have a collection of these, each able to hold about six 16 pin I.C.'s, picked up at surplus stores at various times over the years. This is absolutely the best way I've ever found to try out a circuit idea, making many changes quickly without soldering, before committing the design to a more permanent construction method.

7) Clip leads. This is an 18" piece of the most flexible wire you can find, terminated on both ends with an Alligator clip or a spring action "mini test clip" (R.S. #270-372 or 270-370). The Shack has a set of 10 really el-cheapo but usable light weight leads for $3.69 (#278-1156). You'll need at least four made with 16 gauge or heavier wire with large alligator clips for connecting power supplies. A dozen of the smaller 22 or 24 gauge size with micro-clips is not too many.

8) Hook-up wire. You'll need 22 gauge (or there-abouts) wire in various colors. Get used to using stranded wire for projects designed to last a while—solid wire breaks with flexing. You'll also need some insulated 22 gauge solid wire for use as breadboard socket jumpers.

Getting Started: Parts

You'll need parts for any project you want to build. If you live in an area with "surplus" stores, scrounge around there for your parts. This magazine will regularly publish information on how to mail-order parts. The most widely available source of cheap (in both senses) parts is your local Radio Shack store, so I'll always try to suggest Radio Shack parts that have worked for me. But be forewarned, my experience is that the Shack generally provides low quality at prices that are usually higher than mail order prices. The most reliable (and by far the fastest) source of mail order parts I've found is Jameco Electronics, (415) 592-8097; call and ask for their catalog.

Resistors. You'll need an assortment of ¼ Watt resistors. The Shack has a useful starter box of 350 for $9.95. Surplus stores usually sell junked p.c. boards for a dollar or less; strip the resistors, capacitors and transistors from these. For higher wattages, see if your local TV repair shops will sell or give you junked TV chassies. Strip these for parts.

Capacitors. You'll need an assortment of disc ceramics and another of aluminum electrolytics. The Shack has $9.95 assortments of these.

Transistors and IC's. Haunt the Shack's half-price table, look for specials in magazine ads, but basically buy these as you need them. You might try the Shack's 2N3904 (NPN) and 2N3906 (PNP) sets of 15 for $1.98, but expect to find some that don't work. These are unlabeled, or are actually other near-equivalent types, and look like seconds or rejects; but then, we're not designing for production, remember? The Shack's 2N2222 (15 for $1.98) type transistor works fine for most low power needs.

Diodes. You'll need two kinds; small-signal silicon like 1N914's, and silicon rectifiers like the 1N4002, The Shack has ten 1N914/1N4148's for $0.99, and 1-Amp rectifier diodes (get P/N 276-1102) at two for $0.59. You'll also need Zener diodes and an occassional bridge rectifier, but unless you find a bargain, get these as you need them. If you can pick up an assortment of LED's (ight emitting diodes) at a good price do it; they're useful and fun to play with.

Getting Started: Test Equipment

Electronic test equipment is a whole world in itself; you can spend as much as your budget will allow. However, one item is absolutely necessary and a few more will really enhance your possibilities.

1) You'll need a meter to measure Volts, Ohms, and Amps. It is possible to build your own from surplus parts, but not worth the effort unless this is a major area of interest to you. You can get started with a simple analog "VOM" (Volt-Ohm-Milliammeter) such as the Radio Shack #22-204 at $39.95, but this type meter will not always give accurate voltage readings with transistor and I.C. circuits. A better investment is a "DVM" (Digital Volt Meter) such as the Radio Shack #22-191 at $79.95. A 3½ digit display is fine for our purposes.

2) A logic probe is a very useful tool which is sometimes easier to use than an oscilloscope and an awful lot less expensive. This is a hand held probe with light emitting diodes to indicate a logic "high," "low" or "pulse" signal. Radio Shack has a bare-bones model (#22-301) for $19.95.

3) An oscilliscope is a very worth while investment and is a necessity for the really serious hacker. This tool is the only way to really see what the fast changing voltages in your circuits are doing; it is a window into the world of the electrons. Selecting a 'scope is not a simple task since prices range from a few hundred to over eight thousand dollars. Get the best you can afford, learn to use it, and you'll have acquired an invaluable "sixth sense." For microcomputer work you need an absolute minimum bandwith rating of 35 MHz, and preferably at least 50 MHz. Two vertical channels are very much worth having. Delayed sweep you can do without. A vertical sensitivity of at least 10 mV is needed for analog work, but not often for digital. The ability to invert one channel is necessary if you want to do disk drive alignment. You also need an external trigger capability. Most important is a stable zero-level, accurate and stable vertical sensitivity calibration, and good, stable triggering. I've been disappointed in Heath 'scopes in this respect—the only really good triggering I've seen is in Tektronix and Hewlett-Packard 'scopes. If you have had good experiences with others, let me know. ■

Telecomputing with the Apple][, continued

```
952B:E6 FA                INC  ADDRHI
952D:A5 F9        BRNCH   LDA  ADDRLO     ;HAVE ALL BYTES BEEN SENT?
952F:C5 FD                CMP  ENDLO
9531:D0 0A                BNE  HERE1
9533:A5 FA                LDA  ADDRHI
9535:C5 FE                CMP  ENDHI
9537:D0 04                BNE  HERE1
9539:20 46 95             JSR  SUMCHK     ;YES, SO SEND LAST CHECKSUM.
953C:60           RET     RTS             ;THEN GO BACK TO BASIC.
953D:E8           HERE1   INX             ;HAS AN ENTIRE PAGE BEEN SENT?
953E:D0 D3                BNE  MORE1      ;NO, SO FINISH A PAGE.
9540:20 46 95             JSR  SUMCHK     ;OTHERWISE COMPARE CHECKSUMS.
9543:18                   CLC             ;FORCE A BRANCH BACK FOR
                                           ANOTHER.
9544:90 C7                BCC  BACK1

9546:             ;SUMCHK SUBROUTINE.
9546:20 50 94     SUMCHK  JSR  SERIN
9549:A8                   TAY
954A:20 50 94             JSR  SERIN      ;GET HIGH BYTE OF CHECKSUM.
954D:CD 35 05             CMP  SUMHI      ;IS IT THE SAME HERE.
9550:D0 0D                BNE  BAD        ;NO.
9552:CC 36 05             CPY  SUMLO      ;DO LOW BYTES COMPARE?
9555:D0 08                BNE  BAD        ;NO.
9557:A0 00                LDY  #00        ;CLEAR Y AGAIN.
9559:A9 06                LDA  #06        ;SEND "ACK" CODE.
955B:20 63 94             JSR  SEROUT
955E:60                   RTS
955F:A9 15        BAD     LDA  #$15       ;SEND "NAK" CODE.
9561:20 63 94             JSR  SEROUT
9564:C6 0A                DEC  FLAG       ;SET ERROR FLAG.
9566:D0 D4                BNE  RET        ;BACK TO BASIC.
9568:F0 D2                BEQ  RET        ;GO GACK TO BASIC.

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Listing 3

ROUTINE TO CONVERT AN ASCII B-FILE TO A TEXT FILE.

```
 10  CLEAR :D$ =  CHR$ (4): PRINT "INPUT THE FILE NAME."
 20  INPUT F$: HOME : PRINT D$;"OPEN";F$: PRINT D$;"READ";F$
 30  ONERR  GOTO 70
 40  L = 0:A = 4096:B$ = ""
 50  GET B$: POKE (A + L), ASC (B$):L = L + 1
 60  GOTO 50
 70  PRINT D$;"CLOSE";F$
 80  F$ = "B" + F$: PRINT F$,A,L
 90  PRINT D$;"BSAVE";F$;",A";A;",L";L
100  END
```

Listing 4

ROUTINE TO CONVERT A TEXT FILE TO A BINARY FILE.

```
 10  CLEAR :D$ =  CHR$ (4): PRINT "INPUT THE FILE NAME."
 20  INPUT F$: PRINT D$;"BLOAD";F$
 30  A =  PEEK (43634) + 256 *  PEEK (43635)
 40  L =  PEEK (43616) + 256 *  PEEK (43617)
 50  F$ = "T" + F$: PRINT F$,A,L
 60  PRINT D$;"OPEN";F$: PRINT D$;"DELETE";F$
 70  PRINT D$;"OPEN";F$: PRINT D$;"WRITE";F$
 80  FOR I = 0 TO L - 1
 90  PRINT  CHR$ ( PEEK (A + I));
100  NEXT I
110  PRINT D$;"CLOSE";F$
120  PRINT " "
130  FOR I = 0 TO L - 1
140  PRINT  CHR$ ( PEEK (A + I));
150  NEXT I: END
```

■

BUILD AN "EPRAM"

by Lance Rose—Technical Editor

Introduction

These days it seems that no computer system can get along without at least some of its software in ROM. The smaller "appliance" computers put the entire BASIC in ROM and more powerful disk-based systems usually have at least a little of it to hold a bootstrap loader that loads in the first part of a disk operating system such as CP/M. Numerous microprocessor-controlled dedicated controllers have their operating programs burned into EPROMS or masked ROMS as well.

With all the need for ROMs it stands to reason that the complete hacker should have some way of programming at least the most common of the currently used EPROMS, the 2716. The equipment usually required for this is (1) some sort of hardware for actually programming a blank EPROM, and (2) some device for erasing the EPROM and reprogramming it when you find out the program didn't work quite right the first ten times.

Anyone who had had much experience in developing some ROM based software will probably attest to the fact that although the method is OK, the time cycle for erasing and reprogramming EPROMs leaves something to be desired. With most of the UV lamps in the hands of today's hackers, it takes nearly an hour to erase an EPROM that has a program bug in it so it can be reprogrammed with the (hopefully) correct version. What this means is that you must either keep a number of these devices handy so that you can be programming one while another is being erased, or simply accept a turnaround time of about an hour for making each change to the software. This becomes particularly annoying when programming dedicated controllers as I can personally attest to, not to mention the effect on the lifetime of your UV lamp tube. What would be nice is to have some sort of EPROM that could be both programmed and erased quickly so that the development procedure could be speeded up. The need for multiple devices would be eliminated if this were the case also. Well, there is just such a device readily available today. It is called RAM.

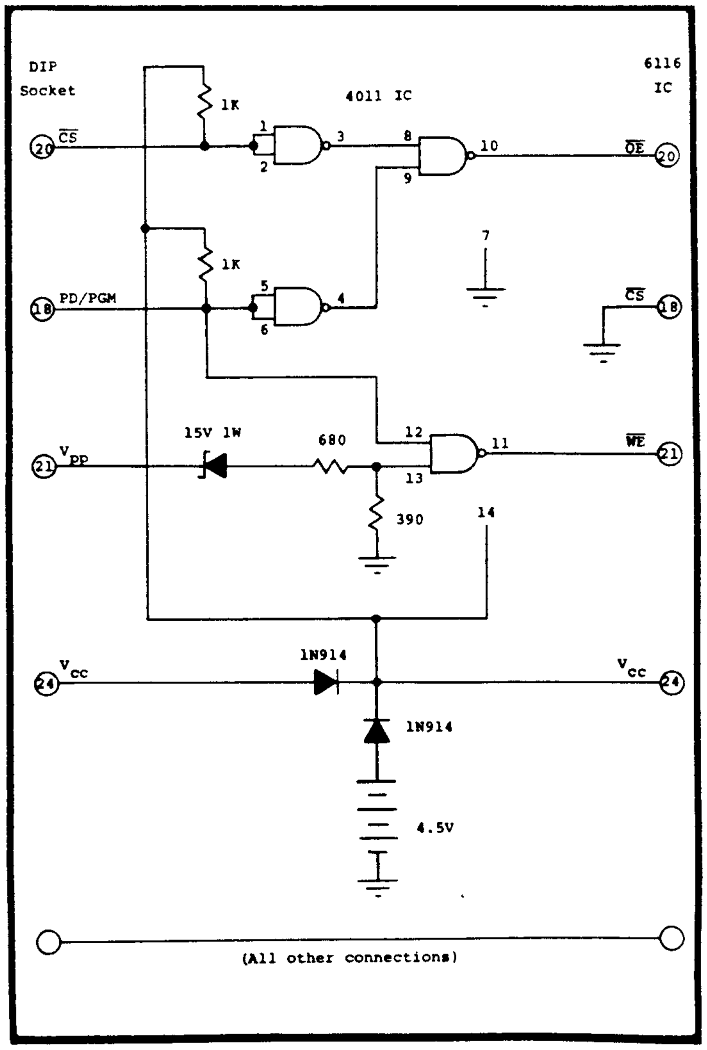
The RAM chip is ready made to be repeatedly and quickly programmed and erased (written to and read from) and would make an ideal substitute for the EPROM chip, at least as far as the development stage of a project goes, if a couple of minor problems could be overcome.

One problem is that in the past the pin compatibility between the RAM and EPROM chips has been almost nonexistent. Now, however there exist pin-compatible substitutes for the commonly used 2716 type EPROMS. These are the 2016 2k x 8 NMOS RAM and the 6116 family of CMOS RAM chips in the same architectural size. These are both 24-pin packages but we will be using the 6116 here due to the low power requirements of CMOS logic.

The second problem is that when the 5-volt supply is turned off, the information stored in a RAM is forgotten. The solution to this is battery backup to preserve the memory. In the case of CMOS, the current requirements are very low and battery life should be quite long.

An additional requirement for any EPROM substitute should be that it ought to be able to be programmed in any standard 2716 EPROM programmer without changes in the programming hardware or software. The device presented here fulfills these requirements and can be built for around $10 to $12 worth of parts and a leisurely day's effort. Similar devices are commercially available but fall in the $50 price range, so this can be quite a bargain.

Figure 1



The Circuit

The EPROM substitute (nicknamed "EPRAM") is shown in schematic diagram form in Figure 1. In order to match the RAM control signals to what most EPROM programmers put out, it is necessary to include some additional logic. This is supplied in the form of an additional CMOS IC, a 4011 quad NAND gate. This inverts some of the signals and prevents the RAM from being written into unless the 25 volt programming voltage is present. The program pulse is converted into a negative going write pulse for the RAM. A 15 volt Zener diode reduces the level of the programming voltage to a manageable level, at which point a resistive divider takes over to present approximately 3.5 volts to the input of the NAND gate. The battery backup supply is isolated from the in-circuit power supply by a pair of small diodes. This prevents the situation of the battery having to attempt to power the whole circuit that the EPRAM is plugged into when the power is turned off but the EPRAM is not unplugged. This way the battery powers only the two CMOS devices and keeps the current drain within reason. The only penalty paid by the isolating diodes is to reduce the power supply voltage to the RAM to 4.3 volts which is still well within the specs of CMOS devices.

After programming, the EPRAM is removed from the programmer and inserted into the dedicated controller, computer logic board or whatever and it then begins to function as any EPROM would. Without the programming voltage it cannot be erased. The same control signals used to select an EPROM are modified slightly to deal with the 6116's very slightly different pin assignments for chip select and output buffer enable.

The battery backup is provided by three ordinary 1.5 volt AA penlight cells in a four cell holder with a wire jumper in place of one of the cells. This is connected by a long, thin flexible wire pair to the actual EPRAM unit.

Construction

The basis for the EPRAM is a 24-pin DIP header. This and the 6116 itself form the actual physical base upon which all the other components are mounted. A good way to start is to attach the 4011 chip and the resistors and diodes to the top of the 6116 using super glue or epoxy (see Figure 2 for the layout). Allow plenty of time for the adhesive to dry as the bending of leads and soldering put some strain on the bond. Overnight doesn't hurt. Then, using a combination of the actual component leads and short pieces of wire-wrap wire, bend the leads and pins and carefully solder everything together. It will be necessary to bend pins 18, 20, 21 and 24 of the 6116 upwards to facilitate making connections to them. Next, place the 6116 unit on top of the DIP header and solder the remaining 20 pins of the 6116 to the corresponding pins of the header. The header can be stuck into some conductive foam while you are doing this. It may be helpful to mention that there are several manufacturers of DIP headers around and their products differ slightly in the dimensions of the protruding pins. I have found that a header with long, thin pins is easier to insert and withdraw from a socket than the one with the shorter, wider pins, so you might want to search for one of the better ones.

At this point I should mention something about CMOS and static. Due to the nature of the construction here, it is difficult to prevent leaving the pins of the 4011 dangling out in space while you are soldering to them. My experience with CMOS here is that if you are reasonably careful (i.e. don't work on a high-static carpet or wear nylon clothing), you can work with unprotected CMOS and not damage it. Take reasonable care but don't get paranoid about it. An alternative would be to solder in a 14-pin DIP socket and insert the chip later but this causes an already-tall package to grow taller yet. Your decision here will show whether you're a conservative or a swinger.

Once the 6116 is soldered to the header you can make the last few connections from the 24-pin header to the 4011. To connect the assembly to the backup battery, I used a twisted pair of wire-wrap wires, one blue and one red. A refinement to this would be to use a small watch or hearing-aid type battery and mount it on the assembly itself. This would eliminate the need for the external battery unit. My goal here was to design this so that it used the most common electronic components possible, components that would be easily available to any hacker within reach of a Radio Shack store.

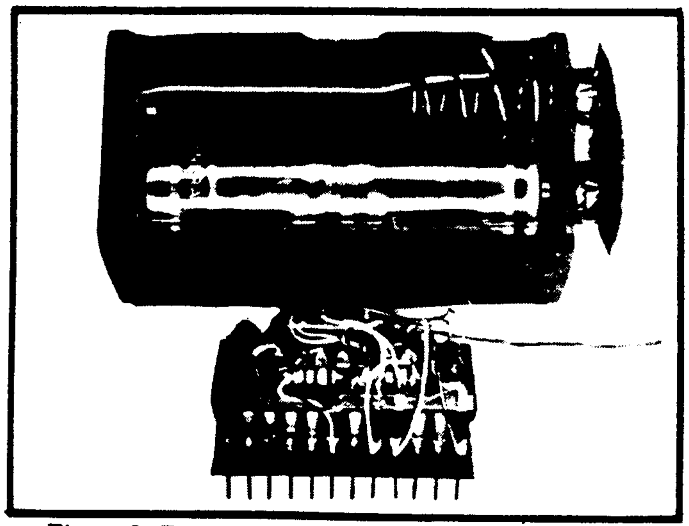


Figure 2a: The assembled EPRAM with battery holder. Note the jumper wire in place of the fourth battery.

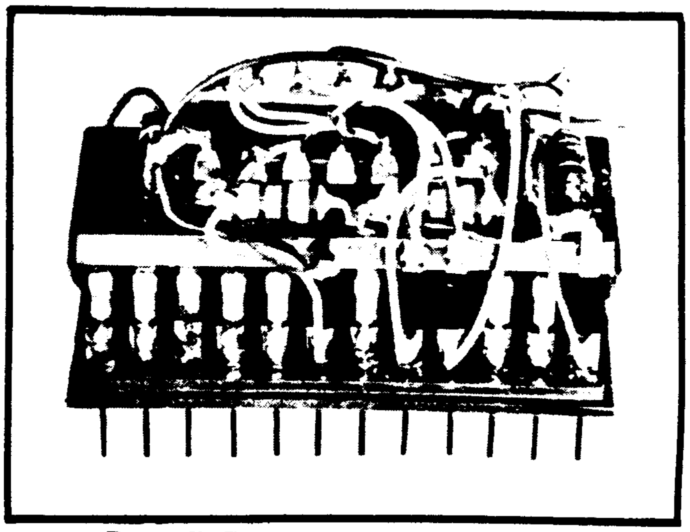


Figure 2b: Enlarged view of the EPRAM.

Checkout

Checking the EPRAM out is really pretty simple. Just plug it into your EPROM programmer and attempt to program something into it. If your programmer provides the supply voltage (I would assume most do, although if it is hacker-built I wouldn't dare make any definite statements), you can skip attaching the battery at this stage. If the EPRAM can be verified after programming you have wired it correctly. If not, examine carefully all the solder connections in the assembly. In order to keep the size small, there are a lot of connections crammed into a small space. Look for solder bridges or shorts. Also make sure all the diodes involved have the correct polarity. If you have a logic probe or a scope, look at the programming signals at the 6116 to see whether they look correct. This is the best way to detect faulty logic. If all else fails it is possible that one of the CMOS chips is bad or has been ruined in assembly. (I don't need to caution you to use a very small soldering iron, do I?) This is one place where sockets come in handy. My first prototype of this used sockets for both chips but the thing ended up so bulky that I waived my normal rule to always use sockets and soldered it together directly. As I stated above, CMOS isn't the bugaboo it once was thanks to diode protection of inputs. Just be careful, try to check out the 6116 and 4011 in another circuit if you can before assembly, and everything should turn out fine.

If all is well and the EPRAM is programmable, the next thing to do is hook up the battery. Remove the EPRAM from the programmer, wait a minute or so and re-insert it. Now verify it again. If the information content hasn't changed, you're ready to start using it. If it doesn't check out, go back and re-examine all connections. Since there are a lot of different EPROM programmers around, it is always possible (but unlikely, I think) that your particular programmer is in some way incompatible with this device. As a hacker I'm afraid it will fall to you to ferret out this type of problem. My own experience with a cheapie, home-built PROM burner is that both units I built worked the first time and have worked every time since. I can replace my boot ROM with this EPRAM and have the computer boot up every time. The EPRAM assembly can even be kept in a piece of conductive foam and not lose its information since "conductive" foam has a resistance in the neighborhood of K-ohms.

Using It

The place I envision this device coming in the most handy is in developing software for ROM-based computers or dedicated controllers. You can program your software into this device, try it out immediately, and if it doesn't work the first time (who ever heard of such a thing), you can make immediate changes to the software and reprogram the EPRAM without bothering to erase it. I think this can be of great help in the software development process. For those of you who have disk-based systems and like to fool around with different boot loaders, you can do it and get instant feedback for the next round of code. Once you have your program the way you want it, you can burn a conventional 2716 to use on a long-term basis.

If you have a commercial EPROM programmer with fixed programming parameters, there will be no problem although you will probably have to wait the full EPROM programming time, something like 100 seconds for a full 2K bytes of code in a 2716. On the other hand, if you're using a homebrew piece of hardware and some software to go with it, you can reduce the timing parameters that control the writing delay for each address location. The 6116 needs no special delay in the write cycle as a true 2716 does. This allows programming the entire device in a fraction of a second. A couple of minutes may not sound like a lot of time, but when you're twiddling your thumbs waiting for something to finish, it can seem like a lot longer.

I think it's only fair to say that eventually the EEPROM (*Electrically* Erasable PROM) will probably provide a way to accomplish what we're doing here. However at the present time, these devices are pretty expensive and require somewhat different programming waveforms than the ordinary UV-erasable EPROMS. The EPRAM should be completely compatible with the latter device and I think you will find it very handy as a development tool. ■

HELP

This space is for you. We encourage you to communicate with other readers through this column by asking for their help with your problems, and by writing in with your solutions to questions like "Where can I...?" or "How can I...?" As a forum for sharing hands-on experience, this section can be an important resource for you. We will try to keep the lead time short for a rapid exchange of information. Let us hear from you!