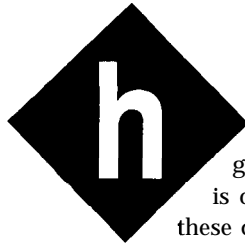


# FEATURE ARTICLE

Ken Davidson

## Programming the Home Control System II

In the last issue of *Circuit Cellar INK*, members of the engineering staff introduced the Circuit Cellar HCS II. Find out now how to program the system to run your home.



Have you taken a good look at what is on the market these days for home automation systems? At the low end is the X-10 CP290 controller. You plug this box into a PC, set on and off times for a bunch of X-10 modules, and disconnect the PC. While inexpensive, it is nothing more than an easy-to-program timer.

Next is the Enerlogic ES-1400. It uses a language similar to what I'm about to describe, but, again, it can only talk to X-IO modules. Its advantage over the CP290 is its capability to receive from the power line and base decisions on what it hears, but it has no direct inputs or outputs. How do you connect a motion detector, light-level sensor, or temperature sensor to it? There are X-10 units that send commands onto the power line based on motion and light level, but given the reliability of X-10 transmissions, do you really want to trust them?

At the high end of the scale are the units that start in the \$3000 range and can only be obtained through "authorized dealers" who must do all the installation and programming. What good are these devices to experimenters who know what they want to do, and just need some specialized hardware to do it!

There are a few low-cost systems available that allow the experimenter to add hardware piecemeal and do the programming, but they are based on the premise that you don't mind leaving your 200-W IBM PC running all the time to do the controlling. I wouldn't want to see that electric bill. And what happens when your kids want to play Tetris?

### HCS II REVISITED

In the last issue of *Circuit Cellar INK*, Steve introduced the new Circuit Cellar Home Control System II and I discussed the brains of the operation: the supervisory controller. The HCS II is based on a number of separate building blocks that may be connected using a network made up of a single twisted-pair wire and is designed for people who know what they want to do and how to do it, but need some low-cost hardware and software to use as tools.

The Supervisory Controller (SC) is responsible for determining how the system operates and when things happen. It has 24 bits of TTL I/O (optionally up to 48 bits of buffered I/O in addition) and eight channels of 8-bit analog-to-digital conversion. The PL-Link module handles all X-10 power line communication, including both transmitting and receiving. The IR-Link [which Steve and Ed discuss in this issue] allows you to issue commands with a hand-held infrared controller. The LCD-Link contains an LCD display that responds to a subset of standard ANSI terminal control sequences and also has four bits of I/O. The DIO-Link adds eight bits of TTL I/O, and its cousin, the ADIO-Link, includes 24 bits of I/O, eight channels of 8-bit analog-to-digital conversion, and four channels of 8-bit digital-to-analog conversion. Up to 32 of these modules may be connected to a single network, although cost will probably limit system size to fewer than that.

A complete HCS II system can be as simple as a lone SC, or as complicated as you want to make it.

### COMMUNICATE THIS

While the HCS II is made up of several autonomous building blocks,

you need some way of interacting with the system as a whole in order to program it. In any properly configured home control system, you should be able to set it and forget it, so we use an IBM PC compatible for doing the "setting," then disconnect it when finished to do the "forgetting."

The program used to communicate with the SC is called **HDST**. When run, **HDST** displays a number of windows containing such information as current time and date, current state of X-10 modules (you decide which housecodes), current state of local inputs and outputs, and what network modules are in use. **HDST** allows you to set a new time and date (it actually reads it from the host PC, so be sure you've set it correctly before running **HDST**) and allows you to load a new program into the SC. I'll cover where that program comes from next.

## THE LANGUAGE OF HOMEOWNERS

The original HCS that Steve presented about seven years ago had a simple menu-driven interface and control scheme. It gave the user several programming options including turning an X-10 module or direct output on and off at specific times, in response to an input, or after a specific time period. While that scheme is easy to use and allows a good degree of control, it falls well short of meeting the needs of a more sophisticated control system that may be applied to both industrial and home environments.

Suppose I have an area with two lights, two motion sensors, and a door sensor. I want the lights to come on if either of the motion sensors is tripped, then go off 15 minutes after no motion is detected. I also want the lights to come on if the door is opened and to stay on as long as the door is open, regardless of motion. When you have a system like the old HCS, where you could only key actions on a single input, such a scenario is impossible to realize without additional circuitry to combine the sensors external to the HCS. Remember this situation for later and you'll see how easy it is to do with the new system.

The HCS II programming language is called XPRESS (expandable Programmable Real-time Event Supervisory System) and is based on what I call an "event equation." The equation consists of an "I F" section, followed by a "THEN" section. If the I F section is true, the **THEN** section is executed. If not, the action is skipped. It's as simple as that. Much of the power comes from being able to combine any number of conditions in the I F portion, and have any number of actions initiated in the **THEN** portion.

A high state and an edge. A falling edge is tested by checking for both a low state and an edge. Inputs are broken into two categories: local inputs and network inputs. Local inputs [tested with the **INPUT** keyword] are those connected directly to the SC and are the fastest to test. Any device that needs quick action (like a motion detector at the top of a flight of stairs) should be connected to one of the local inputs. Network inputs (tested with the **NETBIT** keyword) are those found on the LCD-Link, DIO-Link, and

<b>ADC(n) &lt;op&gt; c</b>	n = analog-to-digital channel <op> = "=", ">", "<", ">=", "<="
<b>Input(n)=ON/OFF/EDGE</b>	c = constant g-255
<b>IRcode(c) &lt;op&gt; n</b>	n = input number O-239
	c = received IR code O-255
	<op> = "=", ">", "<", ">=", "<="
	n = IR-Link module number O-7
<b>Module(m)=ON/OFF</b>	m = module A1-P16
<b>NetBit(n)=ON/OFF/EDGE</b>	n = network I/O bit number O-239
<b>Output(n)=ON/OFF</b>	n = output number C-239
<b>Reset</b>	True after reset
<b>Time &lt;op&gt; hh:mm:dd</b>	hh = hour O-23; mm = min O-59; dd = day SU, MO, TU, WE, TH, FR, SA, DY (daily)
	<op> = "=", ">", "<", ">=", "<="
<b>Timer(n)=ON/OFF</b>	n = timer number O-63
<b>Timer(n) &lt;op&gt; s</b>	n = timer number 031
	<op> = "=", ">", "<", ">=", "<="
	s = O-65535 seconds
<b>Timer(n) &lt;op&gt; s</b>	n = timer number 32-63
	<op> = "=", ">", "<", ">=", "<="
	s = O-65535 minutes
<b>Variable(n)=TRUE/FALSE</b>	n = variable number O-15
<b>Variable(n) &lt;op&gt; c</b>	n = variable number O-15
	<op> = "=", ">", "<", ">=", "<="
	c = constant g-255

Figure 1—XPRESS condition keywords allow the testing of a wide range of sensor inputs and system variables

## CONDITIONAL CONTROL

Figure 1 shows a summary of valid **I F** statement conditions. I've done a lot of work on the system since the last article was written, and have expanded on what was presented there. I'll quickly explain how each of the conditions works. [I know a laundry list of mostly self-explanatory commands can be dry, so I'll try to be brief.] Note as you go through the list that many systems on the market allow you to base actions on time or on inputs, but rarely allow you to combine the two as we do here.

Inputs may be tested for a high state [on], a low state [off], or an edge (either rising or falling). To test for a rising edge, you simply check for both

ADIO-Link modules. Since these modules must be polled over the network, system response to a network input is somewhat slower than to a direct input.

Similarly, the current state of any output may be tested for either on or off. As with inputs, there are local outputs (tested with **OUTPUT**) and network outputs (tested, as with network inputs, with **NETBIT**). The same issue of response time applies to outputs as to inputs.

Analog inputs may also be tested using the **ADC** keyword. A single keyword is used for both local and network ADCs because there are far fewer potential analog inputs than either digital inputs or outputs.

AllLightsOn(h)	h = housecode A-P
AllUnitsOff(h)	h = housecode A-P
DAC(n)=c	n = digital-to-analog channel c = constant 0-255
LCD(n)="string"	n = LCD-Link number 0-7 string = any ASCII text string
Module(m)=ON/OFF	m = module A1-P16
Module(m)=DIM(n)	m = module A1-P16 n = dim level 1-31
Module(m)=BRIGHT(n)	m = module A1-P16 n = bright level 1-31
NetBit(n)=ON/OFF	n = network output bit 0-239
Output(n)=ON/OFF	n = output number 0-239
Refresh=r	r = refresh interval 0-99 minutes (0 = off)
Timer(n)=ON/OFF	n = timer number 0-63
Variable(n)=TRUE/FALSE	n = variable number 0-15

Figure 2—XPRESS action keywords give you plenty of control over your living environment

The current state of any X-10 module may be tested for either on or off using the MODULE keyword. There is no easy way to keep track of a lamp module's dimmed intensity, so we don't even try. A dimmed lamp simply shows up as being on.

The current time of day may be tested in a number of ways using the TIME keyword. The typical =, >, >=, <, and <= operators may be used to compare the time with a constant made up of hour, minute, and day of week. There isn't any way to make comparisons with month or day of month, but we found little use for such comparisons and left them out.

There are many situations where elapsed time must be measured. A common example is turning a light on for a certain number of minutes when motion is detected, then turning it off. Sixty-four timers are defined for the HCS II: 32 that time in seconds and 32 that time in minutes. Each timer may be tested for whether it is on or off, and may be checked for whether it is less than, equal to, or greater than a constant value using the TIMER keyword. Before you think I'm crazy for providing 64 timers, keep in mind who the primary user of this new system is (Steve) and what his control requirements must be.

For those cases where a counter must be maintained or simple true and false states must be stored, there are 16 S-bit variables available.

When multiple IR-Links are connected to the system, knowing which IR-Link received a particular

code or command from the IR remote is sometimes necessary. Using the IR CODE keyword, you can test whether a particular code was received by either a particular IR-Link or a range of IR-Links. For example, writing "IF IRcode( 13 )=0" tests whether code 13 was received by IR-Link number 0. You can use the information to produce different responses based on which room the transmission came from. When the IR-Links are used as part of a people locator system, knowing which IR-Link received the code tells you where a certain person is located. A statement like "IF IRcode( 13 )>=0" can be used to see if the code was received by any IR-Link. Steve's article in this issue covers the IR-Link in more detail.

Finally, setting up default states when the system is reset and continuing from a known condition is often useful. The RESET keyword tests true just once on the first pass through the event equations after a reset, then tests false from then on.

Now that you have all the conditions under your belt, it's time to combine them into useful tests. Any

number of conditions may be combined in a single I F statement using NOT, AND, OR, and parentheses. A NOT preceding any of the above conditions complements the result. ANDs and ORs do pretty much as you'd expect. Some examples of valid conditions might include

```
IF Input(20)=ON AND
    Module(L11)=OFF
or
IF (Input(10)=ON AND
    Output(21)=OFF) OR
    Timer(2)>30
```

## JUST DO IT

On the other side of the coin is the action list following the THEN statement. Figure 2 lists all the valid actions. Any number of actions may be listed, either on separate lines or separated by semicolons. The list is always terminated with an END statement. As before, I'll quickly describe each action, trying not to be too arid.

Any X-10 module may be turned on or off, dimmed, or brightened. Similarly, an "All Lights On" or "All Units Off" command may be sent out to any given housecode.

Outputs may be turned on or off. The same distinction is made between local and network outputs as for the I F tests. Analog outputs may also be set to any 8-bit value.

Timers may be turned on or off. When a timer is turned on, it is cleared to zero and starts counting. If you want to clear a timer that is already on, simply turning it on again will clear the count. Turning a timer off forces all tests on that timer to return a false response.

Variables may be set to any 8-bit value or to true or false. Like many

!	Start of comment
Begin	Start of program
Config <link> = n	<link> = "PL-Link", "IR-Link", "LCD-Link", "DIO-Link", "ADIO-Link"
	n = 0-7
Define <label> = <expr>	<label> = alphanumeric string up to 32 characters <expr> = any valid program statement
Display Modules=<list>	<list> = X-10 module housecodes
GIF/GEND	Start and end of global IF

Figure 3—Additional keywords allow you to configure your system on the fly and make programs more readable.

Listing 1—System actions may be based on any number of input conditions, making complex control scenarios realizable.

```

!
! Example HCS II Program
!
CONFIG PL-Link = 1

DISPLAY Modules = A

DEFINE Lamp1 = Module(A1)
DEFINE Lamp2 = Module(A5)
DEFINE MotionA = Input(2)
DEFINE Motion6 = Input(6)
DEFINE Door-Sensor = Input(10)
DEFINE Basement-Timer = Timer(38)

BEGIN
  IF MotionA=EDGE OR MotionB=EDGE THEN
    Lamp1 = ON: Lamp2 = ON
    Basement-Timer = ON
  END

  IF Door_Sensor=ON THEN
    Lamp1 = ON: Lamp2 = ON
  END

  IF Basement_Timer>=5 AND Door_Sensor=OFF THEN
    Lamp1 = OFF: Lamp2 = OFF
    Basement-Timer = OFF
  END
END

```

Listing 2—As Steve and Ed describe elsewhere in his issue, IR badges may be used with IR-Links to track people throughout a house and base decisions on where people are located.

```

!
! Demonstrate use of IR badges for tracking
!
! Be nice to Steve. but razz Ed if he walks
! in the room without Steve
!
CONFIG PL-Link = 1
CONFIG IR-Link = 3

DISPLAY Modules = A,B,C

DEFINE Steve = IRcode(0) ! Badge number 0
DEFINE Ed = IRcode(2) ! Badge number 2
DEFINE LivingRoom = 1 ! IR-Link number 1

DEFINE Lamp1 = Module(A1)
DEFINE Lamp2 = Module(A5)
DEFINE Stereo = Module(A8)
DEFINE Air-Horn = Module(A16)

BEGIN
  IF Steve=LivingRoom THEN
    Lamp1 = ON; Lamp2 = ON
    Stereo = ON
  END

  IF Ed=LivingRoom AND NOT Steve=LivingRoom THEN
    Air-Horn = ON
    Lamp1 = OFF; Lamp2 = OFF
  END
END

```

# 8051 Family Tools

## In Circuit Emulators

The DryICE Plus is a modular emulator designed so you can get maximum flexibility from your emulator purchase. The base unit contains all the hardware necessary to support pods containing many of the most popular members of the 8051 family of embedded control microprocessors. Buy one base unit, and select one or all of the pods you need to do the job at a much reduced cost. You get the same great functionality found in our popular DryICE 8031 emulator plus real-time Execute-to-Breakpoint, Line-by-Line Assembler, and much more. And the price is (almost) unbelievable! (Yes, it works with the Mac, too!)

**Base Unit (w/RS-232 IF) -- \$299**

**Available Pods: \$149 each**

8031/32, 80C31/32, 80C154, 80C451,  
80C535, 80C552/562, 80C652, 80C51FA,  
8751/52, 87C51/52.

Call about **87C751/752 support**

16K Trace Buffer option: Avail. 2nd Qtr '92

Standard 8031 DryICE -- Still only **\$199**

Enhanced 8031 DryICE -- **\$269**

## 8051 Simulation

The 8051SIM software package speeds the development of 8051 family programs by allowing execution and debug without a target system. The 8051 SIMulator is a screen oriented, menu command driven program doubling as a great learning tool. **\$99.**

## Single Board Computers

**8031SBC - A** fast and inexpensive way to implement an embedded controller. 8031/32 processor, 8+ parallel I/O, up to 2 RS232 serial ports, +5 volt operation. The development board option allows simple debugging of 8031/51 family programs. **\$99ea**

**80C552SBC -10** bit 8 ch. A/D, 2 PWM, 1 RS232 & 2 RS232/422/485 serial ports, sockets for 64k ROM, 64k RAM, +5 volt operation; optional RT Clock w/ battery, 2k EEPROM. Development board version available. **Call for pricing!**

**Call for your custom product needs.  
Free Quote - Quick Response**



# HTE

**HiTech Equipment Corp**  
9400 Activity Road  
San Diego, CA 92126  
[FAX: (619) 530-1458]

**(619) 566-1892**

other programming languages, false is simply zero while true is any nonzero value. Variables may also be incremented or decremented for use as counters. When a variable reaches zero, further decrementing has no effect. Likewise, when one reaches 255, incrementing does nothing. No other math besides incrementing and decrementing is supported.

Text messages may be sent to any LCD-Link in the system. A very useful subset of the ANSI cursor control commands has been implemented on the LCD-Link, and the SC allows you to send any of these commands to the display module.

Finally, the PL-Link's refresh period may be set. If you'll recall Ed's article in the last issue, the PL-Link may be set to send out on or off commands periodically to all modules it's referenced since its last reset. That way, any module accidentally triggered by garbage on the power line will be set back to its proper state. Setting the period equal to zero turns refresh off.

## BELLS AND WHISTLES

Besides the basic language building blocks discussed above, there are additional commands and features that are used for configuration or to make the programmer's job easier. Figure 3 lists these extra keywords.

The SC must know what COMM-Link modules are connected to the network so it doesn't waste its time polling modules that aren't there. The CONFI G keyword is used to define how many of each of the COMM-Link modules are out there. If the program contains no CONFI G statements at all, the SC simply doesn't access the network.

**HDST** displays the current state of any X-10 module, local input, or local output you ask. The **DI SPLAY** keyword is used to define just what housecodes you want displayed.

**DEFINE** is the key to making programs more readable. Instead of using keywords like "Input(2)" and "Module(L3)" when writing a program, **DEF INE** lets you give such keywords descriptive labels like "Kitchen\_Motion" and "Bathroom\_Light."

After all the definitions, a single **BEG IN** is used to denote the start of the program.

One last keyword used in programming is the global **IF**, or **GIF**. "Global" probably isn't quite the right word, but it seems to fit the best. **GIF** allows a single level of nesting of **IF**

statements. The **GIF** keyword is followed by any valid combination of conditions. If it evaluates true, then any number of **IF/THEN** combinations following it up to the **GEN D** are executed. If it evaluates false, the whole block is skipped. **GIF** allows you to selectively execute or skip

**Listing 3**—In a more realistic example, you might control a set of lights based not only on motion but on time-of-day as well.

```

! Practical example of different system behavior
! depending on the time of day
:
CONFIG PL-Link = 1

DISPLAY Modules = C.L

DEFINE Dark = 16:30:DY      ! Sunset
DEFINE Bedtime = 23:00:DY  ! Time for bed
DEFINE Morning = 6:30:DY   ! Starting to get light out

DEFINE BedroomMotion = Input(12) ! Bedroom motion detector
DEFINE BedLight = Module(L4)
DEFINE CeilingLight = Module(L2)
DEFINE BedTimer = Timer(40)

DEFINE HallMotion = Input(10)      ! Hallway motion detector
DEFINE HallLight = Module(L8)
DEFINE HallTimer = Timer(41)

BEGIN
  GIF Time>=Dark AND Time<BedTime THEN
    IF BedroomMotion=EDGE THEN
      BedLight = ON; CeilingLight = ON
      BedTimer = ON
    END

    IF BedTimer>20 THEN
      BedLight = OFF; CeilingLight = OFF
      BedTimer = OFF
    END

    IF HallMotion=EDGE THEN
      HallLight = ON
      HallTimer = ON
    END

    IF HallTimer>10 THEN
      HallLight = OFF; HallTimer = OFF
    END
  GEND
!-----
  GIF Time<Morning THEN
    IF HallMotion=EDGE AND HallLight=OFF THEN
      HallLight = DIM(12)
    END

    IF HallMotion=EDGE THEN
      HallTimer = ON
    END

    IF HallTimer>3 THEN
      HallLight = OFF; HallTimer = OFF
    END
  GEND

```

whole portions of a program depending on some condition. Check out some of the example programs I present later if you're still not clear about **G I F**.

Finally, comments are preceded by an exclamation mark (!). All text following the "!" up to the end of the line is skipped.

## COMPILE TIME

The program actually run by the SC is made up of binary representations of the keywords I described above. I presented the basic format of that binary in the last issue. In order to translate from the English-like program developed by the user to the binary used by the SC, a compiler is necessary. Simply named **C O M P I L E**, the compiler runs on an IBM PC compatible and takes as input straight ASCII text entered with any text editor, does a syntax check, and generates a file called **EVENTS**. **BIN** containing the raw binary code used by the SC.

When the user running **HOST** presses the "L" key, **HOST** looks for **EVENTS**. **BIN**, loads it into memory, and sends it to the SC. The SC immediately starts running the new program and the **HOST** screen reconfigures itself assuming the transfer went all right..

## PROGRAMMING CONSIDERATIONS

A key idea to keep in mind is that a program is made up of a series of event equations. The SC continually runs through the list over and over again. Any action that takes place near the top of the program could affect the evaluation of equations later in the list, so a certain precedence can be achieved by where in the list a particular equation falls.

Input levels and edges stay static throughout a pass through the list, changing only after a pass is complete. An input edge is always cleared at the end of a pass, but may be tested multiple times in a single pass.

One concept I'm sure is going to bite some people is once an equation evaluates true and its companion action has been carried out, that action won't be executed again until the equation evaluates false at least once,

then true again. For example, suppose I have the statement:

```
IF Input(3)=ON THEN
  Module(J2) = ON
END
```

**IF I** were to send out an on command to module J2 every time the condition Input ( 3 )=ON evaluated true, I'd have a very busy power line until input 3 went off. Executing the on command just once, then skipping it until input 3 goes off, then on again is one way to get around the problem.

Another way I get around the above problem is to be a little smart about X- 10 commands. Transmitting an on command to a module if the module is already on is kind of silly. Therefore, I check the status table first, and if the module is on, I don't send another command. This check doesn't work for bright or dim commands, because the light will likely be already on when you want to dim or brighten it to a new level, so you have to make your program a bit smarter

when using those commands to avoid repeat transmissions.

## LEARNING BY EXAMPLE

I find the easiest way to learn something new is by practical application of the concepts. I've listed all the keywords and rules involved in writing a program, but a few simple examples should help clear up some of the haze in the air.

Let me go back to the scenario I cited earlier. I have two motion detectors, two lights, and a door sensor. Listing 1 shows one way a program can be written to solve the control problem. The first equation looks for an edge on either motion detector and turns on the lights if it finds one. It also starts a timer. The next equation checks the door sensor and, if the lights haven't already been turned on by motion, it turns them on. The last statement checks the timer to find out if five minutes have elapsed. If so, and the door is closed, the lights are turned off and the timer is stopped. If the door is open, the lights stay on

# A Speed Development MULTITASKING Simplify Coding CORE



The **TAPIR** Multitasking Package offers a complete kernel for developing and debugging multitasking code. Written for speed, this 100% assembly language program delivers:

- True pre-emptive multitasking
- A debug module to examine register status of all tasks during runtime
- The ability to run with or without DOS
- No Royalties

The **TAPIR** multitasking package is available for the NEC V and Intel 80x86 Series microprocessors for \$595. (The complete source code is available for an additional \$295)

For more information or to place an order, contact:

**TAPIR**  
PO Box 1553  
Seguin, TX 78155  
(512) 372-2587

until it is closed. Also note that if motion is detected again before the timer times out, the timer is restarted from zero. The result is the lights stay on for five minutes from the last detected motion, and not the first.

Since we're dealing with the IR-Link elsewhere in the issue, how about an example of performing one set of tasks based on the presence of a particular person, and another if that person is absent? Listing 2 shows such code. When Steve walks into the living room wearing his IR badge, the system will be nice and turn on the lights and stereo. If Ed happens to walk into the same room, but Steve isn't there, we'll razz him with an air horn and turn the lights off. If Steve is with him, though, we'll be nice again because we don't want to subject Steve to all that noise.

For something more practical, suppose you have a motion detector in the bedroom and you want the lights to come on after dark. You really don't want the lights coming on during the night every time you turn over in bed, and you also don't want them to come on when it's light out. Similarly, you have a motion detector in the hall just outside the bedroom and you want the hall light to come on when it's dark. However, if you happen to take a trip to the bathroom during the night, you'd like the hall light to come on and dim down to a tolerable level.

Listing 3 shows how such a setup might be done. It also points out a number of programming "gotchas" that will likely bite novice HCS II programmers. The first global if is only executed between the time it gets dark out and when you go to bed. The statements are very similar to those in the other examples, with lights going on in response to motion, and off in response to a timeout.

In the second global if, take a look at the condition. The HCS II time of day is based on a midnight-to-midnight cycle, so the condition responds as if it were written

```
IF Time>=0:00:00 AND
   Time<Morning THEN
```

When counting from 0 to 23, it's not necessary to test if the value is greater

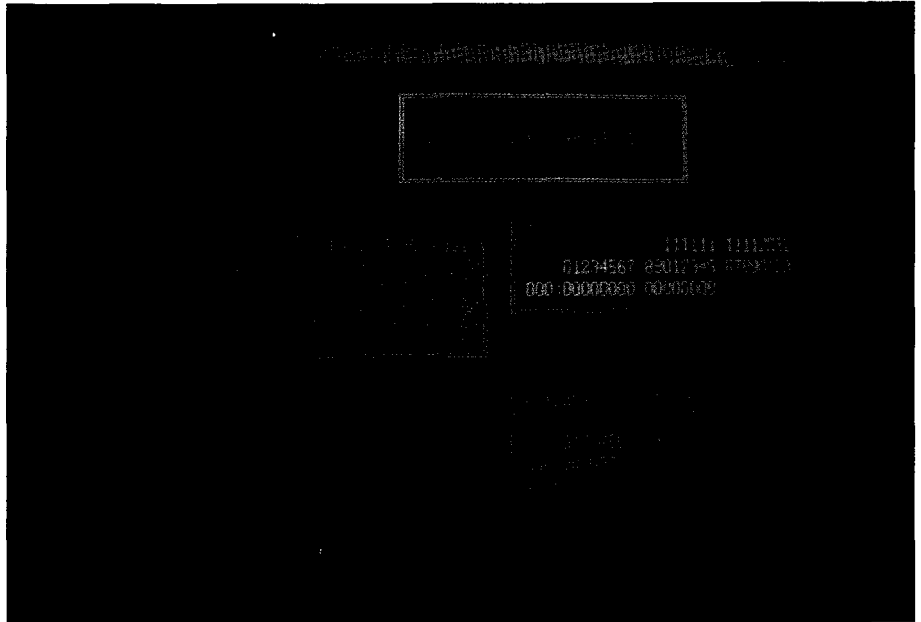


Photo 1-A typical status screen displayed by HOST shows two X-10 housecodes, 16 inputs and 8 outputs on the SC, network activity, and current time and date. The size of the windows and the amount of information displayed change dynamically depending on the equipment you have in your system.

than or equal to 0; it always will be. The same applies here. Obviously, the hall lights won't come on between bedtime and midnight, but how often do you get up within the first hour after you've gone to bed? If it's a problem for you, just add a bit more intelligence to the program to fill the gap.

Next, notice the extra condition in the first IF statement and the extra IF statement following it. Remember I said that if an X-10 module is already on, I don't send another on command. That is why the previous statements can be written without checks to see if the light is on; it's done automatically. If motion is detected after the light is turned on and before the timer times out, the timer will be cleared but the on command is skipped. When you're dealing with dim (and bright) commands, though, you have to be smarter. You may want to intentionally dim a light that is already on, so I can't block the dim command like I can the on command. You have to add an extra condition to check for whether the light is off, and send the dim command only if it is. That way the command is only executed once and you don't end up with a light dimmed all the way to black. The second IF statement is necessary to retrigger the timer should more

motion be detected before the timer times out.

When the conditions in neither GIF are met, which is true any time it's light out, then nothing happens in response to any of the motion detectors, which is just what we want.

## CONTROLLING THE FUTURE

That about does it for version 1.0 of the HCS II. As we get more feedback from those of you living with the system, we'll be refining it to more closely meet your needs. Let us know what you think. □

*Ken Davidson is the managing editor and a member of the Computer Applications Journal's engineering staff. He holds a B.S. in computer engineering and an M.S. in computer science from Rensselaer Polytechnic Institute.*

## SOURCE

Please see page 31 for more information about the availability of HCS II components.

## IRS

- 410 Very Useful
- 411 Moderately Useful
- 412 Not Useful