13

WORLDWIDE REMOTE CONTROL WITH C2TERM

BS2

CONTENTS AT A GLANCE

Want to control or monitor electronic equipment in a remote location? A BS2 and an inexpensive modem team up to create a dial-up communication and control terminal (C2TERM). Using your PC and communication software you can dial up C2TERM from

Programming and Customizing the BASIC Stamp Computer

Scott Edwards

McGraw-Hill

New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

any phone in the world, enter a password, and control lights and appliances throughout your home or business.

This project is a good opportunity to get familiar with asynchronous serial communication and the BS2 instructions Serin and Serout. It also demonstrates the powerful Xout instruction that transmits command signals through the AC wiring to control lights and appliances.

Asynchronous Serial Communication

Serial communication is the process of transmitting data one bit at a time. Asynchronous serial sends the data without a separate synchronizing signal to help the receiver distinguish one bit from the next. To make up for the lack of synchronization, asynchronous serial imposes strict rules for the timing and organization of bits. The reward for obeying these rules is an efficient, reliable way to send data over a single wire.

For the sake of brevity, let's agree that throughout the rest of this chapter "serial" means "asynchronous serial."

SERIAL TIMING AND FRAMING

The basic principle of serial communication is simple—to send multiple bits over a single wire, just place each bit on that wire for a fixed amount of time. For example, suppose you wanted to transmit the byte %10010101 to me in the next room.[1] In your room, you have a switch and a battery that are wired to control a light in mine. We agree beforehand that you will send one bit per second, starting with the lefthand bit, and that light on means 1 and off means 0.

With the rules established, you flick the switch on and off to match the pattern of bits:

Bits	1	0	0	1	0	1	0	1
Seconds:	0	1	2	3	4	5	6	7
Light:	ON	OFF	OFF	ON	OFF	ON	OFF	ON

After our experiment, we meet to see whether the message I received matches the one you sent. I show you my notepad:

```
00000000000000000000000000000000000010010101000000000000000000000000000000
```

Your message was received faithfully, but it's buried in the middle of all those zeros I wrote before and after the actual message. We need another couple of rules: I start copying one second after an initial 1 (light on), and stop copying after receiving eight bits. We'll call that initial 1 the start bit. It's not part of the data, just a signal from you to me that precedes the data. We try again:

1. I'm using the % sign to indicate that the number is in binary notation. This is the same way that you represent binary in a PBASIC program.

check each incoming byte according to some agreed-upon rule. For example, we could add one more bit to the serial frame and call it the parity bit. You would count the 1s in the data to be transmitted to determine whether the total number was even or odd. For an odd number, you'd make the parity bit a 1; for an even number, 0.

When I received the data, I'd perform the same tally and compare my reckoning of odd/even to the parity bit. If they matched, I'd be more confident that my copy of that byte was correct. If they didn't match, I would figure that the data was possibly incorrect. If I had a way to signal you, I would do so and tell you to send that byte over. If not, I'd have to decide what to do with the doubtful byte.

The parity setup described above is called even parity, since the parity bit is set or cleared according to rules that make the total number of 1s (data bits plus the parity bit) an even number. The opposite arrangement is called odd parity. A third common option is no parity at all.

Why do without parity? Parity isn't a sure-fire check for errors. If one bit is incorrect, parity will detect it. If two bits are wrong, it won't. There's also no guarantee that the parity bit itself won't be received incorrectly. And then there's the dilemma of what to do when parity casts doubt on a particular byte.

For all of these reasons, many systems operate without parity. Instead they may send other kinds of double-check information as regular data.

DATA BITS

Since the byte is a common unit of storage for computers, it's natural to assume that serial communications would transfer data in groups of eight bits—one byte. But it's not always so. For example, if you only need to send ordinary text, six or seven bits can be perfectly adequate. In text-oriented applications, it is common to sacrifice one data bit in order to make room for the parity bit. This is one of the serial modes that the BS2 supports: 7E1 or seven data bits, even parity, one stop bit.

Unlike our room-to-room example, the data bits in most serial communication are sent least-significant-bit (lsb; the bit on the righthand end of a number like %11010010) first.

STOP BITS

The stop bit is a pause between the last data bit and the next start bit. In our room-to-room example, we saw that the stop bit allows the receiver to reset timing with each new start bit in order to prevent small timing errors from accumulating over multiple frames of data and eventually causing data errors.

The stop bit must be at least one bit-time long in order for this to work. However, some slow devices might need more than one stop bit between frames in order to do other processing. The specs for such slowpokes would call for one-and-a-half, two, or more stop bits. You may never encounter this specification, but you have to know that it's possible in order to understand why anyone would bother specifying the number of stop bits when it always seems to be 1. After all, the number of start bits is not specified, since it is always 1.

Although the BS2's Serout instruction does not directly support multiple stop bits, it does allow you to specify pacing in milliseconds. Pacing is a delay between frames of data, so it amounts to pretty much the same thing as multiple stop bits. For example, 2400-bps serial data sent with 1-ms pacing amounts to about 3.4 stop bits. There's one stop bit built

into the data frame, plus a 1-ms delay, which amounts to about 2.4 additional bit times. When a specification calls for multiple stop bits, it's really saying "this is the minimum delay between frames that this device can handle," so it's OK to exceed that amount.

A SERIAL FRAME

Now that we're acquainted with all the elements, let's diagram a couple of typical serial frames. First, 2400 bps 7E1:

Time (µs):	0	416.6	833.3	1250	1667	2083	2500	2917	3333	3750
Bits:	start	bit0	bit1	bit2	bit3	bit4	bit5	bit6	parity	stop

Now 2400 bps 8N1, the format our project will use:

Time (µs):	0	416.6	833.3	1250	1667	2083	2500	2917	3333	3750
Bits:	start	bit0	bit1	bit2	bit3	bit4	bit5	bit6	bit7	stop

RS-232 Serial Signals

In addition to the timing and framing of data, serial senders and receivers must also agree on the electrical details of the connection. There are various standards for serial signaling, but the most common is RS-232. It uses signal voltages that are outside the range normally used by the Stamp, but can be readily interfaced by taking a crafty look at the specs.

The Stamp uses 5-volt logic. It outputs 0 volts for a 0 and 5 volts for a 1. When it accepts inputs from other circuits, it regards a voltage less than 1.5 volts as a 0 and greater than 1.5 volts as a 1.

This relationship of voltages to 1s and 0s is common to many digital-electronic devices, but it's not the only one. The serial port(s) on your PC conform to the RS-232 standard, which specifies wider signaling voltages. It makes intuitive sense—the 5-volt logic used to communicate between components separated by a few inches might not be ideal for communication between computers and peripherals separated by 50 feet of cable.

Under the RS-232 spec, a 0 is represented by a higher positive voltage, typically +10V, and a 1 by a negative voltage, typically –10V. The negative voltage is also the stop-bit state; positive is the start bit. The idea is that the larger the voltage difference between a 0 and a 1, the less likely that electrical noise picked up over a long cable would make an RS-232 device mistake one state for the other.[3]

To convert a 5-volt logic level to RS-232 and back normally requires components called RS-232 line drivers and line receivers. However, it's common practice to cheat on the RS-232 standard and do without these parts. Cheating requires a closer examination of the RS-232 rules.

3. Other serial standards, such as RS-422 and RS-485, use the voltage difference between two signaling wires to distinguish a 1 from a 0. This offers much better noise immunity over long wire runs without the need for separate positive and negative power-supply voltages.

An RS-232 sender is supposed to output +5 to +15 volts for a 0. An RS-232 receiver is required to recognize +3 to +15 volts as a 0. The reason for the reduced lower limit (+3V) is to allow for the voltage drop over a long wire run. Looking at this with cheating in mind, a 5-volt logic 1 is equivalent to an RS-232 logic 0.

For a 1, an RS-232 sender is supposed to output –5 to –15 volts. A receiver must recognize –3 to –15 volts as 1, again allowing some voltage drop through the wiring. There's no way to get –3 volts from our 5-volt logic without additional components, but most RS-232 devices also regard a signal that's close to 0 volts (ground) as a logic 1. So, provided that our 5-volt logic's 0 is close to 0 volts, it will suffice as an RS-232 logic 1.

The BS2 has built-in support for this kind of thinking, as you can program its serial-output (Serout) instruction to send inverted serial data. Connect the I/O pin directly to the input of an RS-232 receiver via a short cable and you're ready to go. I emphasize short cable (less than 10 feet), because even a small voltage drop or minor noise on the connection can cause communication errors.

What happens when the Stamp is on the receiving end of a ±15-volt RS-232 signal? Its I/O pins can be damaged by voltages outside the supply range of 0 to 5 volts. However, there's an easy fix. A 22k resistor in series with the ±15-volt line protects the I/O pin. See, each I/O pin is internally protected by a pair of diodes arranged to short out excessive voltages, typically zaps of static electricity. The series resistor limits the amount of current that the diodes have to handle, preventing them from overheating.

The BS2 serial-input instruction Serin has an option that allows reception of inverted serial data. To establish two-way RS-232 communication between the BS2 and a serial port requires a short cable and a cheap resistor. Who says cheaters never prosper?

An even easier route to get serial data into and out of the BS2 is to borrow the programming port. BS2 carrier boards have a DB9 connector that mates with a cable to your PC's serial port.

Figure 13-1 shows how a typical RS-232 frame would look on an oscilloscope screen. (An oscilloscope is a versatile instrument that graphs input voltages versus time.)

Typical RS-232 ports have 9 or 25 pins. Although we have limited our discussion so far to just the data lines, a serial port often includes a number of control lines used for various purposes. The most common use of these lines is handshaking, in which one serial device may indicate that it has data to send and the other signals whether or not it's ready to receive. In our application, we won't be using handshaking. It can be difficult to implement with a relatively slow computer like the Stamp, and is often unnecessary. We will use one of the other control lines—the ring—indicator output from the modem—to determine when to answer the phone.

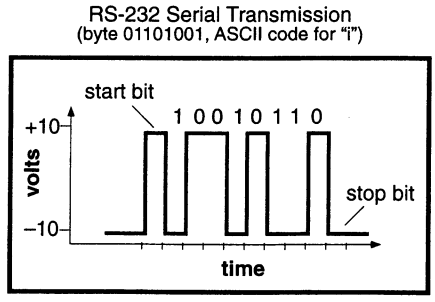

Figure 13-1 One RS-232 serial frame.

TABLE 13-1 RS-232 CONNECTOR PINOUTS

NAME/FUNCTION	DB25 PIN	DB9 PIN
Protective ground	1	—
Transmit Data (TD)	2	3
Receive Data (RD)	3	2
Ready to send (RTS)	4	7
Clear to Send (CTS)	5	8
Data Set Ready (DSR)	6	6
Signal Ground (SG)	7	5
Data Carrier Detect (DCD)	8	1
Data Terminal Ready (DTR)	20	4
Ring Indicator (RI)	22	9

Table 13-1 shows the pinouts of the two common styles of RS-232 connectors.

The names and functions of the RS-232 pins as listed in the table are slightly deceptive. They are valid only for what is known as data-terminal equipment (DTE). A PC is considered DTE, because it can serve as a terminal for sending data. The other flavor of RS-232 devices are called data-communications equipment (DCE). A modem is pretty much the definition of DCE, since its whole purpose is to facilitate communication.

The distinction between DTE and DCE becomes important when you are trying to figure out which pin does what on a particular device. Suppose you need to know which pin transmits serial data. On a PC with a DB25 connector, it's pin 2, the TD pin. On a modem with a DB25 connector, it's pin 3, which the table says is the receive-data pin. It makes a weird sort of sense—the pin through which the PC transmits data to the modem has to be the pin through which the modem receives data! This complementary relationship of inputs and outputs simplifies wiring up cables, since many times you just wire like pins together (1 to 1, 2 to 2, 3 to 3...), but it really messes up the names.

For more on RS-232 interfacing, see the reading list in Appendix E.

Serial Communication by Modem

With the exploding popularity of the Internet and online communication, it hardly seems necessary to define the word modem, but here goes: Modem is a contraction of the words modulator-demodulator, and it refers to a device for sending serial signals via a carrier, like audio tones transmitted by phone or radio. A phone modem typically consists of some analog circuitry for generating and detecting audio tones, and a small, fast computer for interpreting them.

The modem's computer is programmed with routines to communicate with a host computer serially, dial the phone, detect ringing and busy signals, establish a connection with another modem, exchange data, and hang up.

Most modems use the "Hayes" command set, named for the company that set the standard for modems in the early days of personal computing. This language is also sometimes called the AT command set, because most commands begin with AT for "attention."

A modem has two modes of operation—command mode and data mode. When the modem is not linked to another modem, it's in command mode, awaiting instructions from the local computer. The computer can tell the modem to dial or answer a call from another modem. When two modems make contact, they investigate each other through handshaking—a ritual exchange of tones and preliminary data that sets the ground rules for communication.[4]

Once handshaking is complete, the modem goes into data mode. In data mode it acts like a direct connection between local and remote computers. Once in data mode, a modem ignores commands unless it is first returned to command mode by a sort of secret knock (usually "+++" followed by a delay).When the exchange of data is complete, one of the computers issues the secret knock to return its modem to command mode, then instructs it to hang up. The other modem senses this loss of carrier, and hangs up too.

The modem you'll need for this project is an external modem—one designed to be connected through a serial port, not installed inside a desktop computer (internal). Feel free to use an older, "obsolete" modem, since this project uses no advanced features and communicates at only 2400 bps.

A benny of using an older modem is that it usually includes a good manual on the AT command set. In the old days users generally dealt directly with the modem, manually typing commands through a terminal program.

To really understand how a modem works, you should try using it manually with your PC and terminal software. If you have Windows installed, there's a simple terminal "accessory" program. Procomm is a popular commercial package, available in both DOS and Windows flavors.

Although the modem manual lists dozens of commands, you can get by with few important ones. Table 13-2 lists the ones I found helpful in getting the BS2 on line. Once your terminal program is set up and talking to the modem, you may send commands by typing them at the keyboard. For example, if you type "ATDT 5551234" followed by the Enter key, the modem will go off-hook (connect to the phone line) and send the touch-tone digits 555-1234.

To sum up, a modem is a small computer that transports your serial data across the phone lines. Next we'll look at a system for moving data through the AC power lines of your home.

X-10 Appliance Control

Some people dream about a Jetsons-style home in which every appliance and feature is under pushbutton control from the comfort of an easy chair. The technology has been around for years. The only thing missing is some sort of household data network to control all those appliances. Our homes come equipped with wiring for electricity, phones, and lately even cable TV, but no network.

4. The handshaking that modems do to establish communication is similar in concept and purpose to the hardware handshaking done through the control lines of the RS-232 port. But don't confuse the two. For example, when a terminal program offers you the option to turn off handshaking, it means the extra lines on the serial port, not the modem-to-modem handshaking process.

TABLE 13-2 USEFUL AT MODEM COMMANDS

INSTRUCTION	OPERATION
ATDT n	Dial n using touch tones. ATDT 4594802 dials 459-4802. The number to dial can include other symbols that change the way it is dialed, including: , Pause dialing W Wait for second dial tone ; Stay in command mode after dialing
A/	Repeat last command entered.
ATH0	Hang up.
ATA	Answer the phone now (manual answer).
ATV0	Set modem responses to numbers (0–10).
ATV1	Set modem responses to words (e.g., CONNECT 2400).
ATZ	Return modem to default settings.
AT&W	Store configuration settings to nonvolatile memory.
ATS0 = 0	Turn off auto-answer function.
ATS0 = n	Turn on auto-answer and set for n rings.
ATM0	Silence speaker.
ATM1	Turn speaker on while dialing; off during communication.
ATE0	Do not echo commands.
ATE1	Echo commands.
+++	Escape command: shifts modem from data to command mode after a preset time delay (set by register S12).
ATS12 = n	Set escape (+++) delay to n number of 20-millisecond units. For example, ATS12=50 sets the escape delay to 1 second.

In 1978, Sears and Radio Shack introduced household remote-control systems that didn't need separate control wiring. Under this system, called X-10, the control signals ride on the existing AC power wiring. Since an appliance needs power anyway, control signals are available everywhere there is an appliance.

The X-10 code is an oddball kind of serial data format. It is sent as bursts of 120-kHz tones timed to coincide with the zero crossings of the AC power line. It's not necessary to know all of the gory details of this code, only how it is used. The BS2 instruction Xout generates the X-10 code automatically.[5]

5. I'm not providing a lot of detail on the X-10 code format because there's not much practical use for the information. You can't use it from within PBASIC except via the Xout instruction. If you're curious, you can contact X-10 Powerhouse for a detailed tech note; call 201-784-9700 or fax 201-784-9464.

To control an appliance with X-10, you must plug it into an X-10 control module. There are basically two categories of modules: on/off appliance/light controllers and on/off/dim light controllers. Within these groups, you'll find units with different power ratings and other features.

You assign each module an address by setting a pair of dials for a house code (A through P) and a unit code (also called a key, 1-16). Normally, all of the appliances in a given house would be assigned the same house code, but households with more than 16 appliances under X-10 control might use two or more house codes. If a nearby neighbor is using X-10, you want to make sure that your house codes or sets of house codes are different, since X-10 signals sometimes travel house-to-house.

To control X-10 modules from a manual controller, you'd set the house code, press one of 16 keys to send the unit code, then press a button for the desired action, such as ON, OFF, DIM, or BRIGHT.

The BS2 instruction Xout mimics this operation. Just tell it what house, unit, and action codes you want to send, and it takes care of the rest. Of course, before Xout can do anything, the signals from the BS2 must be connected to the power line. The only way to do this is through a safe, optically isolated interface like the X-10 Powerhouse PL-513 or TW-523. Figure 13-2, part one of the schematic for C2TERM, shows the hookup.

The two X-10 signal hookups are called zPin and mPin. The zPin is an output from the X-10 interface that sends a pulse to the BS2 at the instant the AC power waveform crosses

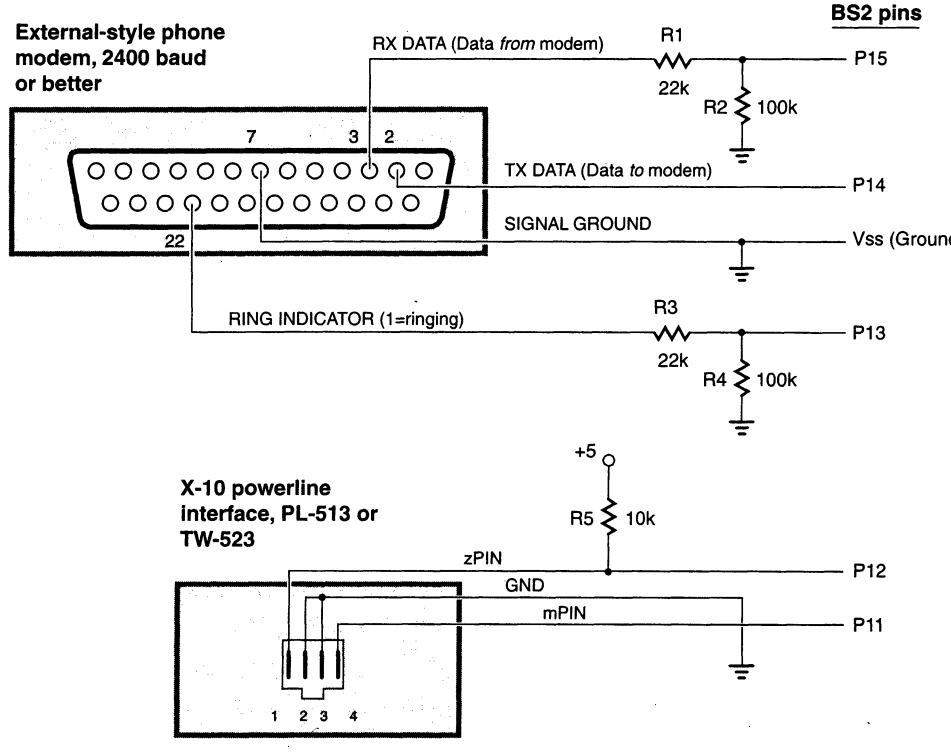

Figure 13-2 Connecting the modem and X-10 controller to a BS2.

zero volts. This cues the BS2 to take control of mPin, the pin that modulates (controls) the 120-kHz X-10 signal. Xout automatically generates the right signals with the right timing to broadcast your X-10 commands.

C2TERM

This is a fairly involved project, so we're going to take it a step at a time. We'll start with the X-10 interface.

INITIAL X-10 CHECKOUT

Connect the X-10 interface (PL-513 or TW-523) to your BS2 as shown in Figure 13-2. Do not connect the modem for now. You may plug the PL-513 or TW-523 into a wall outlet at any time; it's designed to provide a safe, isolated connection to the power line. But heed this warning:

You can be hurt or killed by AC line voltages! Do not open the case of the PL-513 or TW-523 for any reason. These units are designed to be safe, but only in their original, unmodified form.

Set an X-10 lamp or appliance control module to House A, Unit 1 and plug it into a wall outlet near the PL-513 or TW-523. Plug a lamp into the control module and turn it on so that when the module supplies power the lamp will light. Run the following short program:

```
zPin          con   12                    ' zPin on P12.
mPin          con   11                    ' mPin on P11.
houseA        con   0                     ' 0=A, 1=B, 2=C...
Unit1         con   0                     ' 0=Unit1, 1=Unit2...
xout mPin,zPin,[houseA\Unit1]             ' Talk to Unit 1.
xout mPin,zPin,[houseA\uniton]            ' Tell it to turn ON.
pause 1000                                ' Wait a second.
xout mPin,zPin,[houseA\unitoff]           ' Tell it to turn OFF.
stop                                      ' End the program.
```

If all is well, when you run the program, the lamp will come on for 1 second, then turn off. If it doesn't, double-check your setup. In troubleshooting X-10, it can be very useful to have a manual X-10 control box on hand. If you can operate the modules manually, then any problem has to be with the BS2 hookup or programming. Less likely, but still possible, is a problem with the PL-513 or TW-523 powerline interface. But exhaust all other possibilities before letting yourself suspect this.

Once you have the setup working, try modifying the test routine above to address other house or unit numbers. Use additional control modules to check your modifications. Remember that once you have sent a given module's house and unit number, you have its attention. To send subsequent commands, you only need to precede them with the house number, as the test listing demonstrates. This is just like the manual X-10 controllers, so you may find it useful to rehearse X-10 communication by physically pressing the buttons.

MODEM CHECKOUT

Before you can use a modem with the BS2, you have to configure it properly and save those configurations in nonvolatile memory. This requires a temporary hookup to your PC.

You will need a modem cable and simple terminal communication software, such as the free Windows terminal accessory. Set the terminal program for 2400 bps, N81, no handshaking.

For the purposes of configuring the modem, you don't need to connect it to the phone line yet.

Once you have the modem connected and powered up and the terminal software configured, try typing AT <return>. The modem should respond with "OK" or "0" depending on whether it's currently set for text or numeric responses. If you don't get any sort of answer back from the modem, check your cabling and terminal software settings and try again. Some modems have lights that can help with troubleshooting; an "RD" light should flash briefly whenever you type a character in the terminal program.

Once the PC and modem are talking, you can type in the configuration commands shown in Table 13-3. In the table, <Enter> means press the enter key.

The last command, AT&W, causes the modem to commit this new configuration to nonvolatile memory. Even with the power turned off, your modem will remember its new settings.

The next step is to download a test program to the BS2, connect the modem, and run a test. Unless you have two phone lines or a phone-line simulator (see parts list for source), you will need some help with this test. You may have to borrow a friend's computer and modem. If you have a really helpful (and computer literate) friend, you can have him or her dial into your BS2/modem test setup. Make sure that they set up for 2400 baud, N81. Here's the program that will enable your BS2 to answer the phone:

```
tLink       con     20000                   ' Wait 20 seconds for linkup.
N2400       con     16780                   ' Baudmode for 2400 bps inverted.
TxD         con     14                      ' Pin to output serial data.
RxD         con     15                      ' Pin to input serial data.
RI          var     IN13                    ' Ring-indication output of modem.
name        var     byte(10)                ' String to hold user name.

waitForRing:                                ' When phone rings, RI goes high.
 if RI = 0 then waitForRing                 ' Wait here while RI is low.

pickUpPhone:
 serout TxD,N2400,["ATA",cr]                ' Tell modem to pick up.
 pause tLink
 serout TxD,N2400,["Please enter your name: ",cr,lf]
 serin RxD,N2400,[str name\10\cr]            ' Get user name.
 serout TxD,N2400,["Thanks for calling, ", str name\10,cr,lf]
```

TABLE 13-3 CONFIGURING A MODEM FOR BS2 COMMUNICATION

TYPE THIS COMMAND	MODEM RESPONDS	PURPOSE
ATS0 = 0 <Enter>	"OK" or "0"	Disable auto-answer
ATS12 = 50 <Enter>	"OK" or "0"	Set "+++" response to 1s
ATV0 <Enter>	"0"	Set numeric responses
ATE0 <Enter>	"0"	Disable command echo
AT&W <Enter>	"0"	Memorize configuration

```
 pause 1000
 serout TxD,N2400,100,["Hanging up now.",cr,lf]

Disconnect:
 pause 2000
 serout TxD,N2400,["+++"]                    ' Switch to command mode.
 pause 2000
 serout TxD,N2400,["ATH0",cr]                ' Send hang-up command.
goto waitForRing                             ' Ready for another call.
```

This program will answer the phone, wait about 20 seconds for the modems to finish linking up, send a test message, get the user's name, send a customized response, and hang up. If the test message is not received completely, or the BS2 hangs up without the other computer receiving the test message, try increasing the value of the constant tLink. This will make the BS2 wait longer for the modems to finish linking up.

Modem-savvy readers may wonder why I used a time delay to wait for modem linkup when I could have employed the Serin instruction's WAIT option to look for the modem connect message. When modems establish a connection, they send the message "CONNECT" followed by the baud rate and other information to their host computer. When a modem is set for numeric responses, it sends a code number corresponding to the connection details.

In my experiments with BS2s and modems, I found that the BS2 often had trouble catching the connect message, which (with many modems) is accompanied by a bunch of random data. It's inelegant, but more reliable, to simply have the BS2 wait for the modems to do their thing before attempting to send any data.[6] (If it sends data too early, while the modem is still in command mode, the modem will hang up.)

DISPLAY AND BUTTON CHECKOUT

As a final check, we'll add the 4x20 serial display module and the four pushbutton switches that will serve as our user interface. The display works like a simplified, receive-only version of the PC terminal programs you have used to configure your modem. It understands many of the standard control characters, like carriage return, linefeed, tab and backspace, in addition to some specially suited to the LCD such as backlight control, fast cursor positioning, and automatic generation of four-line-tall numeric characters. Table 13-4 lists the control codes.

Connect the display and the buttons as shown in Figure 13-3. Set the LCD for 9600 baud, and Plus command set (configuration switches 1 up, 2 down; see the LCD instruction book for more details). Run the following short program:

```
clrLCD      con   12              ' Clear entire LCD screen.
posCmd      con   16              ' Position cursor.
colTen      con   74              ' Position 10 on the 4x20 screen.
lf          con   10              ' Linefeed control character.
bigNums     con   2               ' Begin big numbers.
N9600       con   $4054           ' Baudmode for inverted, 9600-bps output.
LCD         con   10              ' Serial LCD on P10.
i           var   nib             ' Temporary counter, 0-15.
state       var   bit             ' State of button pin.
```

6. Another explanation for unreliable results with Serin/WAIT during modem linkup stems from the fact that the BS2 cannot simultaneously send and receive serial data. Given that limitation, what happens if the BS2 initiates a Serin in the middle of an incoming byte? That's right; the first 0 in that byte will be mistaken for a start bit, and the byte will be garbled. Subsequent bytes may also be messed up, depending on the rate at which the data is being sent and the distribution of 0s that might be mistaken for start bits.

```
pause 1000                              ' Wait for LCD startup.
serout LCD,N9600,[clrLCD]               ' Clear the LCD screen.
for i = 1 to 4                          ' Print four labels: "Button #:"
 serout LCD,N9600,["Button ",dec i,":",cr]
next
 serout LCD,N9600,[posCmd, colTen]      ' Move to position 10.

checkBtns:                              ' Check each button.
for i = 9 to 6                          ' For each pin, 9 to 6..
 state = 1                              ' If pin=1 then state=1 else state=0.
 if ins & (DCD i) then isOne            ' -See text for explanation-
 state = 0
isOne:
 serout LCD,N9600,[dec state,lf,bksp]    ' Print state, goto next line
next                                    ' ..and backspace over old state.
goto checkBtns                          ' Do continuously.
```

TABLE 13-4 CONTROL CODES FOR 4X20 SERIAL LCD

ASCII VALUE	CONTROL CODE	FUNCTION
0	cntl-@	ignored before buffer; used for time delay
1	cntl-A	cursor to position 0 (home)
2	cntl-B	begin big-number display
3	cntl-C	ignored
4	cntl-D	blank cursor
5	cntl-E	underline cursor
6	cntl-F	blinking-block cursor
7	cntl-G	pulse buzzer output (ring bell)
8	cntl-H	backspace; back 1 space and erase character
9	cntl-I	tab (cursor to next multiple-of-4 position)
10	cntl-J	linefeed; cursor to line below
11	cntl-K	vertical tab; cursor to line above
12	cntl-L	formfeed; clear the screen
13	cntl-M	carriage return; cursor to start of line below
14	cntl-N	turn backlight on
15	cntl-O	turn backlight off
16	cntl-P	accept cursor-position data
17	cntl-Q	clear vertical column
18–31	—	ignored
32+	—	ASCII alphanumeric character set

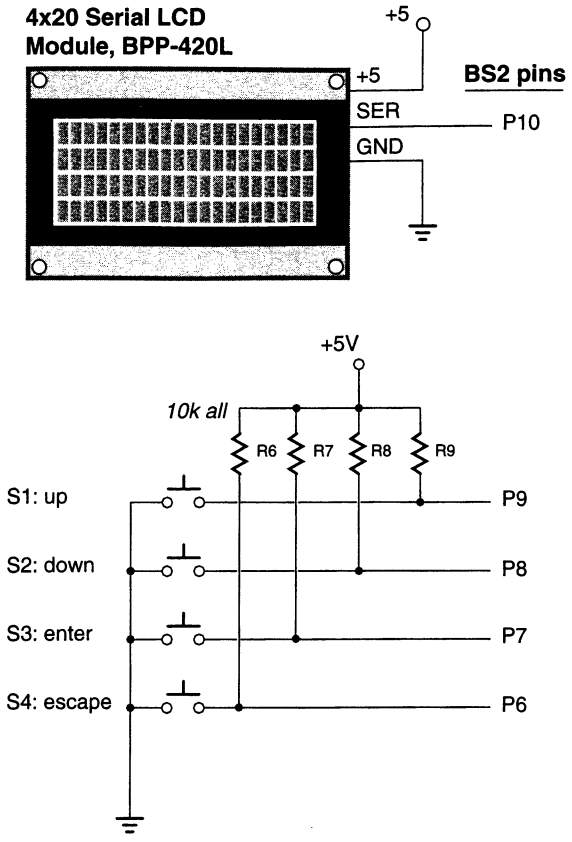

Figure 13-3 User interface consists of a 4x20 display and buttons.

When you run the program, the display will clear, then display the states of the four buttons in the format "Button 1: 1." When you press a button, the corresponding state shown on the display should change to 0. If it doesn't, or if the wrong state changes when you press a particular button, check and correct your wiring.

One part of the checkout program is worth extra attention. The program uses a For...Next loop to check the pins connected to the buttons. For each pin, we want to know whether it's 1 or 0 and print the appropriate state on the display. Unfortunately, there's no direct way to do this. You can write IF IN6 = 1 THEN... but you cannot use a variable to set the pin number to 6. No variable means no For...Next loop. Of course, it wouldn't kill you to use a separate group of instructions for each of the four pins, but that would bloat the program.

There is an indirect way to determine the state of individual bits of a variable using the Stamp's DCD (decode) function. DCD takes a number from 0 to 15 and returns a 16-bit value with a 1 in that position, and 0s in all other positions. Table 13-5 shows how this works:

So we can get a number with a 1 in bit 6 by writing DCD 6. Next we can use that number to determine whether there's a 1 or a 0 in the same position of another number—such as the INS variable that holds the states of the BS2's pins. The logical AND operator (&) combines two values to give a result that contains a 1 in only those positions in which both

TABLE 13-5 THE DCD OPERATOR

FUNCTION	RESULT
DCD 0	%0000000000000001
DCD 1	%0000000000000010
DCD 2	%0000000000000100
DCD 3	%0000000000001000
DCD 4	%0000000000010000
DCD 5	%0000000000100000
DCD 6	%0000000001000000
DCD 7	%0000000010000000
DCD 8	%0000000100000000
DCD 9	%0000001000000000
DCD 10	%0000010000000000
DCD 11	%0000100000000000
DCD 12	%0001000000000000
DCD 13	%0010000000000000
DCD 14	%0100000000000000
DCD 15	%1000000000000000

input values contain 1. If you write INS & (DCD 6), the result will be %0000000000000000 if IN6 is 0 and %0000000001000000 if IN6 is 1. The states of the rest of the pins doesn't matter.

The final piece of the puzzle lies with the If...Then instruction. We normally use If...Then on comparisons, in the form IF x < > 0 THEN notZero. But PBASIC will also let you write IF x THEN notZero, which has the same effect.[7] If...Then regards 0 to mean false and any value other than 0 to mean true. That means that the line If ins & (DCD i) Then... means "If there's a 1 at the pin whose number is stored in variable i, then..."

This is a very valuable technique, and just a sample of the kinds of programming miracles that can be wrought with Boolean logic. For more on Boolean logic, see Appendix B.

COMPLETE C2TERM APPLICATION

Now that you have checked out each subsystem of C2TERM separately, it's time to load the complete program, shown in Figure 13-4, and give it a whirl. C2TERM operates in two modes, local and remote. Under local mode, the program lets you control X-10 devices by pressing the buttons to select options on the display. In remote mode, the program presents similar choices via modem to a remote computer.

7. Be careful when employing this form; the word-logic functions NOT, AND, OR, and XOR normally used with If...Then can produce unexpected results. See the BS2 manual on the CD-ROM for a complete explanation.

The program uses one house code, so it can control 16 X-10 devices. I have assigned example names to the devices which you are welcome to change. One of the program's strong points is its flexible storage and processing of strings, sequences of bytes that make up a text message or label. The BS2 lets us stash strings in unused portions of the program memory and assign names (constants) to their starting addresses. The program takes care of the rest.

Efficient handling of strings is vitally important to this program. In order to be user-friendly, the program is very chatty—substituting descriptive names for X-10 unit codes.

Embedding lots of text in a PBASIC program is a fast way to run out of program space. For one thing, each character of a text string takes a full byte of storage space. For another, many programmers embed each string in a separate Serout instruction. Think about that

```
'PROGRAM: X10CTL.BS2 (X-10 local and remote control)
'This program interfaces a BS2 to a modem, X-10 powerline
'device, serial LCD module, and switches to provide user-
'friendly remote and local control of 16 X-10 appliances.
'For local control, a user can view the name and ON/OFF
'status of the X-10 device on the LCD screen. By pressing
'UP/DOWN/ON/OFF buttons, the user can pick an appliance and
'command it on or off.
'For remote control, the program monitors the ring-indicator
'output of a modem. When the phone rings, the BS2 answers
'it and requests a password. If the password matches, it
'allows the logged-on user to view and change the states of
'the X-10 appliances.

'=============================================================
'               CONSTANTS FOR SERIAL LCD MODULE
'=============================================================
'These constants define characters that help format the 4x20
'serial LCD screen. Some formatting characters like CR
'(carriage return) aren't on this list, because they are already
'defined by the BS2 for Debug and Serout.
clrLCD  con     12      ' Clear entire LCD screen.
posCmd  con     16      ' Position cursor.
colTen  con     74      ' Position 10 on the 4x20 screen.
lf      con     10      ' Linefeed control character.
N9600   con     $4054   ' Baudmode for inverted, 9600-bps output.
LCD     con     10      ' Serial output pin for LCD (P10).
arrow   con     ">"     ' Pointer to highlight selected item.
statCol con     17      ' Column in which to show ON/OFF status.
pntrCol con     statCol-1       ' Column in which to show arrow.
'=============================================================
'           CONSTANTS FOR MODEM COMMUNICATION
'=============================================================
tLink   con     20000   ' # of milliseconds to wait for link up.
N2400   con     $418D   ' Baudmode for 2400 bps inverted.
TxD     con     14      ' Pin to output serial data to modem.
RxD     con     15      ' Pin to input serial data from modem.
FF      con     12      ' Form-feed code—clears terminal screen.
'=============================================================
'                       X10 CONSTANTS
'=============================================================
myHouse con     0       ' House code—0=A, 1=B, 2=C...
zPin    con     12      ' zPin on P12.
mPin    con     11      ' mPin on P11.
```

Figure 13-4 Program listing for C2Term.

```
'============================================================
'                    X10 DEVICE NAMES
'============================================================
'The Data directives below define the names of the X-10 devices.
'You can change these names—just make sure they are 16 characters
'or less, and end in the ASCII null character (0). Subroutines
'that use these strings start at the address constant (d0, d1, etc.)
'and continue reading data from EEPROM until they find a null.
'This is a common and efficient way to store and retrieve
'text strings of varying length.
' ADDRESS              STRING DATA
'CONSTANT              (CHARACTERS/CONTROLS)          NULL
'--------------------------|---------------
 d0             DATA    "Address",                    0
 d1             DATA    "Path",                       0
 d2             DATA    "Porch",                      0
 d3             DATA    "Pool/Spa",                   0
 d4             DATA    "Alleyway",                   0
 d5             DATA    "Garden",                     0
 d6             DATA    "Garage",                     0
 d7             DATA    "Kitchen",                    0
 d8             DATA    "Living Rm",                  0
 d9             DATA    "Dining Rm",                  0
 d10            DATA    "Master Bdrm",                0
 d11            DATA    "Pool Pump",                  0
 d12            DATA    "Spa Heat",                   0
 d13            DATA    "Workshop Pwr",               0
 d14            DATA    "Attic Fan",                  0
 d15            DATA    "Holiday lights",             0
'============================================================
'                OTHER TEXT STRINGS
'============================================================
'Note that longer strings can be broken into two or more
'lines. The only thing that matters is the end-of-string
'marker, ASCII null.
 ON             DATA    "ON ",                        0
 OFF            DATA    "OFF",                        0
 _ON            DATA    "  ON",cr,lf,                 0
 _OFF           DATA    "  OFF",cr,lf,                0
 answPhone      DATA    "ATA",cr,                     0
 Logon  DATA    "X-10 WORLDWIDE CONTROL",cr,lf
 Prompt DATA    "Please enter your "
 Prompt2        DATA    "password: ",                 0
 Standby        DATA    clrLCD,"***Remote Access***"
 SB2            DATA    cr,cr, "  Please stand by",   0
 hangUpNow      DATA    cr,lf,"Hanging up now.",cr,lf, 0
 pwdOK  DATA    cr,lf,"Logged on.",cr,lf,       0
 offerChoices   DATA    cr,lf,"Enter a device # (1-16)"
 choice2        DATA    cr,lf,"to toggle its state",cr,lf
 choice3        DATA    "or 0 to log off",cr,lf,      0
 logOffNow      DATA    "Log off now (Y/N)? ",        0
 confirm        DATA    cr,lf,"Confirm (Y/N), device: ",      0
'============================================================
'                  STORAGE VARIABLES
'============================================================
'Some variables are used by both the LCD and modem routines.
'This works because the program does not attempt to service
'the LCD/buttons and modem at the same time.
strAddr var     word    ' Address of string in EEPROM.
```

Figure 13-4 ***(Continued)***

```
baud    var     word    ' Baud rate for stringOut.
serPin  var     nib     ' Serial output pin for stringOut
LCD_mdm var     bit     ' String to LCD (0) or modem (1).
char    var     byte    ' Character to send to LCD or modem.
reply   var     byte    ' User's reply to modem prompt.
item    var     nib     ' Selection from list of strings.
stats   var     bit(16) ' Status (ON/OFF) of the X10 devices.
slectn  var     nib     ' Currently selected item.
tempN1  var     nib     ' Nibble-sized temporary counter
tempN2  var     nib     ' "      "     "          "
'=============================================================
'                    I/O VARIABLES
'=============================================================
'Most I/O is done by instructions that use pin-number
'constants; see the constants for the LCD and modem. These
'variables are used in IF/THEN instructions for simple input.
RI      var     in13    ' Ring-indication output of modem.
upSw    var     in8     ' Button to move selection arrow up.
dnSw    var     in9     ' Button to move selection arrow down.
onSw    var     in7     ' Button to turn X10 device on.
offSw   var     in6     ' Button to turn X10 device off.

'=================================================================
'                          MAIN PROGRAM
'=================================================================
Initialization:
pause 2000              ' Wait for LCD startup.
gosub newScreen         ' Display first screen.
gosub showPntr          ' Show the selection pointer.
'Main program loop: The program continuously checks the states of
'the switches and the ring-indication input from the modem.
'When any of these become active, it jumps to appropriate routines
'(answer the modem, move the selection pointer, send an X-10
'command, etc.).
Main:
 if RI=1 then getModem
 if onSw = 0 then turnOn        ' Turn on an X-10 device.
 if offSw = 0 then turnOff      ' Turn off " "     "
 if (upSw & dnSw) = 1 then main ' If neither up or down pushed, try again.
 if upSw = 1 then tryDown       ' If upSw isn't pushed, check dnSw.
  gosub hidePntr                ' Up switch: prepare to move pointer.
  tempN2 = tempN2+1             ' Increment temporary selection.
  if tempN2 & %11 <> 0 then posPntr    ' If user pressed "up" and new
  gosub newScreen               ' selection ends in %00, switch screens.
  goto posPntr                  ' Reposition the pointer.
tryDown:                        ' Check the down switch.
 if dnSw = 1 then main          ' Not pressed? Back to main.
 gosub hidePntr                 ' Pressed: prepare to move pointer.
 tempN2 = tempN2-1              ' Decrement temporary selection.
 if tempN2 & %11 <> %11 then posPntr    ' If user pressed "down" and new
 tempN2 = tempN2 & %1100        ' selection ends in %11, switch screens.
 gosub newScreen                ' Show new screen.
posPntr:                        ' Reposition the pointer.
 slectn = tempN2
 gosub showPntr
hold:
 pause 50                               ' Brief (50-ms) delay, then make
 if (upSw & dnSw) = 0 then hold         ' sure that switch is released
goto main                               ' before returning to main loop.
```

Figure 13-4 ***(Continued)***

```
'=================================================================
'                          MODEM CONTROL
'=================================================================
getModem:
 strAddr = Standby              ' Put standby message on LCD.
 gosub stringOut
 LCD_mdm = 1                    ' Direct strings to modem.
 strAddr=answPhone              ' Tell modem to pick up.
 gosub stringOut
 pause tLink
 strAddr=Logon                  ' Send logon message.
 gosub stringOut
 serin RxD,N2400,5000,Disconnect,[WAIT ("X10_OK")]      ' Get PASSWORD.
 strAddr=pwdOK                  ' Confirm password OK.
 gosub stringOut
mdmStats:
 serout TxD,N2400,[FF]  ' Clear user's terminal screen.
 for item = 0 to 15             ' Display status of 16 devices.
  serout TxD,N2400,[DEC2 (item+1),". "]  ' Number device list 1-16.
  gosub pickStr         ' Get address of the device name.
  gosub stringOut               ' Print the name.
  lookup stats(item),[_OFF,_ON],strAddr ' Now print ON/OFF.
  gosub stringOut
 next                           ' Continue for all 16 devices.
choices:
 strAddr=offerChoices           ' Let the user pick an action:
 gosub stringOut                ' 1-16=toggle device; 0=log off.
 serin RxD,N2400,10000,Disconnect,[DEC reply]     ' Get reply.
 if reply = 0 then Done ' 0 is "quit"
 slectn = reply-1               ' Selection is 0-15; reply is 1-16.
 strAddr=confirm                ' Verify choice.
 gosub stringOut
 serout TxD,N2400,[DEC2 reply,"->"]     ' Show device number.
 lookup stats(slectn),[ON,OFF],strAddr  ' and new state.
 gosub stringOut
 serin RxD,N2400,10000,Disconnect,[reply]   ' Get reply.
 if reply = "Y" or reply = "y" then switchIt
 goto mdmStats
switchIt:
 if stats(slectn) = 1 then turnOFF
 goto turnON
Done:
 strAddr=logOffNow                              ' Confirm logoff.
 gosub stringOut
 serin RxD,N2400,10000,Disconnect,[reply]       ' Get reply.
 if reply = "Y" or reply="y" then Disconnect    ' If (Y)es, hang up.
 goto mdmStats                  ' Else redisplay choices.
Disconnect:
 strAddr=hangUpNow              ' Send hang-up message.
 gosub stringOut
 pause 2000
 serout TxD,N2400,["+++"]       ' Put modem into command mode.
 pause 2000
 serout TxD,N2400,["ATH0",cr]   ' Tell it to hang up.
 LCD_mdm = 0                    ' Reroute serial to display.
goto Initialization             ' Re-initialize the display.
'=================================================================
'                          SUBROUTINES
'=================================================================
'==stringOut:  Output EEPROM strings to LCD or modem.
'The bit variable LCD_mdm picks the output device; 0=LCD
```

Figure 13-4 (*Continued*)

```
'1= modem. The address in variable strAddr is the starting point
'of the string in EEPROM. The routine outputs EEPROM bytes until
'it reaches the end-of string character (null).
stringOut:
 baud = N9600: serPin = LCD     ' Output to LCD if LCDmdm=0
 if LCD_mdm = 0 then getByte
 baud = N2400: serPin = TxD     ' Output to modem if LCDmdm=1
getByte:
 read strAddr,char              ' Get the character.
 if char <> 0 then continue     ' If char is 0, then return
return
continue:                       ' ..else continue
 serout serPin,baud,[char]      ' ..and send char to the LCD.
 strAddr = strAddr+1            ' Point to next character in string.
goto getByte                    ' Repeat until char = 0.
'=====================================================================
'==pickStr: Get starting address of names for X10 devices 0-15
'and place in variable strAddr.
pickStr:
lookup item,[d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15],strAddr
return
'=====================================================================
'==showStat: Display the status (ON or OFF) of the currently
'selected X10 device. That cluttered Serout instruction tells the
'LCD to expect a position value (posCmd), then calculates what the
'current position should be. It takes the last two bits of the
'selection (slectn & %11) to get the LCD line number. Multiplying
'that by 20 gets the beginning of one of the 4 lines, 0-3. Adding
'the constant statCol (17) gets the exact position—character 17
'of the selected line. Finally, adding 64 gives the single-byte
'value that the serial LCD expects. Once the cursor is in position,
'the routine looks up the address for the text that says "ON" or
'"OFF" depending on the state of the selected X10 device, and goes
'to stringOut to print that text on the LCD. Since stringOut is
'the last instruction in the routine, a Goto is used instead of
'a Gosub. This saves an unnecessary Return instruction.
showStat:
 serout LCD,N9600,[posCmd,(((slectn & %11)*20)+statCol)+64]
 lookup stats(slectn),[OFF,ON],strAddr
 goto stringOut
'=====================================================================
'==show/hidePntr: Show or hide the arrow that points to the currently
'selected X10 device status. Basically the same kind of cursor-
'positioning job as showStat above; see those comments for explanation.
showPntr:
 serout LCD,N9600,[posCmd,(((slectn&%11)*20)+pntrCol)+64,arrow]
return
'==
hidePntr:
 serout LCD,N9600,[posCmd,(((slectn&%11)*20)+pntrCol)+64," "]
return
'=====================================================================
'==newScreen: Display a screenful of X10 status information on LCD.
'This routine clears the LCD and then writes the names and states
'of four X10 devices to it. Which four devices is determined by bits
'2 and 3 of the variable tempN2, a nibble variable that keeps count
'of the arrow position on the screen.
newScreen:
 serout LCD,N9600,[clrLCD]      ' Clear the LCD screen.
  for tempN1 = 0 to %11 ' Display four devices/states.
    slectn = tempN2 | tempN1    ' Combine bits 2,3 of tempN2 with
```

Figure 13-4 (*Continued*)

```
    item = slectn                ' ..bits 0,1 of tempN1.
    gosub pickStr                ' Get the string.
    gosub stringOut              ' Display it.
    gosub showStat               ' Show device status
  next                    ' ..for four devices
return                           ' ..and return.
'================================================================
'==turnON/OFF: Turn the selected X10 device on or off in response
'to the buttons. After the X10 code is sent, this subroutine
'makes sure that the user has released the on or off button
'in order to avoid sending redundant codes.
turnON:
 stats(slectn) = 1
 xout mPin,zPin,[myHouse\slectn]          ' Talk to unit
 xout mPin,zPin,[myHouse\uniton]          ' Tell it to turn ON.
 if LCD_mdm = 1 then mdmStats             ' If modem selected, skip LCD.
 gosub showStat                           ' Update the LCD.
hold1:
 if onSw=0 then hold1                     ' Wait til button up.
goto main                                 ' Back to main loop.

turnOFF:
 stats(slectn) = 0
 xout mPin,zPin,[myHouse\slectn]          ' Talk to unit
 xout mPin,zPin,[myHouse\unitoff]         ' Tell it to turn OFF.
 if LCD_mdm = 1 then mdmStats             ' If modem selected, skip LCD.
 gosub showStat                           ' Update the LCD.
hold2:
 if offSw=0 then hold2                    ' Wait til button up.
goto main                                 ' Back to main loop.
```

Figure 13-4 ***(Continued)***

for a minute: each Serout instruction has to be stored as a tag that identifies the instruction as "Serout;" an I/O pin number from 0 to 16 (I/O pins 0 to 15, plus 16, representing the programming port); and a baudmode representing the serial data rate and format.

Making some educated guesses, let's assume that the name Serout is represented by 7 bits, the pin number by 5 bits and the baudmode by 16 bits—that's 28 bits (nearly 4 bytes) of program memory used—before Serout has said anything! This program has 22 occasions to send data serially, so it might have had more than 80 bytes of overhead associated with using Serout.

Instead, the program stores text as strings and uses a routine called stringOut to reduce 22 Serouts to just 10. Now there's also overhead involved with Gosub and Return, but chances are good that the technique saved perhaps 50 bytes of program memory.

The program also manages to be frugal with program memory in another way. It uses a bit variable called LCD_mdm to switch stringOut and other subroutines between the LCD and the modem. When LCD_mdm is 0, the subs talk to the LCD; when it's 1, they talk to the modem. This saves us from writing near-identical, but separate, code for the LCD and modem.

Going Further

Why the mania about conserving code space in this program? Well, I just know that after the initial novelty wears off, you're going to want to add more features to this program,

and I wanted to leave you plenty of room. The same philosophy applies to the hardware design; I could have interfaced the LCD directly (as in the RS-485 terminal project), but that would have deprived you of both code space and I/O pins.

An exciting use for those extra I/Os might be to add some monitoring capability to the program. Add a temperature sensor here, an intrusion detector there, and pretty soon you have a state-of-the-art home-control system—all based on a computer that fits in less than one square inch.

PARTS LIST

Resistors (all 1/4W, 10% or better)
R1, R3—22,000 ohms
R2, R4—100,000 ohms
R5–R9—10,000 ohms
Other Components
S1–S4—any normally-open pushbutton switches
External-type phone modem, AT command set, 2400 baud or better
X–10 powerline interface, model PL-513 or TW-523 (Parallax or home-automation suppliers)
4x20 Serial LCD—model BPP-420L (Parallax, Jameco, JDR Microdevices, or Scott Edwards Electronics, Inc.)
Phone-line simulator kit—Party Line by Digital Products Company.