

DEPARTMENTS

74 Firmware Furnace

82 Practical Algorithms

92 From the Bench

100 Silicon Update

108 Domestic Automation

111 ConneCTime

FIRMWARE FURNACE

Ed Nisley

Two-Way Power Line Communication

The major portion of most of today's home control systems is power line communication. Ed shows us his version of an X-10 power line interface that can be used with the HCS II or in other applications.

Contrary to what you might suppose, Steve really can be jured out of town. But guess who baby-sits his Home Control System. Right. I get a page of X-10 module addresses, the HCS reload sequence, and a backup diskette. I'm supposedly prepared for anything short of plate tectonic activity.

The day after he left on the last jaunt, his main garage door latch clattered onto the floor. No fried electronics, no smoking power drivers, not even a program bug: a piece of pot metal with remarkable timing picked that day to end its service life.

Oft' the best laid plans...

X-10 remote control modules are much the same. They are utterly reliable for years, then snap on at 2:30 A.M. every morning for a week, and revert to normal without a hint as to what changed. Steve happened to be acquainted with this feature and designed his original HCS to remind the modules of their state in life every four minutes; a glitched module is reset by the next refresh command.

Elsewhere in this issue you will find Steve's description of Circuit Cellar's new generation home control system. Ken's doing hard work on the supervisory controller (SC) and command compiler, and I'm supplying a few widgets. This column describes

the bidirectional power line interface (PL-Link) that the new HCS II uses to control the X- 10 modules scattered throughout the house. The PL-Link has a simple job: convert the SC's orders into X- 10 command sequences and keep the X- 10 modules in line by resending those commands every few minutes.

"The Smart X-10 Controller PL-Link" earns its moniker because it listens to the AC line and maintains a table of module states based on what it hears. Turn a lamp on using a standard X-10 control unit, and PL-Link takes note and refreshes that lamp ON until you (or the HCS II's supervisory controller) turn it off. The SC can extract module states from PL-Link and adjust itself to manual overrides without special one-time program changes, so the whole system is, if not truly intelligent, at least dimly aware.

Because we have covered the X- 10 specs (in gory detail) in previous issues, I will describe how to coerce C into producing X-10 messages. The language is Dave Dunfield's Micro-C, which has sprouted three new memory models and a host of other features since the last issue.

TAKING COMMANDS

The new HCS II supervisory controller communicates with its peripherals through bidirectional RS-485 transceivers using a single twisted pair of wires. The peripherals are all passive until the SC sends them a command or status inquiry; only after decoding a properly formatted and addressed command may a peripheral send anything to the SC. In fact, the peripherals cannot echo command characters because RS-485 communications are half-duplex.

The RS-485 messages must include a header and a fixed-format address field to select the appropriate peripheral, so even though the commands use plain ASCII text, they are fiendishly difficult to type (macro keys help!). PL-Link includes a demo mode that eliminates the address overhead and echoes each character as it is received, giving you manual control using a standard communications program. A text file included with the

D	Dump program status (debugging use)
E	Show and clear error flags (debugging use)
Ln	Set logging mode (bit mapped)
	L report current mode
	L0 disable (default)
	L1 show received X-10 messages
	L2 show transmitted X-10 messages
	L4 show refresh changes
Nn	Set network/interactive mode
	N report current mode
	N0 set interactive mode
	N1 network mode (no error messages) (default)
	N2 network mode with command echo (no err msgs)
P	Report power failure status: n,n
	first digit = 1 if power is currently OFF
	second digit = 1 if power failed since last P command
Cl	Query X-10 module status
	Q report all modules for all housecodes
	Qh00 report housecode h: h=sss... (mod 01 first)
	O=off 1=on/dim/bright X=not used
	Ohmm report module mm for housecode h: hmm=ss
	OF=off ON=on/dim/bright XX=not used
R	Set refresh period
	R report current period in seconds
	Rmm set period in minutes
	RSss set period in seconds
	RCmm clear refresh buffer, set period in minutes
S	Send X-10 message
	Shmmff send function ff to housecode h module mm
	Sh01AF send ALL UNITS OFF command to housecode h
	Sh01AN send ALL LIGHTS ON command to housecode h
	Shmmffrr send function ff to housecode h module mm
	function message repeated rr times
	(may be strung together: Shmmff,hmmffrr,hmmff)

Figure 1—The PL-Link responds to the above commands sent over the serial interface. In normal mode, each command must be preceded by the network header and controller's address (LX-10.). Note that the underscores are actually spaces in the message sent

downloadable code shows how to activate "human" mode.

Figure 1 shows PL-Link's command set. There are only three really useful commands: send an X- 10 message, query the status of a house code or single module, and clear the refresh tables. The remaining commands are handy, but the Big Three do most of the work.

PL-Link uses serial interrupt handlers, ring buffers, and interface routines similar to those I have presented in the past and don't require description here. For me to say that serial interrupts are set to low priority allowing the high-priority power line interrupts to occur without delay is enough. The timer and AC line sync interrupts require only 100-200 microseconds, so they do not cause lost characters.

The Micro-C run-time library includes serial port drivers, but they are not appropriate for the task at hand

because they assume ordinary "human" serial input instead of a robot network. Such is firmware....

Listing 1 shows the code invoked by a "Send" command. The outer `while` loop picks off repeated X- 10 commands following the "S" character, extracts and validates each set of parameters, and sends the appropriate X- 10 bit sequences to the power line using the `PwrSendSeq()` function. The `RefreshNewEvent()` function enters the module's ON/OFF state in the refresh table.

A complete X- 10 transmission consists of a house and device code to select a particular module, followed by a house and function code to tell that module what to do. The house/device and house/function messages are normally sent twice, except for the DIM/BRIGHT functions, which may be sent up to 32 times. There should be no delays between repetitions, but there must be a delay between the

house/device and house/function groups.

The `PwrSendSeq()` function shown in Listing 2 translates the house, device, function, and repeat values into X-10 transmission sequences (as I understand the rules, anyway!). In most cases, there will be four X-10 transmissions for each PL-Link command, each with a different bit pattern or length.

`PwrFormatMsg()` creates the bit patterns and inserts them into a ring buffer, which is emptied by an interrupt routine that sends each bit at the right time.

Before I discuss how bits are sent, an examination of where the bits come from and go to in the hardware is necessary.

BURSTS TO BITS AND BACK

As you can see from Figure 2, there are two big hardware blocks: the TW523 bidirectional power

line interface and a COMM-Link microcontroller. You may substitute your favorite microcontroller (with some code changes, of course), but the TW523 is essential. In effect, it is a modem that converts digital data into X-10 power line signals and returns digital data when it hears a valid X-10 command on the power line.

The TW523 provides optical isolation between your computer circuitry and the power line's AC voltages and currents. Unlike the earlier transmit-only PL513, the TW523 has two open-collector outputs: in addition to the 60-Hz square wave for zero-crossing synchronization a new bit presents X-10 data pulses received from the power line. Both are open-collector transistors driven by optoisolators, which require external pull-up resistors. The internal 8051 pull-ups are entirely satisfactory for this purpose.

Driving the TW523 data input requires a buffer transistor because an 8051 I/O bit can neither source nor

sink the 4.5 milliamps required to drive the optoisolator LED. The PL-Link board shown in the photos has a 2N2907 connected to INT1 for this purpose.

In addition to those three "mandatory" I/O pins, the firmware also provides for three optional LED outputs: a heartbeat to indicate the main loop is running, an indicator that goes on when an X-10 message is being sent, and a power failure indicator. Steve's HCS II runs from +12-volt gel cell batteries, so the X-10 controller is

and when to sample the TW523 data output for received bits.

Photo 1 shows how the firmware produces the right bits at the right times. The primary synchronization comes from the 60-Hz signal shown in the upper trace, which triggers the 20 interrupts from Timer 0 shown on the lower trace. There are three groups of three interrupts in each half cycle that mark the beginning, middle, and end of each X-10 data bit, and a tenth interrupt to handle "setup" functions for the next half cycle. All of these

Listing 1—PL-Link's command decoding uses a simple switch table to identify each single-character command. This shows what the PL-Link does in response to a "Send" command.

```
case 'S' : /* send X-10 command */
    while (*pChar){ /* handle multiple parms */
        ParseErr = CmdGetXArgs(&pChar);
        if (ParseErr){
            Nputstr("*** syntax error\n");
        }
        else {
            PwrSendSeq(CmdHouseNum,CmdModuleNum,CmdFuncNum,CmdRepeats)
            RefNewEvent(CmdHouseNum,CmdModuleNum,CmdFuncNum,CmdRepeats);
        }
    }
    break;
```

the only unit that knows when the power goes off!

The PL-Link's RAM and EPROM configuration depends on the C program's memory model. For a Micro-C Compact model program, the EPROM is at 0000-7FFF and the RAM at 8000-9FFF, with Code and Data spaces overlapped. I edited the start-up code to define those addresses, but your compiler may require different contortions. In any event, make sure the code matches the hardware!

So much for the machinery. Now for the tricky part.

PRECISION PULSES REDUX

The key to transmitting and receiving X-10 messages is maintaining synchronization with the AC power line. The TW523 provides a 60-Hz signal that identifies both zero crossings, but the firmware will be interrupted only by the falling edge. Further, there are tight specs controlling when the three X-10 transmitted bits must occur within each half cycle

interrupts occur every power-line cycle, so there are 20 x 60 or 1200 interrupts per second.

A twenty-first timer interrupt will occur if (when?) the 60-Hz input disappears. A state machine triggered by that interrupt times out after eight missing cycles and flags a power failure; there must be eight consecutive good cycles before power becomes "OK" again. This feature copes with power line frequency variations down to 55 Hz as well as glitches without flagging too many errors. The HCS SC can find out whether the power is currently off and whether it has failed since the last status report.

During power failures the timer interrupts free-run at about 55 Hz, with the twenty-first interrupt filling in for the 60-Hz signal. The rest of the firmware is unaware of the change, although X-10 messages don't go anywhere and there is no received data. The program in the supervisory controller decides how to recover from the failure and what to tell the X-10

Listing 2—Each PL-Link order can produce several X-10 commands. This function adds the proper commands to the transmitter ring buffer for each Send order.

```

/*-----*/
/* Create X-10 message given all the information... */
/* This produces several messages with varying repeats for each part... */
/* House, device, function are all indexes into the ACLINE tables */

PwrSendSeq(House,Device,Func,Repeats)
WORD House,
WORD Device,
WORD Func;
unsigned int Repeats;
{
    PwrFormatMsg(House, Device, XTMSGSHORT, 1); /* no pause after msg */
    PwrFormatMsg(House, Device, XTMSGBITS, 1); /* short pause */

    Repeats = max(1,Repeats); /* ensure good repeats */
    Repeats = min(255,Repeats);

    if (1 == Repeats) {
        PwrFormatMsg(House,Func,XTMSGBITS,1); /* ensure pause */
    }
    else {
        PwrFormatMsg(House,Func,XTMSGSHORT,Repeats-1); /* no pauses */
        PwrFormatMsg(House,Func,XTMSGBITS,1); /* short pause here */
    }
}

```

adding excessive overhead on time-critical parts of the code.

The TW523 receiver is independent of the transmitter, so outgoing messages are echoed back to the PL-Link board. PL-Link updates its refresh table based on both transmitted and received messages to ensure the table always has the latest information. The refresh table remains current even when the TW523 is disconnected or the power fails since outbound messages need not be echoed.

Photo 2 shows a complete X-10 transmission as seen at the PL-Link I/O pins. There are two pairs of X-10 messages: the first two set the house and

modules when they wake up. When power returns, the timer interrupts will lock to the 60-Hz interrupt and normal operation resume.

The interrupt handlers are written in 8051 assembly language using Micro-C's in-line assembler to embed the code within C functions. Suffice it to say that the code picks a bit from the tail of the transmitter's ring buffer entry, sends it during each of the three bit times, then steps to the next bit. A variable determines how many bits are sent in each message, and when the bits are finished the code chucks up the next ring entry and starts over again.

The firmware also samples the TW523 receiver output at the middle of the first bit time in each half cycle, even while transmitting a message. A state machine finds the start sequence (four successive half-cycles with the otherwise invalid bit pattern 11 10), validates the next 18 half-cycles to assemble a complete message, and adds it to the head of the receiver's ring buffer.

The X-10 transmitter and receiver ring buffers are located in External

RAM. The interrupt routines use Internal RAM variables to speed access to the bits and touch External RAM only when a message is complete. This timing provides space for many incoming and outgoing messages while not

device and the second two set the house and function. Remember the 2N2907 transistor inverts outgoing data bits and the TW523 presents inverted bits to INTO; in both cases, a "low" voltage indicates a "1" bit.

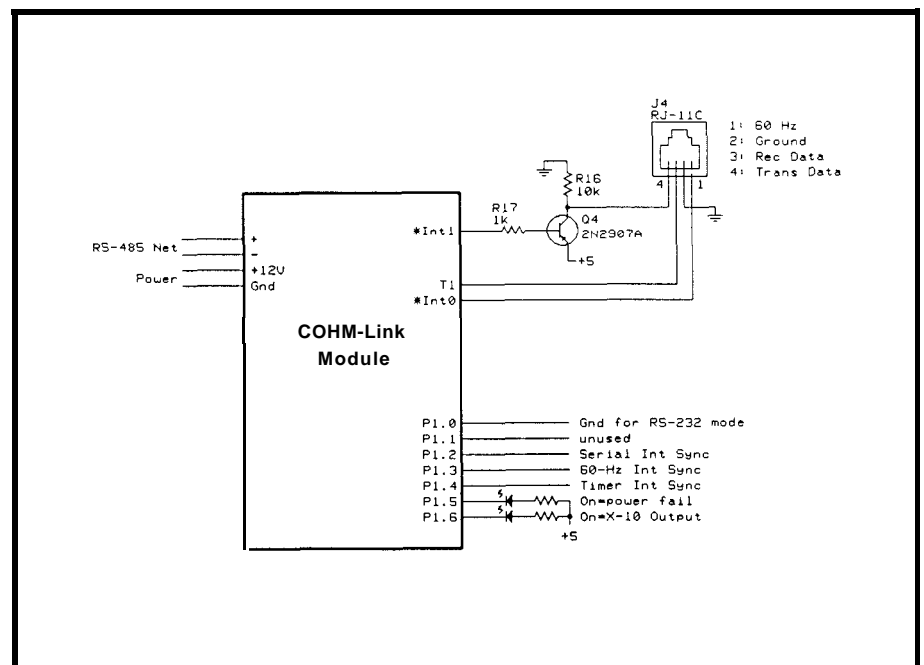


Figure 2—The PL-Link module uses the standard COMM-Link as its basis and adds a TW523 interface. Note that the "heartbeat" LED connected to P1.7 is on the COMA-Link board, so isn't shown here.

Interestingly, the TW523 does not behave quite like the documentation would have you believe. The receiver output is actually a stored copy of the most recent valid X-10 message; in effect, the TW523 records a message and plays it back immediately, regardless of what happens on the power line. It cannot record and play simultaneously, so you must send each message twice to prevent data loss.

For example, if you send one house/device message (say, "N" and "02") followed by a single house/function message ("N" and "ON"), the TW523 will return only the house/device message. X-10 modules do not have this restriction, so the lamp at NO2 will turn ON. Spending a little time sending various combinations of messages to see how the TW523 reacts is worthwhile.

MODELING MEMORY

The key to writing good firmware is keeping track of your variables. The 8051 architecture provides three

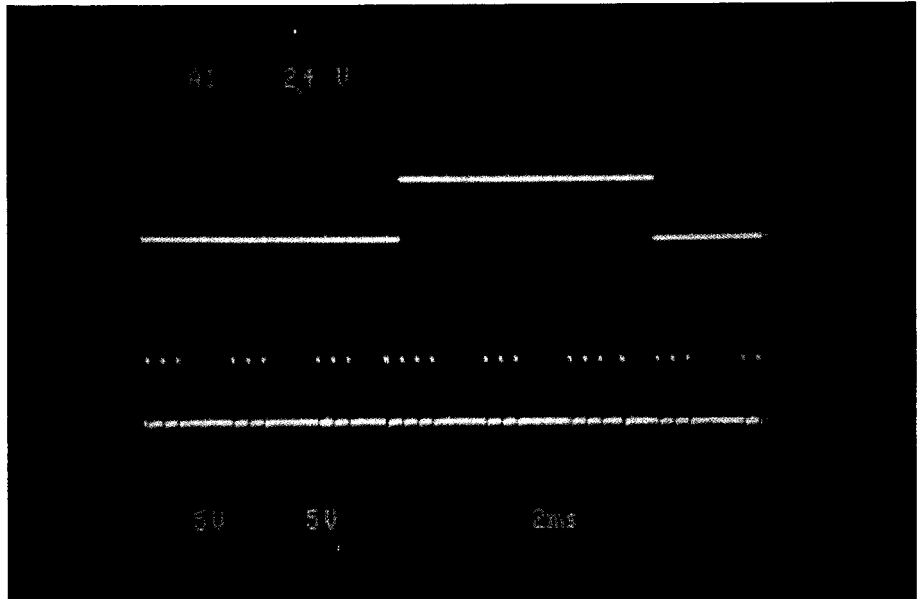


Photo 1—During an X-10 transmission, the primary synchronization comes from the 60-Hz signal shown in be upper trace, which triggers the 20 interrupts from Timer 0 shown on the lower trace.

address spaces in which to store variables: Internal RAM, External RAM, and (for constants only!) in EPROM with the program code. In assembly language, you appreciate full well the differences between the three,

but C language syntax masks the subtleties.

In addition to the program's variables, you need RAM to hold subroutine return addresses, save CPU registers during interrupts, and so

8051 MULTITASKING EXECUTIVE

Develop Real Time Applications FAST!

The DCE51™ Executive provides:

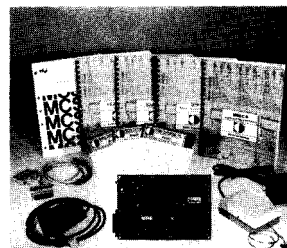
- Preemptive Scheduling
- 4 Priority Levels
- Up to 8 Tasks
- Memory Management
- Message Passing
- Interrupt Management
- Dynamic Task Creation
- Monitor Task Available
- \$300, No Royalties

Iota Systems, Inc.

PH: (702) 831-6302 FAX: (702) 831-4629
POB 8987
Incline Village, Nevada 89450

The \$595 Solution to 8051 System Development

PDK51



The PDK51 is a fully integrated hardware, firmware, and software system designed to help you develop your products quickly and cost effectively.

All you need to use the PDK51 is an IBM-PC/XT/AT or compatible. We supply the rest.

- SIBEC-II microcontroller board (8052AH-BASIC CPU) with 16K RAM
- BASIC interpreter source on disk
- SIBEC-II hardware manual with schematic
- Monitor/debugger ROM plus manual
- Monitor/debugger hard copy source listing
- UL listed 5V power supply
- Programming power supply adjustable from 4V to 26V
- KERMIT communications program
- SXAS1 8051/8052 cross assembler
- BTK52 BASIC programmers toolkit with tutorial
- 160 page BASIC Programming Manual by Systronics
- RS232 cable
- PDK51 tutorial
- Optional: aluminum case—\$45, monitor/debugger source—\$25, battery-backed real-time clock—\$39.
- All components are manufactured to exacting standards and warranted for one year.

PDK51 PLUS includes everything in the PDK51 plus Vers. 3 of our popular BXC518051/8052 BASIC compiler—\$800.

Call Now! 603-469-3232 or FAX 603-469-3530



Binary Technology, Inc.

Main Street • P.O. Box 67 • Meriden, NH 03770



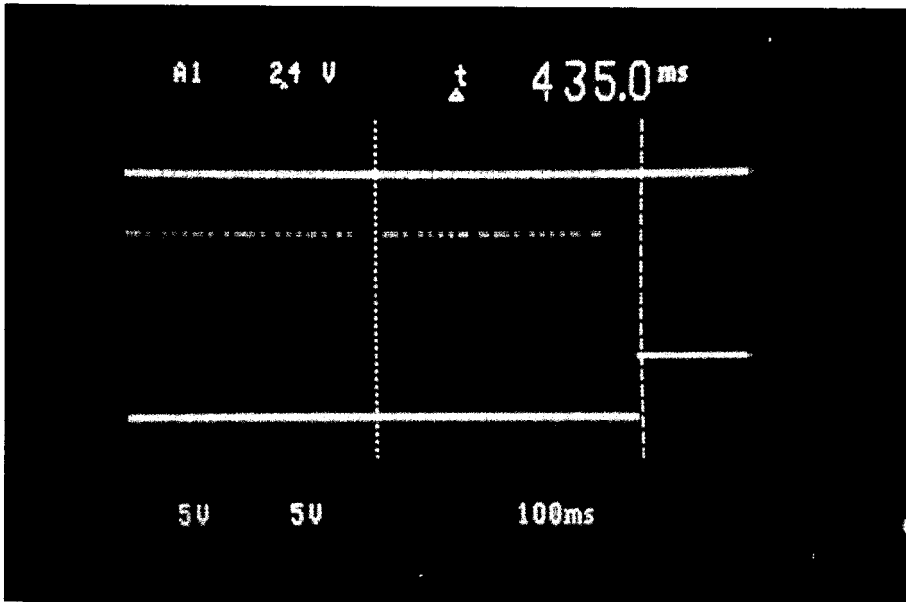


Photo 2-This complete X-10 transmission consists of two pairs of X-10 messages. The first two set the house and device and the second two set the house and function.

forth. C needs space to pass function arguments, evaluate expressions, and allocate heap storage. The memory model you use tells the compiler where to put everything, but given the sheer number of choices it is no

wonder memory models are often confusing.

Micro-C now supports five memory models: Tiny, Small, Compact, Medium, and Large. Although the names may sound familiar to

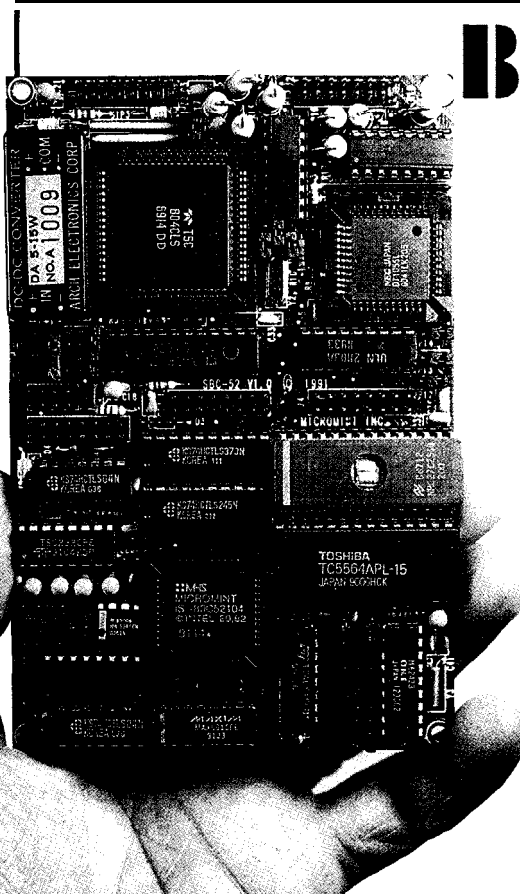
80x86 C programmers, the meanings are not obvious. Worse, 8051 memory models vary by compiler and "Compact" may mean two different things for two different 8051 compilers.

Remember: it's not what you don't know that hurts you, it's what you think you know that ain't so.

Tiny model is the simplest; everything must fit into the 8051's Internal RAM. Given that there are only 128 bytes of RAM (256 bytes in 8052/32 CPUs), Tiny model programs are easily overwhelmed. Nevertheless, the BLinkBOX and COMBO programs in the last two columns were both Tiny programs, even if BLinkBOX needed 256 bytes of Internal RAM.

Small model allows you to put variables into either Internal or External RAM, but keeps local variables (those defined within C functions) on the Internal RAM stack. Small model may be precisely right for your programs if you forego local variables.

Compact model moves the local variables to a simulated stack in



BRUTE-52 **Maximum Power Minimum Space**

Micomint's BRUTE-52 is the ultimate compact controller. One look at the list of features will tell you that this full-featured controller has the power to crush your most demanding applications:

- CMOS 80C52/80C32
- Three 16-bit counter timers
- 11.0592MHz System Clock
- Hardware Watchdog Timer
- Hardware Clock-calendar
- Optoisolated Serial Communications
 - RS-232 or RS-485/300-9600 bps!
 - Optoisolated Serial Printer Port. RS-232/150-9600 bps
 - 5V-only Operation
 - Up to 56 Kbytes RAM and/or EPROM
 - 1 Kbit EEPROM
- 12-bit parallel TTL I/O
- 8-bits buffered high-voltage, high-current outputs
- 8-bits optoisolated non-polarized DC inputs
- 12-bit plus sign analog-to-digital converter
 - 8 channels! 60 Samples/second! 1.2mV resolution!
 - 12-bit digital-to-analog converter
 - 2 channels! 1.2 mV resolution! Selectable ranges!
 - Only 3.5 x 5.3 Inches!
 - Operates at c-70°C
 - Consumes only 100-200 mA (depending on configuration)
 - Use networked or stand-alone
 - DEMO/Diagnostic ROM

BRUTE-52 offers you all these features at only \$459! (\$379/100 quantity OEM) We also have a starter system for \$289. When you add in Micromint's renowned quality, services, and support, you won't find a better value in compact control.

To order BRUTE-52, or for more information, contact:

Micomint, Inc.

4 Park Street • Vernon, CT 06066
Phone (203) 871-6170 • FAX (203) 872-2204

External RAM, so code using those variables becomes larger and slower. However, this model is a must if your functions twiddle large character arrays or use many local variables. The PL-Link uses Compact model because Internal RAM is chock-full of time-critical global variables, leaving precious little room for local variables on the stack.

All three of these models enforce a nonstandard restriction on your code: any initialized variable is treated as a constant stored in EPROM. While this may sound like a terrible disadvantage, your code ought to have an initialization section anyway. In point of fact, putting true constants in EPROM makes a great deal of sense because they cannot be changed by a glitch or program bug.

In addition, these models require combined Code and Data spaces, which is usually accomplished by ORing the CPU's -PSEN and -RD outputs with an '08 gate. This process limits the total address space to 64K bytes total and forces RAM and EPROM to cover different address ranges. You can now see the reasoning behind PL-Link's peculiar addressing with EPROM from 0000 to 7FFF and RAM from 8000 to 9FFF.

Medium model is similar to Small model, but with separate External RAM and Code spaces. The C start-up code copies all the initialized variables from the Code EPROM to the Data RAM before passing control your C program, which can then alter the values as needed. All local variables must still fit on the Internal RAM stack, so Medium model may not be particularly useful.

Large model is the least restrictive of the five because it uses separate Data and Code spaces and puts local variables in External RAM. Of course, the resulting code can be much larger and slower than any other model, but it's the only way to get 64K of Data, 64K of Code, and not run out of Internal RAM.

In all cases, the CPU return address stack is in Internal RAM because that's where the hardware must find it. If you have allegedly portable C code that twiddles the stack

```

SMARTX10 Controller Version 0.6c
Copyright 1992 Circuit Cellar Inc
for single, non-commercial use only
Software Furnace #25, Ed Nisley

Available commands
values are decimal unless noted)
dump internal status variables (most in hex)
dump and clear error flags
flush trans/recv buffers
set reporting modes (bitmapped)
set network mode
display and clear power fail status
dump all devices
dump all devices for housecode
dump single device for housecode
dump all devices in hex
dump refresh interval
set refresh in minutes
clear refresh buffer, set in minutes
set refresh in seconds
dump refresh table entries
perform power-on reset initialization
send command to device at housecode

XXXXXXXXXX
n 02
h 4 2=0n 4=0ff ?=0ff
04 0n
H=N=08 D=14 F=25 Rep=2 (Device 4 On)
H=N=08 D=11 F=08 Rep=2 (Device 4 On)
H=N=08 D=F14 (Device 4)
H=N=08 D=F05 (On)

)Rep: H=N=08 D=F1A (Device 8)
H=N=08 D=07 (Off)
H=N=08 D=07 Rep=2 (Device 8 Off)
H=N=08 D=F05 Rep=2 (Device 1 On)
H=N=08 D=05 (On)

```

Photo 3-A screen shot of a session with the PL-Link shows what the "human" mode of operation looks like

directly, it won't work correctly in all models because the Internal stack grows up and the External stack grows down.

Many 80x86 C compilers implement the storage class keyword `n e a r` to identify variables located within the default segment and far for those variables in other segments. Micro-C slightly abuses the `r e g i s t e r` keyword to designate Internal RAM variables. I prefer the former convention, even though it is not directly applicable to 8051 CPUs, so my `8051base.h` file has a `#d e f i n e` statement to replace all NEAR variables [note the capitalization] with register variables.

Micro-C has a separate start-up file for each memory model. The code defines the starting address and size for the system's RAM and EPROM. I also modified the Compact start-up code to leave room for the interrupt vectors; the changes are detailed in `SMARTX10.C`.

SUMMING UP

SMARTXIO as implemented on the PL-Link hardware can form the basis of a powerful home control system because it relieves the main supervisory controller of the gubby details (and critical timings) required to send and receive X-10 messages. It uses either RS-485 or RS-232 communications, so you should be able to adapt it to a wide range of systems; I'd be

interested in hearing what you come up with.

The hex file required to create a PL-Link EPROM can be downloaded from the Circuit Cellar BBS. PL-Link's source code will not be placed on the BBS, but may be licensed from Circuit Cellar Inc.

The BBS files do, however, include the complete source code for MON X 10, an X-10 monitor program that reports what it hears from a TW523 X-10 interface over the RS-232 serial link. MON X 10 will show you how to create a Micro-C program with in-line assembler and bit manipulations.

Next issue: a people tracker. □

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnectTime" in this issue for downloading and ordering information.

Ed Nisley is a Registered Professional Engineer and a member of the Computer Applications Journal's engineering staff. He specializes in finding innovative solutions to demanding and unusual technical problems.

IRS

- 416 Very Useful
- 417 Moderately Useful
- 418 Not Useful