

The X-10 IR543 Infrared Gateway/Controller

Control your *lights* with your *trainable IR* remote

by Ken Davidson

How does the television commercial go? "My wife left me, so I bought an expensive television set, and it came with a remote. Then my dog died, so I bought a VCR, and it came with a remote. Then I was transferred to Alaska, and it came with two remotes.."

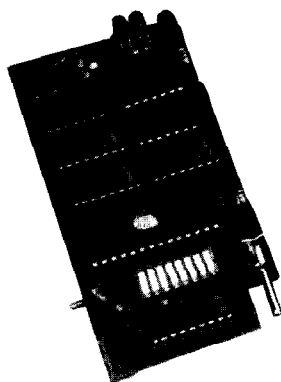
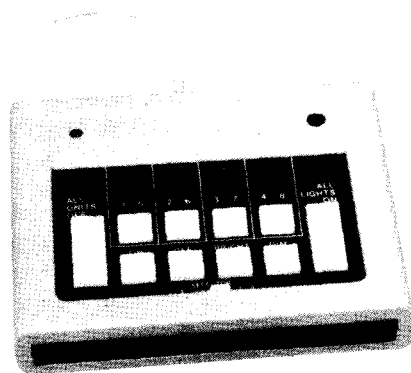
Well, you get the picture. It seems every device even closely related to consumer electronics comes with a hand-held infrared remote control these days. **And once** you start putting together your entertainment center, you end up with a pile of incompatible, but necessary, remotes.

To combat the problem, a host of trainable controllers have been introduced to the market. Indeed, Steve did a Circuit Cellar article for the March '87 issue of BYTE which described his design of just such a trainable remote.

So now we can control the TV, VCR, cable box, CD player, and so on with a single IR remote. What about the lights or other appliances we might have plugged into X-10 modules? How can we remotely control those without using the hard-wired consoles?

When BSR first introduced their System X-10, there was a hand-held ultrasonic remote available for it. Back in the late seventies, those TVs that had remotes usually used ultrasonic, so the choice was appropriate.

When System X-10 was taken over by X-10 (USA) Inc., the ultrasonic remote/base pair was discontinued and replaced by an RF system, the RC5000. The RT504 hand-held remote can send out signals to the RR501 base unit to turn modules on or off and can dim or brighten lights. While you have all the advantages of RF (you can use the remote from virtually anywhere within your house), you still can't point the hand-held RF unit at your trainable IR remote and expect the IR unit to learn the signals. Add one more box to the remote pile.



A Solution Appears

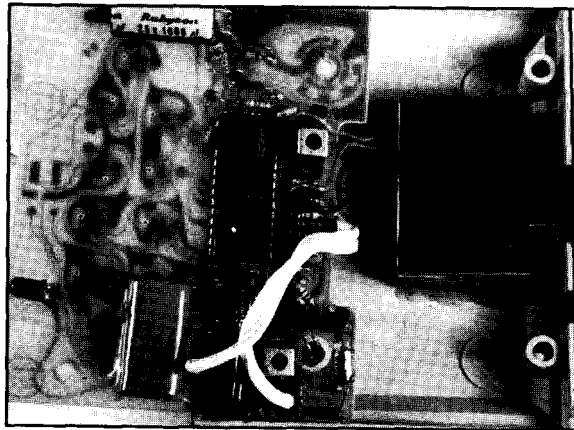
The latest addition to X-10's product line goes a long way toward rectifying the situation. The IR543 is an infrared-to-X-10 gateway/controller which will receive X-10 commands from a hand-held IR remote control, tack on house code information, and broadcast them over the power line.

The schematic for the IR543 is shown in Figure 1. The box contains the usual requirements for an X-10 transmitter: power supply, zero-crossing detector, free-running 120-kHz

oscillator, and output drive circuitry. (For details about how the X-10 system works, see "Power-Line-Based Computer Control" in issue #3 of *CIRCUIT CELLAR INK*.)

The key element in the IR543, though, is the custom 78542C controller chip. The 78542C takes care of all the unit's operating details such as scanning a keyboard, translating a keycode into an X-10 bit sequence, tacking on housecode information, and sending the code onto the power line by watching the zero-crossing input and gating the 120-kHz signal onto the power line using the correct timing. In addition, the chip has a surreptitious "serial" pin for accepting serial input.

This serial input pin has been quietly overlooked for years. It turns out that the 78542C was used in the original BSR ultrasonic unit. The ultrasonic front end's output was sent to the chip's serial input so the signal could be rebroadcast over the power line. The end of production of the ul-



The inside of a prototype IR543 clearly shows the metal-encased IR receiver section.

trasonic control unit didn't mean the end of the 78542C, however. The chip has been used for years as the basis for the SC503 maxi controller and, until recently, the MC260 mini controller. (The MC260 has eight small push buttons and two large buttons. The newer MC460 mini controller has six rocker switches and uses a different controller chip.) The serial input pin has simply been tied to ground in these controllers for all these years.

It logically follows that the overlooked capability of the 78542C would one day be tapped again. With the addition of an IR front end in place of the old ultrasonic front end, we can extend the system's functionality with a remote control. The IR front end of the IR543, which is encased in a metal can for added noise immunity, is responsible for receiving the IR from the hand-held remote and translating it into a series of bits for the 78542C's serial input.

Hand-held Dilemma

So now we have a box that will receive IR commands and retransmit them onto the power line. How do the IR commands get to the box, though? Unlike the RF remote/base pair, X-10 doesn't make a low-cost, hand-held transmitter that is dedicated to the IR543. The IR543 was originally designed for a company who built the commands into their own "universal"

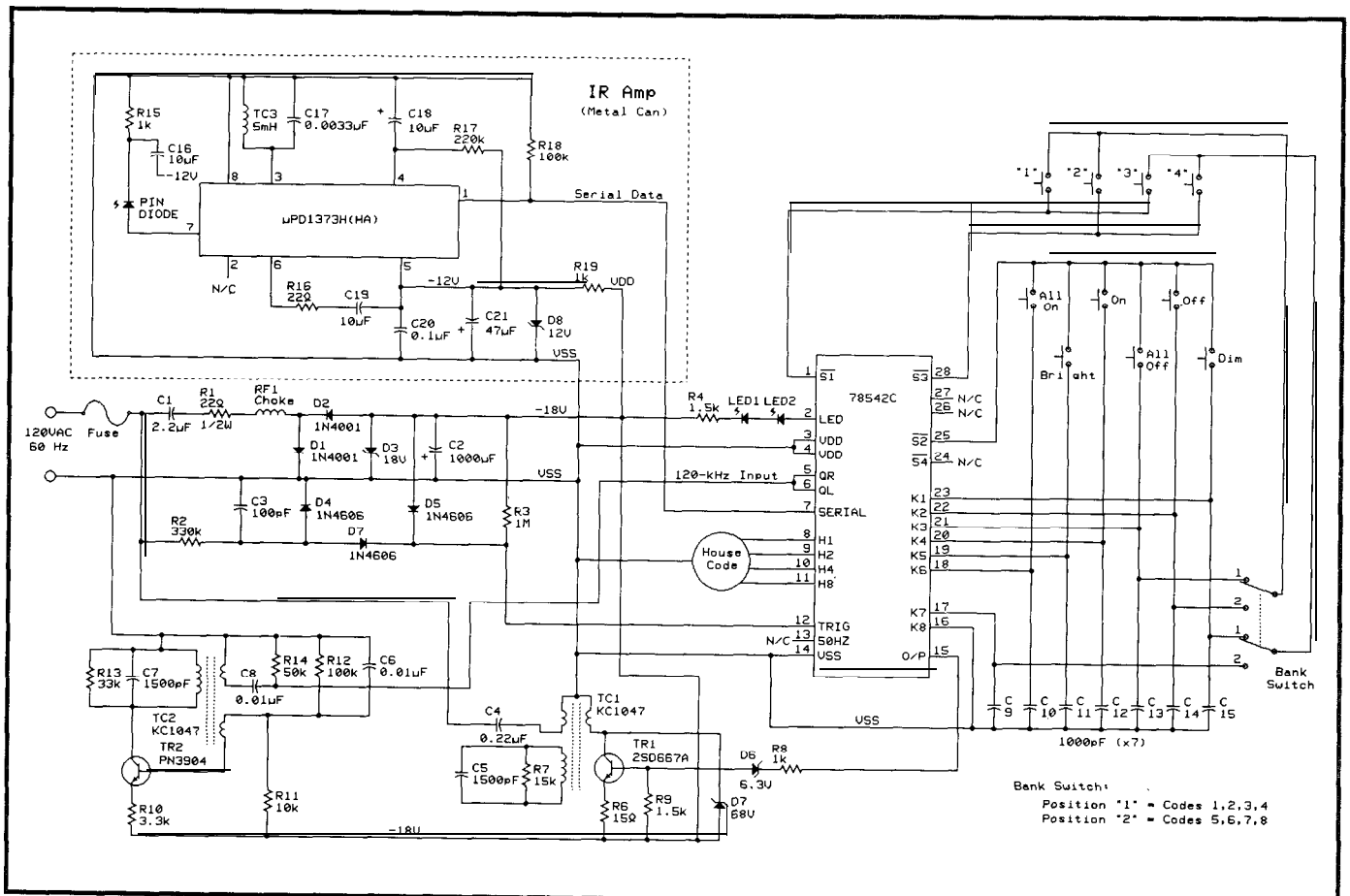


Figure 1 —The schematic for the IR543 is virtually identical to that of the MC260 mini controller.

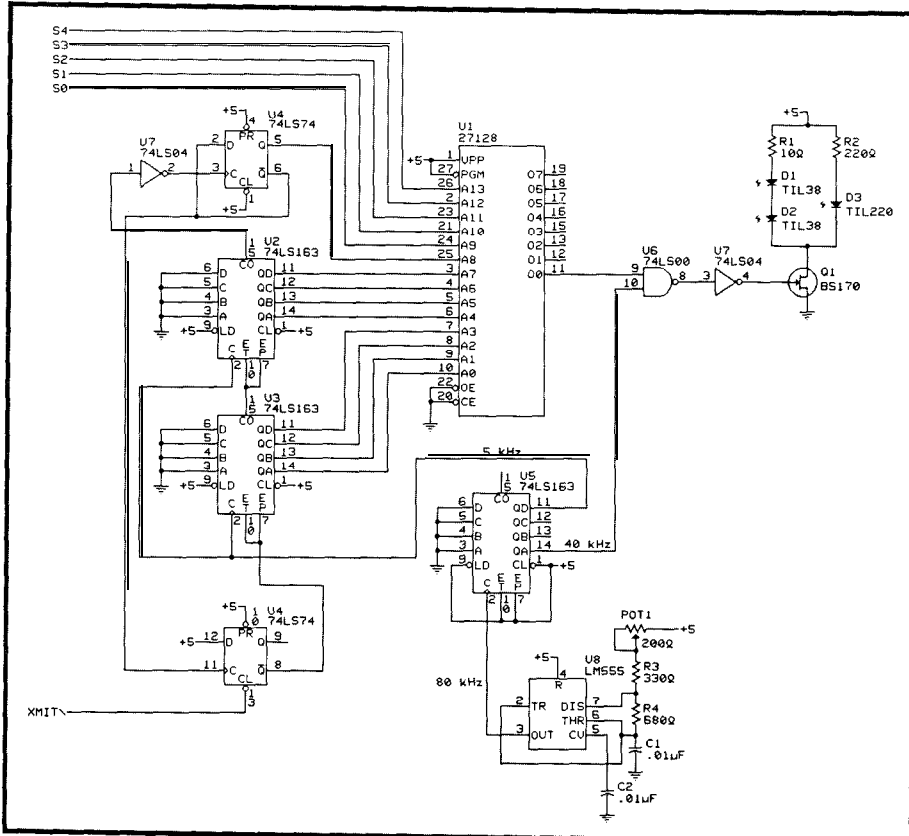


Figure 2—The basic circuit used for generating X-10 IR codes consists mostly of an EPROM, a few counters, and a master clock.

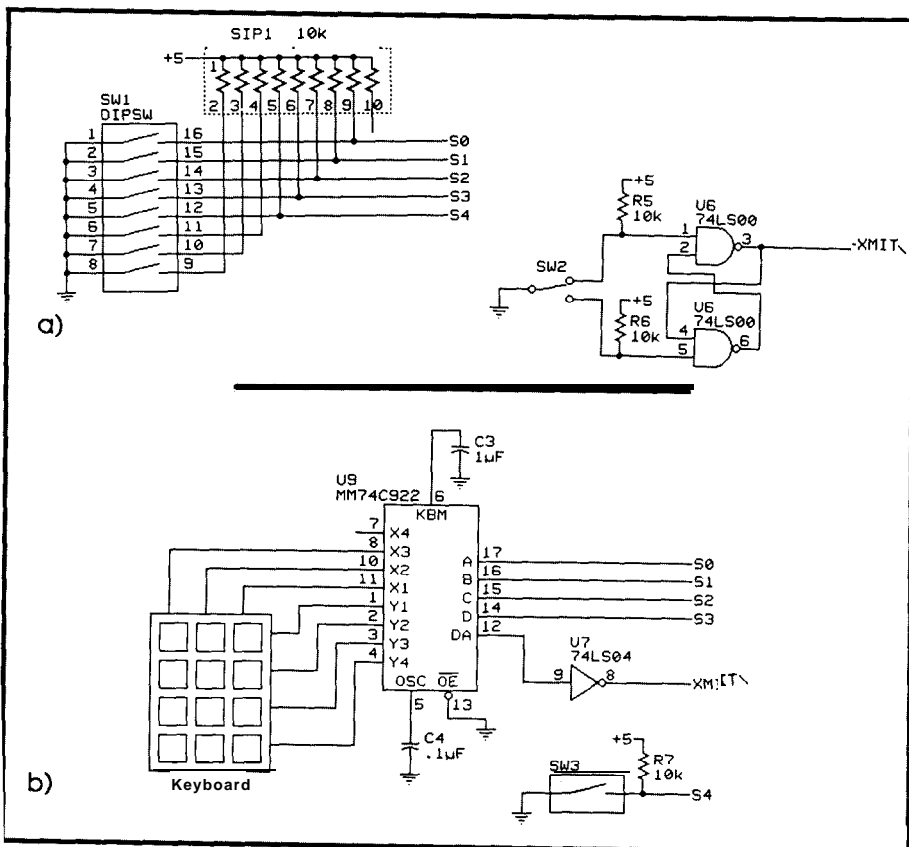


Figure 3—a) A set of DIP switches plus a 'transmit' switch are all that are necessary for a one-time-use circuit. b) A keyboard may be added for everyday use of the circuit.

IR remote (the URC-5000 "ONE FOR ALL" from MTC/USA). If you're already in the market for an all-in-one remote, buying the URC-5000 for use with the IR543 may be a good alternative. However, if you already own a trainable remote, or don't want to spend the \$100 for one, there must be a better way to generate the proper IR signals for use with the IR543.

Even though X-10 tells me they may make such a low-cost, dedicated remote in the future, there is a definite gap that needs filling. Since X-10 was able to give me complete specs on the IR codes being used, I decided to build my own IR command generator. And since it would only be used to train my trainable remote anyway, it could be quick, and dirty.

The Better Alternative

It is possible to interface a simple IR LED driver circuit to an output bit on a personal computer, then drive the LED with some software to train the trainable remote, but it was more fun to build a dedicated, hardware-only board to generate the IR command strings. Using a hardware-only solution also opened the door to the addition of a real keyboard so those who don't have a trainable remote and don't want to spend the money on one can build a usable dedicated remote for use with the IR543. Figure 2 shows the basic circuit for generating the IR commands. Figure 3a has the additional circuits necessary to build a one-time-use-only circuit for use with trainable remotes, and Figure 3b shows what is necessary to add a keyboard to the circuit.

Figure 4 shows a sample command string expected by the IR543. "One" bits are represented by a 4-ms burst of 40-kHz signal, followed by 4 ms of silence. "Zero" bits are represented by a 1.2-ms burst of 40 kHz, followed by 6.8 ms of silence. In both cases, one bit time is 8 ms long.

The command string consists of a "one" bit, followed by five command bits, the complements of the five command bits, then an "end code," which is a 12-ms burst of 40 kHz followed by 4 ms of silence.

As far as timings go, a complete command string consists of an 8-ms start bit, ten 8-ms data bits, and a 16-ms end code, for a total length of 104 ms. Suppose we divide this string into discrete time segments, where each segment is described with a "1" to represent the presence of 40 kHz, or a "0" to represent its absence. We need to generate envelopes for the 40-kHz bursts of 1.2, 4, 6.8, 12, and 16 ms, so the largest subdivision we can use is 0.2 ms. The complete command string can then be described as a series of 520 bits, each bit representing a 200-microsecond slice of time.

Theoretically, we could assemble a table of all 22 commands (16 module numbers and six true commands) with the bits packed into 8-bit bytes that would end up being 1430 bytes large. If this command table is going into EPROM, though, why throw away most of the EPROM and make the external circuitry more complex by trying to pack things together? If we encode just one bit per byte, we still only take up 11K of a 16K 27128 EPROM.

For the sake of simplicity, let's round the 520 bits down to 512 bits. The missing eight bits represent 1.6 ms, so the delay between bursts will only be 2.4 ms instead of the 4 ms called for in the spec, however the spec also says the minimum delay between bursts can be as short as 2.5 ms, so we're pretty close. It turns out 2.4 ms works just fine. Now all we have to do is clock 512 bits out of the EPROM, one bit every 200 μ s.

Two 74LS163 synchronous counters (U2 and U3) plus half of a 74LS74 (U4a) provide us with the 9-bit address necessary to access the 512 bits. The outputs of synchronous counters change simultaneously, so we don't have to worry about glitches coming from the EPROM data line caused by rippling address lines. The output of the EPROM is fed into a 74LS00 (U6c) which gates the 40-kHz signal coming from the master oscillator. The output of the LS00 is inverted and drives a high-current FET (Q1). This FET drives the IR LEDs (D1 and D2) plus a visible LED (D3) so we know something is being sent.

The master clock for the counters is derived from an 80-kHz oscillator made up of an LM555 (U8) and a few discretes. The 80 kHz is divided down by another synchronous counter (U5) to generate the 40-kHz IR signal (+2) and a 5-kHz signal (+16) whose period is 200 μ s. Almost any counter can be used (synchronous or asynchronous), but why not keep the parts list simple and use the same kind all around?

To start the transmission of the command string, we use a push button which is debounced with a pair of NAND gates (U6a and b). When the button is pushed, U4b is cleared, the Q output goes high and enables the low-order counter, and code transmission begins. When U3 overflows (the count has reached some multiple of 16), it enables U2 for one clock cycle so the high-order counter increments once. When U2 overflows (the count has reached 256), the overflow output clocks U4a, causing the Q output to go high so the second 256 bits of the command can be accessed. Also note that the Q output of U4a goes low at the same time.

When counters U2 and U3 reach 256 for the second time, the overflow output of U2 clocks U4a again, causing the Q output to go back to zero and Q to go to one. If the push button is still being held down, U4b will continue to be held clear and a second transmission of the code begins. Transmissions continue until the button is released. Once the button is released, the rising edge of U4a Q generates a clock signal for U4b, which clocks a one bit into the flip-flop. This action causes the Q output to go low, disables the counters, and stops the transmission process. To restart things, the button must be pushed again.

To select which command is sent, a set of DIP switches is connected to the five high-order address lines of the EPROM. Just set a code from 0 to 21 on the switches, press the "send" button, and the selected code is sent out. Obviously, setting DIP switches and pressing a button would be pretty inconvenient for everyday use, but remember, all we want to do is train the trainable remote. After that we can throw the circuit away.

Heathkit

A leader in quality electronics for the technically sophisticated customer.

When you need kit or assembled electronic products for work, home or hobby, you can be sure Heathkit products are designed to perform reliably and effectively...year after year.

See what we have to offer. To get your **FREE Heathkit Catalog**, fill out and mail the coupon below or call toll-free today!

1-800-44-HEATH
(1-800-444-3264)

YES! Please send me a **FREE** copy of the Heathkit Catalog.

Send To: Heath Company, Dept. 026-774
Benton Harbor, Michigan 49022

Name _____

Address _____

Apt. _____

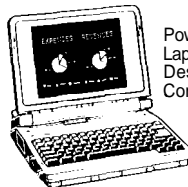
City _____

State _____

Zip _____

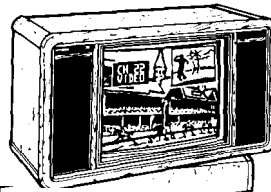
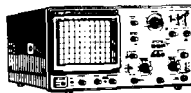
A subsidiary of Zenith Electronics Corporation

CL-801

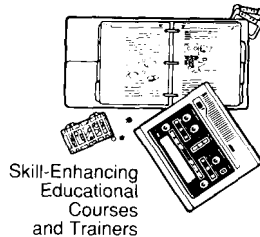


Powerful Kit Laptop and Desktop Computers

Precision Test Instruments



Dynamic Home Entertainment Products



Skill-Enhancing Educational Courses and Trainers

Circle No. 127 on Reader service Card

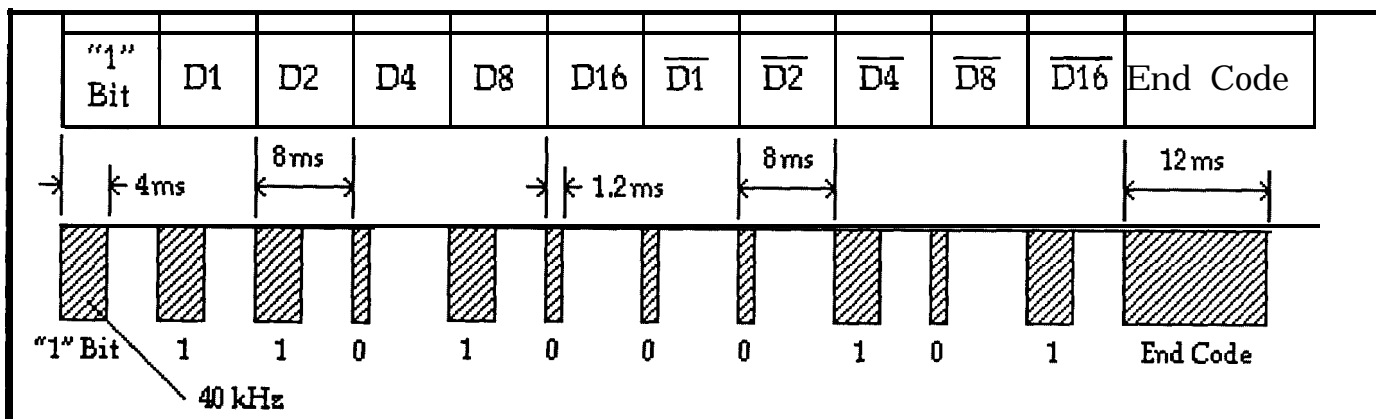


Figure 4—IR command strings consist of a leading start bit, the command code, the complement of the command code, and an end code.

For those who do not want to use the circuit every day, a keyboard circuit can be added so that any standard 16- or 20-key matrix keyboard can be used to select which code to send and to start code transmission. The National Semiconductor MM74C922 is a 16-key encoder chip that scans a 4x4 array of SPST push buttons and outputs a 4-bit scan code and a "data available" strobe.

Ideally, since there is a total of 22 commands, a 22-key keyboard would be used with an encoder that presents a 5-bit code. Since the best we can do with the '922 is 16 keys, I decided to use a 12-key keyboard with an extra switch to select between modules 1-8 and modules 9-16.

Figure 3b shows the keyboard interface circuit. The '922 contains all the necessary pull-up resistors and debounce logic, so implementing it in a circuit requires just two capacitors: one for the master scanning oscillator and one for the debounce circuitry.

When a key is pressed, the '922 latches the corresponding keycode into its data output latches and the "data available" (DA) output high. DA is held high for as long as the button is held down.

Looking back at the original transmitter circuit, all that must be done to use the '922 in the circuit is to connect the data outputs to the upper EPROM address lines and an inverted version of DA to the CLEAR input of U4b. When a button is pressed, the correct X-10 code is selected by the data lines and it will continue to be sent as long as the button is held down.

Command	D1	D2	D4	D8	D16
1	0	1	1	0	0
2	1	1	1	0	0
3	0	0	1	0	0
4	1	0	1	0	0
5	0	0	0	1	0
6	1	0	0	1	0
7	0	1	0	1	0
8	1	1	0	1	0
9	0	1	1	1	0
10	1	1	1	1	0
11	0	0	1	1	0
12	1	0	1	1	0
13	0	0	0	0	0
14	1	0	0	0	0
15	0	1	0	0	0
16	1	1	0	0	0
All Units Off	0	0	0	0	1
All Lights On	0	0	0	1	1
on	0	0	1	0	1
off	0	0	1	1	1
Dim	0	1	0	0	1
Bright	0	1	0	1	1

Figure 5—The command codes sent via IR are identical to those sent over the power line.

The EPROM

Now that we have the circuit, we need to put something in the EPROM. I wrote a quick program in Turbo Pascal to generate an Intel hex file that can be sent directly to an EPROM programmer. The basic 5-bit codes for each command are placed in an array. The program then generates a legal command from the base bit sequence, inserting the proper start code, complemented bits, and end code. Then it breaks the command sequence up into its 512 component bits, and finally writes them out to a file. The extra 5K

at the end of the EPROM is filled with zeros so we don't flood the room with IR should the switches be set to an illegal command code. [Editor's Note: Software for this article is available for downloading from the Circuit Cellar BBS and on Circuit Cellar INK Software On Disk #9. For information on downloading and disk orders, see page 78.1

The beauty of using an EPROM in this application is that the keyboard or DIP switches may be mapped to any corresponding X-10 codes simply by remapping the EPROM.

Parts cost for the quick-and-dirty version of the circuit should be under \$20 (add about \$10 for the keyboard version) and the circuit can probably be built in an evening or two. Construction techniques are very noncritical; the highest frequency we're working with here is 80 kHz, so noise isn't much of an issue (we'll save the 16-MHz 68020s for another day). ❖

Special thanks to Dave Rye for his contributions to this article.

Diagrams and schematics pertaining to the IR543 are reprinted by permission of X-10 (USA) Inc.

Ken Davidson is the managing editor and a member of the CIRCUIT CELLAR INK engineering staff. He holds a B.S. in computer engineering and an M.S. in computer science from Rensselaer Polytechnic Institute.

IRS

- 204 Very Useful
- 205 Moderately Useful
- 206 Not Useful