CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

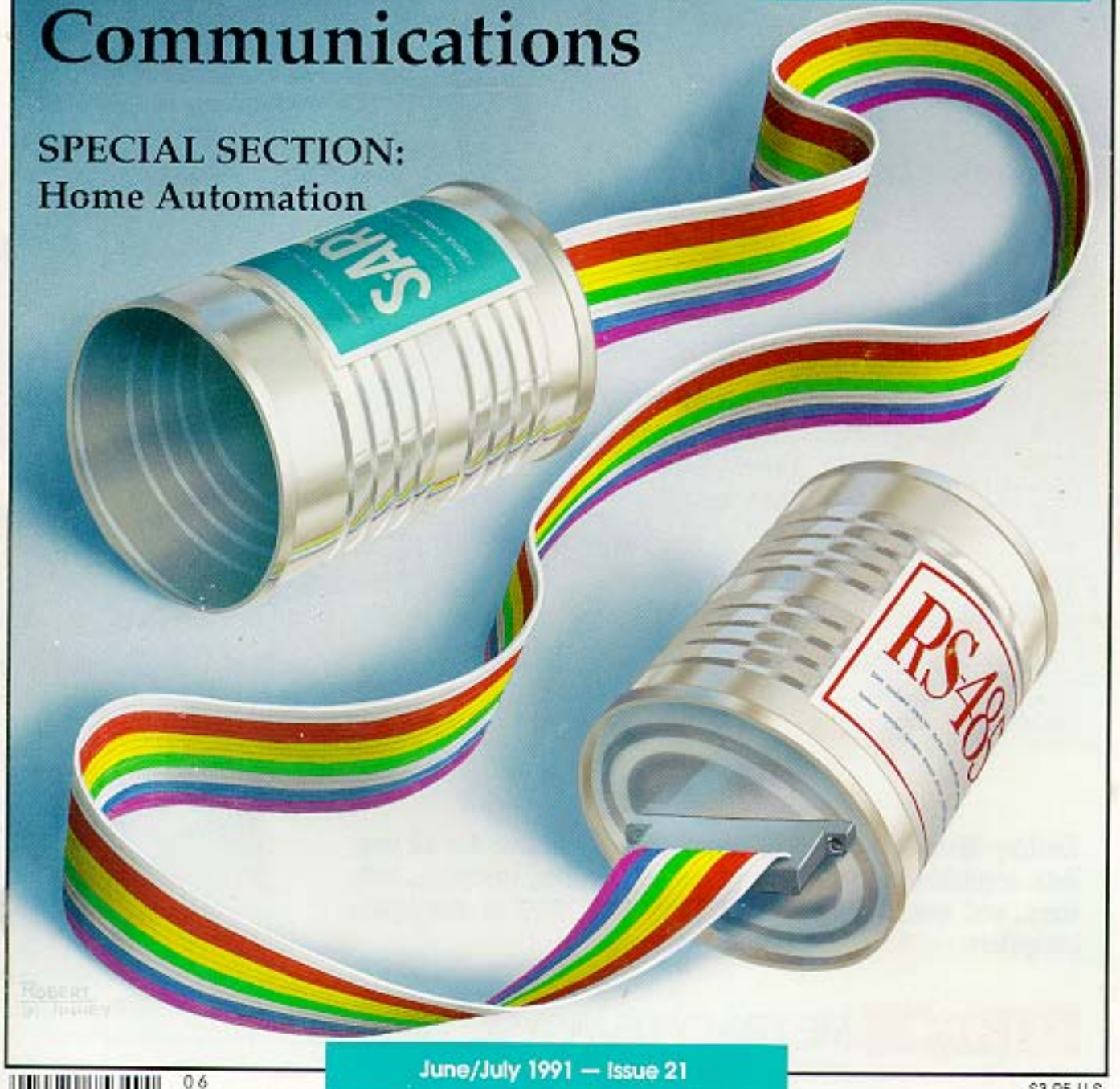
Networking Microcontrollers

Ciarcia's Talking House

Writing TSR Software

Communications

SPECIAL SECTION:
Home Automation



June/July 1991 — Issue 21

$3.95 U.S.
$4.95 Canada

0 74470 75349 0 06

EDITOR'S INK

Home, But Not Alone

Curtis Franklin, *Jr.*

If you listen closely, you can almost hear it. Around most of us, systems, devices, and people are communicating at an unprecedented rate. Wires and fibers are filled to bandwidth limit and the air is saturated with infrared and radio-frequency transmissions. In most of the places we spend our time—home, office, factory, car, or store-communications have become so entrenched in the routine of daily life that they fade into the background. As the companies which provide us with information and data carriers compete to offer more complete and more complex services, the odds are overwhelmingly in favor of communications playing an increasing role in our professional and personal lives.

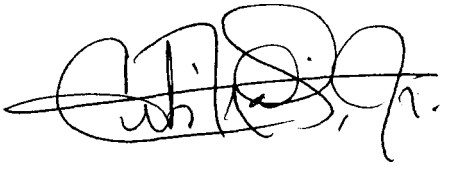
One of the technologies looming on the horizon is Integrated Services Digital Network, or ISDN. To be sure, ISDN has been looming on that **horizon** for nearly ten years now, but pieces arc falling into place for its eventual availability in most offices and homes. If roadways are appropriate analogies for data channels, residential ISDN is rather like have an eight-lane expressway running out of your garage. There will be sufficient bandwidth for applications such as video conferencing to move out of high-priced satellite centers and into the living room. (Whether residential customers will want video telephones and the like is a totally separate and utterly unresolved issue.) Entertainment, education, and centralized security are among the application areasthatareunderexplorationfortheday whenbandwidthwill not be the limiting factor in residential communications. Telephones revolutionized the way individuals lead their private lives: It is not a great stretch to imagine that the next ten to fifteen years will seecommunication development that will haveat least as much impact as the original telephone.

When you start looking at the implications of increasingly capable residential communications, it's hard to avoid the topic of home automation. After competing with one another for small niches for years, the manufacturers of home automation products learned a lesson from the small computer and home entertainment industries: Market size and standards are related. Customers are so repulsed by the notion of buying a proprietary system which might be orphaned that they will choose to buy nothing rather than take that risk. A good standard removes the risk, and with it a high barrier to customer outlay. EIA's CEBus standard has sufficient weight and support to throw home automation into a period of tremendous growth. According to a study by Rose Associates, home automation systems (of some sort) are now in two percent of U.S. homes.They predict that the percentage will rise to 15% by 2000, and hit 50% in twenty years. As they also estimate the current home automation market at nearly 1.5 billion dollars a year, you can see the potential for market growth in sheer dollar terms. While I won't deny the importance of self-contained stand-alone machine intelligence, the potential available to fully communicating dwellings begins to stagger the imagination.

One of the important social ramifications of the trends I've been discussing is that they make where you are much less important than how you're connected. Obviously, there are hundreds of jobs that require face-to-face human interaction, but many industries have already found that workers can be as productive (or more productive) in their own environment as they are in the office. In areas such as Southern California, air quality regulations are adding a governmental component to the trend toward keeping as **many** people as possible away from the office. Social trends having nothing to do with computers may also act on the "urge to stay home" with a resulting reliance on communications channels. During the recent war, a number of companies turned to videoconferencing as a replacement for international travel. When thecapabilityexists for at-home videoconferencing...

AN UNABASHED PLUG

During the last couple of years, we've received more project-based articles than we can get into print. In addition, we've printed a few articles that, because of space limitations, had to be severely reduced from their original length. In order to get more articles (and more of the articles) into your hands, we're publishing the first in a series of books: Circuit Cellar Project File, Volume I. This book has twelve projects featuring hardware and software and is available with a companion disk containing all the software. There's an ad on page 36 that tells you how to order the book. We hope you'll enjoy it and will write telling us what you'd like to see in Volume 11.



Cover Illustration
by Robert Tinney

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

In This Issue...

FEATURES

SPECIAL SECTION: Home Automation

DEPARTMENTS

Circuit Cellar BBS-24 Hrs. 300/1200/2400 bps, 8 bits, no parity, 1 stop bit, (203) 871-1908: 9600 bps HST
(203) 87 I-0549

The schematics provided in Circuit Cellar INK are drawn using Schema from Omatiin Inc. All programs and schematics in Circuit Cellar INK have been carefully reviewed to ensure that their performance is in accordance with the specifications described, and programs are posted on the Circuit Cellar BBS for electronic transfer by subscribers.

Circuit Cellar INK makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors. Furthermore, because of the possible variation in thequalityand condition of materials and workmanship of reader-assembled projects, Circuit Cellar INK disclaims any responsiblity for the safe and proper function of reader-assembled projects based upon or from plans, descriptions. or information pub lished in Circuit Cellar INK.

CIRCUIT CELLAR INK (ISSN 0896-8985) is published bimonthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second-class postage paid at Vernon, CT and additional offices. One-year (6 issues) subscription rate U.S.A. and possessions $17.95, Canada/Mexico $21.95, all other countries$32.95. All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank. Direct subscription orders to Circuit Cellar INK, Subscriptions, P.O. Box 3050-C. Southeastern, PA 19398 or call (215) 630-1914.

POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 3050-C. Southeastern, PA 19398.

READER'S INK

Letters to the Editor

THE FUNGUS AMONG US

This letter is in reply to the letter published in CIRCUIT CELLAR INK #18 from Mr. Andrew Mancey of Guyana.

I study in Manipal, situated on the west coast of India. The weather is hot and sunny for eight months of the year, but during the remaining four months it rains cats and dogs. We **have** a total of over 200 PC systems **here and** most of us engineering students use and have floppy disks.

During the rainy season we have the problem of fungus and mildew. I've been able, for the last four years, to keep my floppy disks going by doing the following:

Put a small packet of silica gel (in an active state, i.e., deep blue color) in each box of floppies. The silica gel changes to white as it absorbs moisture. To activate it again, simply bake it until it turns blue.

Bury the floppy disks, when storing them, under clothesin a cupboard which has a 50–60-watt bulb switched on 24 hours a day.

Follow these two tricks and you can say goodbye to fungus and mildew. I hope these techniques can help another reader keep their floppy disks and data intact.

J.N. Kumar
Coimbatore, India

STANDARD DEVIATIONS

I have some comments on the article "Adjusting Standard Deviation to Sample Size" by Charles I'. Boegli in **CIRCUIT** CELLAR INK #20. The flaw in his discussion of standard deviations is that he fails to distinguish between the standard deviation of a population and that of a sample drawn from that population. Likewise,a distinction should be made between the population mean and the sample mean (he uses X for both). The sample mean is the average of the observations comprising the sample, and the sample standard deviation is a measure of the dispersion of just those observations about that average.

Mr. Boegli states that "some texts recommend, in certain cases, dividing by n-l instead of n.. .," implying thatthispracticemaynotbewidelyaccepted;Alltextsthat I've consulted follow this practice. (Meyer, "Data Analysis for Scientists and Engineers," Wiley 1975, gives both formulas, stating that "For most cases of interest, there is no significant difference between s and σ, and either may be used." I would disagree, especially for small samples.) Here we are considering the sample statistics, and the Roman s is used for the standard deviation in order to distinguish it from the standard deviation of the population, denoted by the Greek cr. So, why n-l? Well, the proper divisor is the "degrees of freedom" associated with that sample, and where a single parameter (here the mean value of the population) is estimated, this is one less than the sample size. (More generally, it is n-p, where p is the number of parametersbeing fitted.) For example, consider a sample size of 10 with average, say, 5. We can arbitrarily change any nine of the observations without changing the average; the tenth can then be calculated, and so cannot also be given an arbitrary value. Remember we are dealing with estimates based solely on the sample taken.

For the standard deviation of the population, the proper divisor is n, and Mr. Boegli's equation (1) would be correct if the sample mean, X, is replaced by the population mean (commonly denoted by the Greek η). Dividing by η is also proper for the sample standard deviation if the sample mean is replaced by the population mean, which may be known (or accurately estimated) from prior information. Since information not obtained from the sample is used, the constraint on degrees of freedom is removed.

Mr. Boegli's *absurdum* is not really absurd at all. For a sample size of 1, the average is the observation itself, and the standard deviation becomes O/O, which is indeterminate and not necessarily infinite. This simply tells us that a single observation per se tells us nothing about the distribution of the population.

I don't know how the correction factor table cited in the article was derived, and would be curious to learn. Is there a mathematical statistician in the house? It leads to more pessimistic (i.e., larger) estimates of standard deviation than does using n-l. *I've* no idea, though, as to why it differs from the conventional approach, and inclined to view it with suspicion. For a sample size of 1, Mr. Boegli's equation (3) estimates the standard deviation as zero,

implying that the population mean is always estimated exactly by a single observation!

Norman F. Stanley
Rocklan, ME

A PENNY SAVED...

I just wanted to thank CIRCUIT CELLAR INK for recently publishing an article by J. Conrad Hubert, "Running VGA on an IBM Professional Graphics Display."

As owner of an automation consulting firm, I know that it pays to have **more** information than the competition. For those of you who may not agree, Mr. Hubert's article allowed my firm to save approximately $10,000.

Based on his work, I purchased several surplus Mitsubishi 19" analog monitors. With his circuit, we now have 800 x 600 VGA monitors for our CAD stations.

There is one snag, however. This setup is not a multisync. When the software kicks the video into a different (lower) frequency, you are stuck.

The best solution would have been to find a way to "force" the card to stay in the higher resolution (something akin to DOS MODE, perhaps?). Unfortunately, we weren't able to do that.

Our solution was to purchase 12" mono VGA monitors and daisy-chain them to the fixed-frequency 19" monitors. For about $100 per unit, the combination makes for a very workable and good solution.

Gary **Kuelzow**
Navesink, NJ

A TSR FIX

I have found that the ability of transferring the segment and offset address of the TSR location to the new keyboard interrupt vector seems to fail on some assemblers. **The sequence**

```
mov dx,offset new_kb_vec
mov al,16h
mov ah,25h
int 27h
```

seems to read the proper vector but fails to set the vector properly. So a quick "universal" fix is possible by making the following changes in the code:

1. In the beginning of the code, just below the title, add:

```
TITLE   expand
ABSO    segment at 0
        org 4*16h
kb_int  dd ?        ;BIOS keyboard interrupt
ABSO    end
        ;
_TSR    segment
```

2. and change the INIT procedure too:

```
init:
    sub ax,ax       ;make 0
    mov es,ax       ;point to beginning of mem
                    ;set up no seg overrides
    assume cs:_TSR,ds:_TSR,es:ABSO
    mov ax,word ptr kb_int  ;get orig int handler
    mov bx,word ptr kb_int+2
    mov word ptr old_kb_vec,ax ;save old int vect
    mov word ptr old_kb_vec+2,bx
    mov word ptr kb_int, offset new_kb_vec
    mov word ptr kb_int+2,cs

    mov dx,offset int
    int 27h
```

Christopher Ciarcia
Los Alamos, NM

We Want to Hear From You!

Write letters of praise, condemnation, or suggestion to the editors of Circuit Cellar INK at:

Circuit Cellar INK
letters to the Editor
4 Park Street
Vernon, CT 06066
Tel (203) 875-2 199
Fax: (203) 872-2204

Circuit Cellar BBS: 'editor'

HOME AUTOMATION CONTROL CENTER

A home automation system that allows remote control of essential home functions without retrofitting or hard wiring has been announced by Group Three Technologies Inc. SAMANTHA, an acronym for Security And M ANagement Through Home Automation, is a microprocessor-based system featuring keyboards/keypads, touchtone phones, and remote sensory inputs. It utilizes both digitally generated and prerecorded voice, as well as digital display outputs. SAMANTHA is a "plug and play" system in that major components communicate with each other through existing wire harnesses (power and telephone) and radio frequency signals. It is both FCC and UL approved.

SAMANTHA comes with a Personal Home Director (PHD) that is the nerve center of the system. It contains an emergency dialer, battery back-up, backlit liquid crystal display, voice synthesized help, LED status indicators, 64 programmable macro sequences, and full electronic controls. SAMANTHA also includes Room Directors to allow remote control of its functions from up to fifteen rooms by transmitting commands over existing phone wires. It contains a speaker and microphone for audio/voice communications, a keypad for entering commands, and an optional temperature sensor.

Some of the features offered by the unit include: heating and cooling management, lighting and appliance control, telephone answering device, personal emergency response, selective control intercom, personal memos/time management, and a full-featured security system.

The basic SAMANTHA package is priced at $1495. The advanced system begins at $1995.

Group Three Technologies, Inc.
2125-B Madera Rd.
Simi Valley, CA 93065
(805) 582-4410
Fax: (805) 582-4412

Reader Service #504

VIRTUAL UART COMMUNICATIONS PACKET CONTROLLER

A device that enables the serial port of a personal computer to accept virtually any synchronous or asynchronous communications protocol has been announced by Silicon Systems Inc. The SSI 73111650 SPC Serial Packet Controller employs a novel virtual UART technology to achieve this flexibility.

The personal computer sees the "650" chip as a common 550-type asynchronous UART, but to a device communicating with it, the 650 can emulate virtually anything, including an 8350-type synchronous UART. The 650 features Manchester encode/decode capability so it can even be used with fiber-optic links. NRZ, NRZI, and FM encoding are also supported.

Applications for the 650 include: input/output control for any PC or workstation (regardless of operating system), an emulator for prototyping communications links, a packet controller for LAN and WAN applications, and multitasking operations.

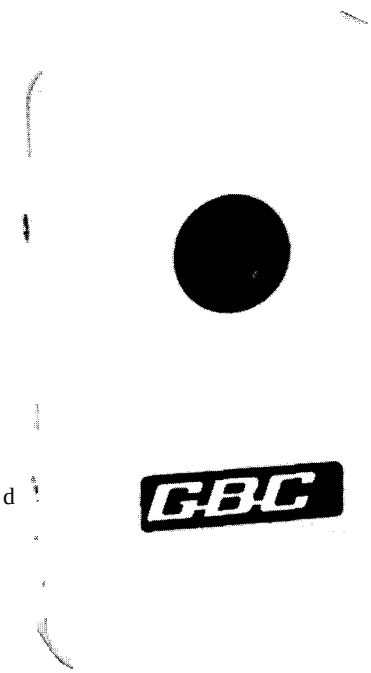
The 650 includes a power-down mode to extend battery life when used in portable applications, and 32-bit CRC error checking for full V.42 capability.

The SSI 73M650 SPC sells for $15 in quantities of 100.

Silicon Systems, Inc.
14351 Myford Rd.
Tustin, CA 92680
(714) 731-7110
Fax: (714) 669-8814

Reader Service #505

MINIATURE SOLID-STATE CCD CAMERA



A series of ultra compact, highly versatile **CCD** cameras has been introduced by CCTV Corp. The cameras are unique, not only because of their small size (3" x 2" x 1"), but in the fact that while utilizing a solid-state imager, the camera does not require an autoiris. Their design allows for full-range light compensation and superior handling of backlit situations.

The present family consists of six models, all using identical electronics, and come standard with a 4mm lens for a 78-degree field of view. They operate with 7 to 12 volts DC and include an AC power module and mating video connector. Full video is achieved with light levels as low as 2 lux. The electronic shutter has a speed from 1/60 to 1/15,000 second and there is no geometric distortion. A minimum of 10 shades of gray scale can be resolved.

The Model CD-100, $495, camera (shown) is designed to be mounted in any door. It comes with the electronics attached to a steel front plate that is mounted on the outside of the door and a matching back plate for the inside of the door. These plates are clamped across a rectangular hole cut into the door.

The Model IW-100, $495, is designed to be installed flush in any wall in a standard single gang electrical box; the SM-100, $495, is similar but comes with a surface-mounted Wire-Mold-compatible box.

The **PC-100,** $525, can be discreetly concealed within a 5" x 7" picture frame, and the IS-100, $525, camera is hidden within a working intercom speaker. The **CCD-100, $475,** is ideal for general-purpose surveillance.

CCTV Corp.
315 Hudson St.
New York, NY 10013
(212) 989-4433
Fax: (212) 463-9758

Reader Service #500

HIGH-CAPACITY ROM DISK AND DRIVE EMULATOR

The ROMDISK PCF from Curtis Inc. is a versatile solid-state peripheral storage device for PC/XT/AT and EISA computers. It can be used in most portable, desktop, and industrial computers to provide low-power, nonvolatile, solid-state memory without mechanical floppy and hard disk drives. The PCF is up to 20 times more reliable and many times higher in performance than mechanical systems.

The PCF can emulate a single high-capacity data disk, or dual disks to provide auto-booting from a diskette image, as well as providing high-capacity data storage. One SIMM can emulate a bootable diskette. The remaining six SIMMs must be Flash Memory for a Flash File System or SRAM for a read/write disk. SIMMs provide up to 14 MB total data storage by using seven 1-MB or 2-MB Flash Memory modules or 1-MB SRAM modules. A proprietary Flash gate array and microcontroller provide the core architecture that allows high-speed erasure and programming of Flash independent of the computer system's CPU.

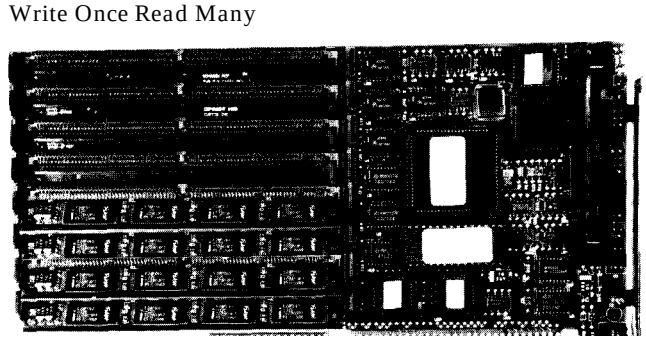
The PCF provides a dramatic enhancement of disk read performance as well as "instant-on" capabilities. Using Flash technology and the Microsoft Flash File System, the PCF operates as a Write Once Read Many (WORM) device that can add data to the Flash memory until full. Full memory can be bulk erased and reprogrammed. The PCF can bulk erase Flash Memories at a rate of 4 MB in ten seconds and program Flash at a data transfer rate of up to 100 KB per second.

The ROMDISK PCF uses high-density surface-mount, low-power CMOS digital logic, a gate array, and microcontroller. Standby power consumption is less than 0.03 watts and the bus interface is 8-bit compatible.

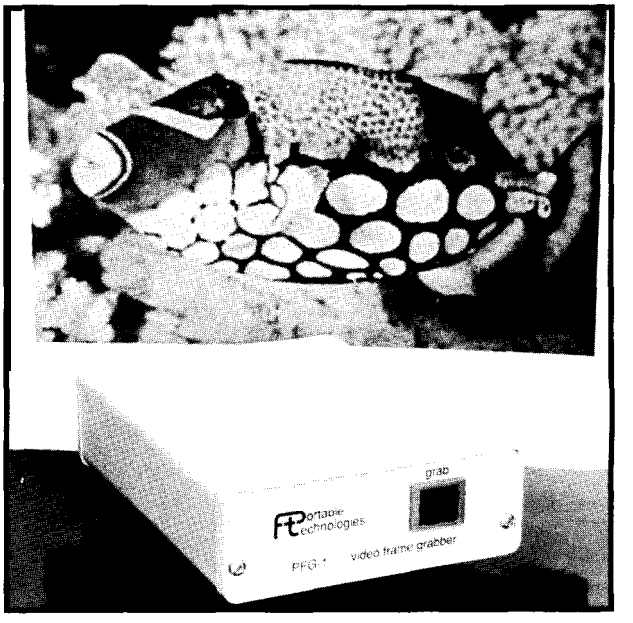
The price for the ROMDISK PCF is $895 including one Flash SIMM or $995 including one SRAM SIMM. Additional SIMM prices are $395 for Flash SIMMs and 5495 for SRAM SIMMs.

Curtis, Inc.
2837 North Fairview Ave.
St. Paul, MN 55113
(612) 631-9512
Fax: (612) 63 l-9508

Reader Service #501



PORTABLE VIDEO FRAME GRABBER



An easy, affordable way to capture gray-scale images from live video sources is available from Portable Technologies. The **PFG-1 Portable Video Frame Grabber** is a battery-powered unit that can digitize and store a video frame in 1/60 second. The resultant image has a resolution of 320 x 200 pixels with 64 levels of gray. Square pixels are obtained with standard NTSC or RS-170 video signals from video cameras, camcorders, VCRs, or TV monitors.

The PFG-1 can operate without being connected to a computer, and keeps its captured image in internal memory until overwritten by another frame grab. The stored image can be transferred to any computer equipped with a standard RS-232 serial interface, with selectable data rates of 4800–38.4k bps. The image readout time is 14 seconds at 38.4k bps. Frame grabbing can also be initiated under computer control through the serial interface.

The 4.2" x 1.5" x 5.5" unit is able to operate for up to 10 years, or more than 100,000 frame grabs and readouts from a single 9-volt lithium battery. The unit is powered continuously and is therefore ready for image capture at any time.

The PFG-1 comes with PGRAB software which runs on IBM PC and compatible computers with either CGA, EGA, or VGA graphics and at least 128K of RAM. PGRAB controls both the frame grabbing process and readout of the frame grabber. The image can be displayed on screen, adjusted for brightness and contrast, and printed directly to an HP LaserJet II and compatible printers. Images can be saved as linear files for quick recall or as TIFF files which can be imported to most desktop publishing applications.

The PFG-1 is shipped with the 9-volt lithium battery installed, and includes, in addition to the PGRAB software, a DB-9 serial cable, DB-9-to-DB-25 adapter, and an RCA video cable. The PFG-1 sells for $269.00.

Portable Technologies
P.O. Box 20763
Castro Valley, CA 94546
415) 537-4954

Reader Service #502

PC BUS DATA ACQUISITION INTERFACE

A short-slot, PC-compatible card offering 12-bit A/D conversion, 12-bit D/A conversion, digital I/O, and timing capability has been announced by Real Time Devices Inc. The ADA2100 incorporates six-layer construction and sample-and-hold circuitry to ensure low-noise performance and accurate digitization of dynamic signals. The A/D converter is the industry standard HI-574 that converts an analog signal into its digital equivalent in under 20 microseconds.

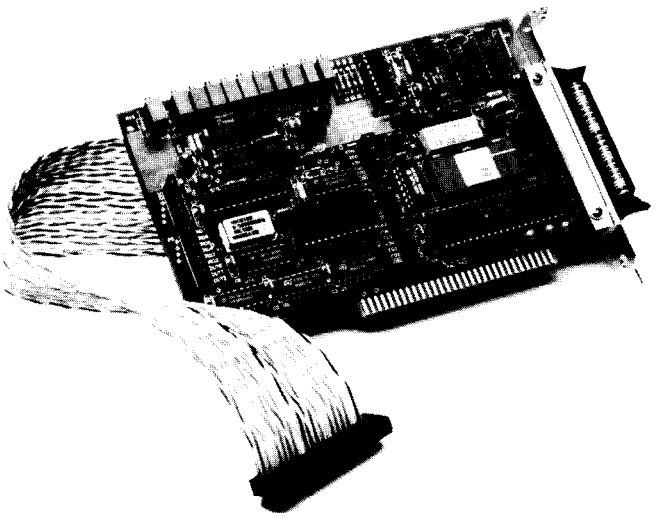
The ADA2100 supports eight singleended or four differential high-impedance analog input channels. Input ranges of 5 volts or 10 volts are supported, and a programmable gain amplifier provides software-selectable input gains of 2, 4, 8, and 16.

Two fast-settling, closely matched D/A converters are provided to generate analog output signals. The output ranges are jumper-selectable for unipolar or bipolar operation and can be configured for 5- or IO-volt output ranges.

Timing and counting functions are provided by three 5-MHz timer/counters based on the popular 8254 chip. The board also includes 16 digital I/O lines from an 8255 chip that can be configured as digital input or digital output.

Included with each card is a disk with sample programs illustrating control in BASIC, Turbo Pascal, Turbo C, and assembly language. The ADA2100 is compatible with the Atlantis series data acquisition software from Real Time Devices.

The ADA2100 sells for $395.



Real Time Devices, Inc.
820 N. University Dr.
P.O. Box 906
State College, PA 16804-0906
(814)234-8087
Fax: (814) 234-5218

Reader Service #506

WANT TO SEE YOUR NEW PRODUCT IN CIRCUIT CELLAR INK'S NEW PRODUCT NEWS SECTION?
JUST SEND US YOUR NEW PRODUCT ANNOUNCEMENTS OR PRESS RELEASES

SEND TO:
CURTIS FRANKLIN, JR.
EDITOR-IN-CHIEF
CIRCUIT CELLAR INK
4 PARK STREET
VERNON, CT 06066

MICROCOMPUTER DEVELOPMENT SYSTEM

A low-cost microcomputer development system for the Motorola MC68HC05 family of microcomputer chips has been announced by The Engineers Collaborative Inc. The MCPM-3 is a complete PC-based development system to design, test, debug, and program the MC68HC05 family. It supports the MC68HC705C8 and MC68HC805C4 versions and others with optional adapters, and requires only an IBM PC or compatible computer, text editor, and RS-232 cable to create a complete system.

The MCPM-3 accepts standard text source code files and offers stand-alone programming capability. Each system includes a cross-assembler, simulator/debugger, programmer, and driver program.

The TASM05 cross-assembler creates object code files from standard text input. Either Motorola S-record or Intel hex output formats may be selected. Extensive error checking and descriptive error messages are provided to help avoid bugs.

The SIMHC05 simulator/debugger allows the simulation and verification of program logic before programming the microcomputer EPROM. A full-screen display of the microcomputer's memory, registers, I/O ports, and special-purpose pins is provided. Single step and breakpoint features are included.

The PROG05 programmer driver software provides a menu-driven environment for programming the microcomputers. The software downloads files to a nonvolatile RAM chip in the programmer, and the RAM chip, in turn, programs the microcomputer. The PC environment is initially used to set up the RAM. Removing the programmer's RS-232 connection to the PC allows the unit to function in a stand-alone mode. A wall-mounted transformer provides all power to the programmer.

The MCPM-3 base system sells for $495. Programmers with driver program only are **$395.**

The Engineers Collaborative, Inc.
Route 3, Box 8C
Barton, VT 05822
(802) 525-3458
Fax: (802) 5253451

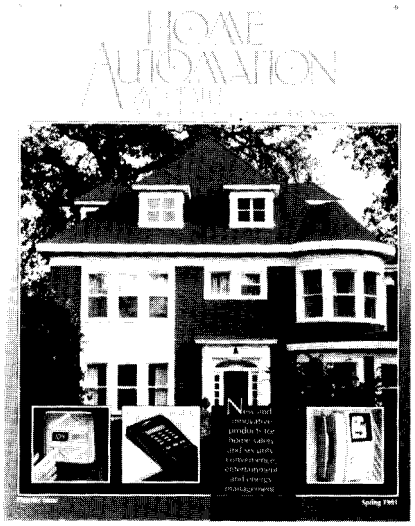
Reader Service #503

HOME AUTOMATION CATALOG

Heath Company, a leader in high-quality electronics for over forty years, launches the premier issue of **Home Automation** by Heath, an all-new journey into innovative home electronic products.

Home automation encompasses a broad scope of consumer products designed for safety, security, convenience, entertainment, and energy management. These products are designed to create a home that is a safer and easier place to live.

The premier issue of Home Automation by Heath is a 36-page, full-color catalog that is highly consumer oriented, aimed at the do-it-yourselfer home enthusiast, as well as the electronics innovator. Products include whole-house automation and



security systems, motion-sensing lighting controls, wall switches, security cameras, portable security alarms, energy-saving thermostats, automatic lawn sprinklers, and more.

Home Automation by Heath is a catalog that will inform and educate the consumer. It provides valuable information on the various types of technology used in the products, and how these technologies make the products work. The consumer will learn about topics such as X-10 products and how they work, passive infrared technology, and how to install a wireless security system.

Heath Company
Dept. 350-056
Benton Harbor, MI 49022
(800) 444-3284
(616) **982-5950**
Fax: (616) **982-5577**

Reader Service #507

HOW TO AUTOMATE YOUR HOME

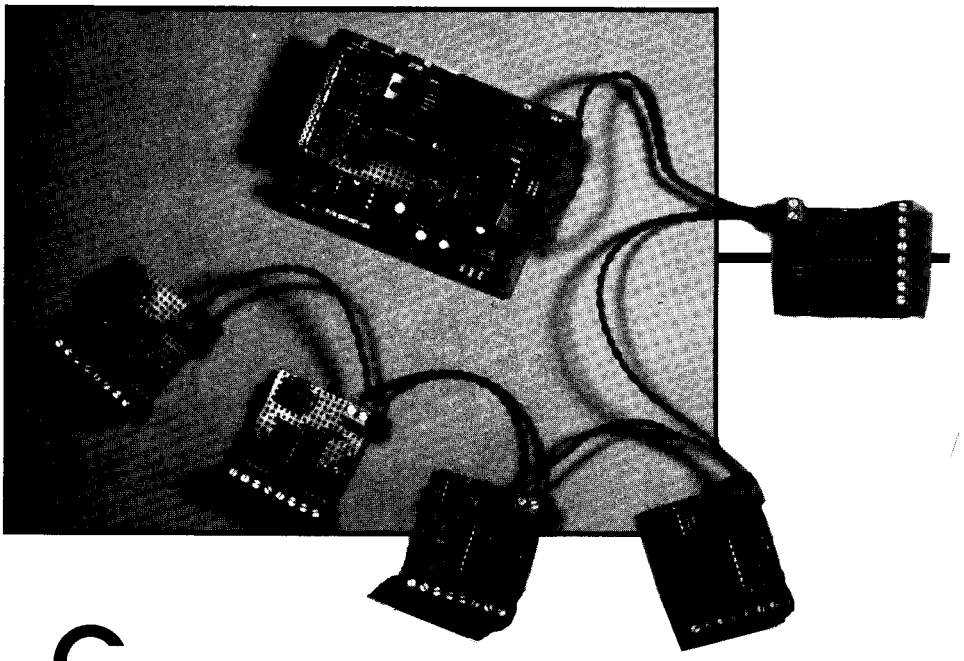
Home Automation USA announces their new book entitled "How to **Automate Your** Home." Aimed at the do-it-yourselfer, the book covers all aspects of home automation including light and appliance control, security, entertainment, communications, energy management, and more.

"How to Automate Your Home" contains 128 pages and 115 illustrations that are designed to make it easy to understand and install various levels of automation systems in new or existing homes. Itsells for *$29.95.*

Home Automation, USA
P.O. Box 22536
Oklahoma City, OK 73123
(405) 840-4751
Fax: (405) 842-3419

Reader Service #508

FEATURE ARTICLES



Controlling and monitoring dispersed locations can be accomplished in several ways. Typically, you would use either a central control unit or distributed controllers, multidropped on an RS-485 network. If all we want to do is monitor and control some binary points, these configurations may not be the optimal solution. The drawbacks of the centrally controlled system is the need to run multiple cable bundles from the sense/control devices back to the control unit. In the distributed approach, a protocol must be devised for the controllers to communicate with one another and perhaps to some central device. Powering the controllers may also be a problem. Power can be carried on extra wires in the communications cabling or local power can be supplied if a source is available.

The S-ARTnet offers a solution to these problems. The S-ARTnet consists of two elements: a network controller that operates under control of stored programs and multiple SART satellite modules. Each S-ART satellite supports two sense inputs and two control outputs; up to 30 satellites can be daisy chained on a single wire pair. Cablingconstraintsare eased due to the fact that the satellites use a single wire pair not only as the signaling medium but for power as well. The controller can be adapted to function as a translator for a PC-based control system via a serial RS-232 link or can operate in a stand-alone fashion without the need for any central intelligence. I'll cover the satellites and S-ART network controller in detail, then we will look at applying the system. First, let's get familiar with S-ARTs.

S-ARTS

S-ART: Serial, Addressable, Receive/Transmitter. This part, also known as a Cherry CS-212, is a 16-pin circuit designed for data transmission on a two-lead cable. The CS-212 is an I^2L/Linear IC consisting of approximately 275 gates, 100 bipolar transistors, and 40 resistors. The circuit is specially developed for alarm systems where it is desired to identify each detector individually. There can be up to 30 S-ART circuits on the same two-lead cable. This cable carries information both to and from the S-ART and provides power to the S-ART.

The S-ART works on the principle where a packet with an address header is sent out over the common cable. When an S-ART on the cable recognizes its address in the header, it carriesout the command contained inkhe packet. The command can be one of two things:

1. Transmit data from the line cable to the S-ART's two outputs OUT0 and OUT1.

S-ARTnet-A Powerful Controller Network

FEATURE ARTICLE Part 1

John Dybowski

Designing a low-cost Network Around the S-ART

2. Answer the S-ART controller with the condition of inputs IN0 and IN1.

The signal on the line is divided into three states in order to give a time signal for synchronizing and a data signal containing addresses, commands, and so on. Typical signal levels for the three states would be 15 V, 7.5 V, and 0 V.

NETWORK CONTROLLER

Conceptually, the S-ARTnet controller can be thought of in two parts: the microcontroller module and the interface adapter. For the microcontroller, we will use an 8031 with an address latch, EPROM, and a serial RS-232 port. In order not to belabor this standard configuration, we will simply think of this as a component of our system. A standard module exists that satisfies the requirements: the Cottage Resources Control-R I. Figure 1 shows the circuitry on the Control-R I. We will proceed by structuring the interface adapter to be plug compatible with the Control-R I expansion header.

The interfaceadapter provides the 15-V S-ART power and the interface circuitry to communicate to the satellite modules and is shown in Figure 2. The configuration described contains the power supply and drive circuitry forasingle-cables-ARTnetwork.This compact,low-costarrangementallows monitoring up to 60 sense points and driving up to 60 control points.

S-ART power is derived from the logic +5 volts and is stepped up to 15 volts with a National Semiconductor LM2577-15.0 switch mode regulator (shown in Figure 2a). Requiring a minimum number of external components, the LM2577 is inexpensive and simple to use. Included on the chip is a 3-amp NPN switch and its associatedprotectioncircuitry,consistingof current and thermal limiting and undervoltage lockout. Other features include a 52-kHz fixed-frequency oscillator that requires no external components, a soft start mode to reduce in-rush current during startup, and a fixed internal voltage reference.

The cable interface in Figure 2b consists of four saturation mode transistors wired to provide switching of

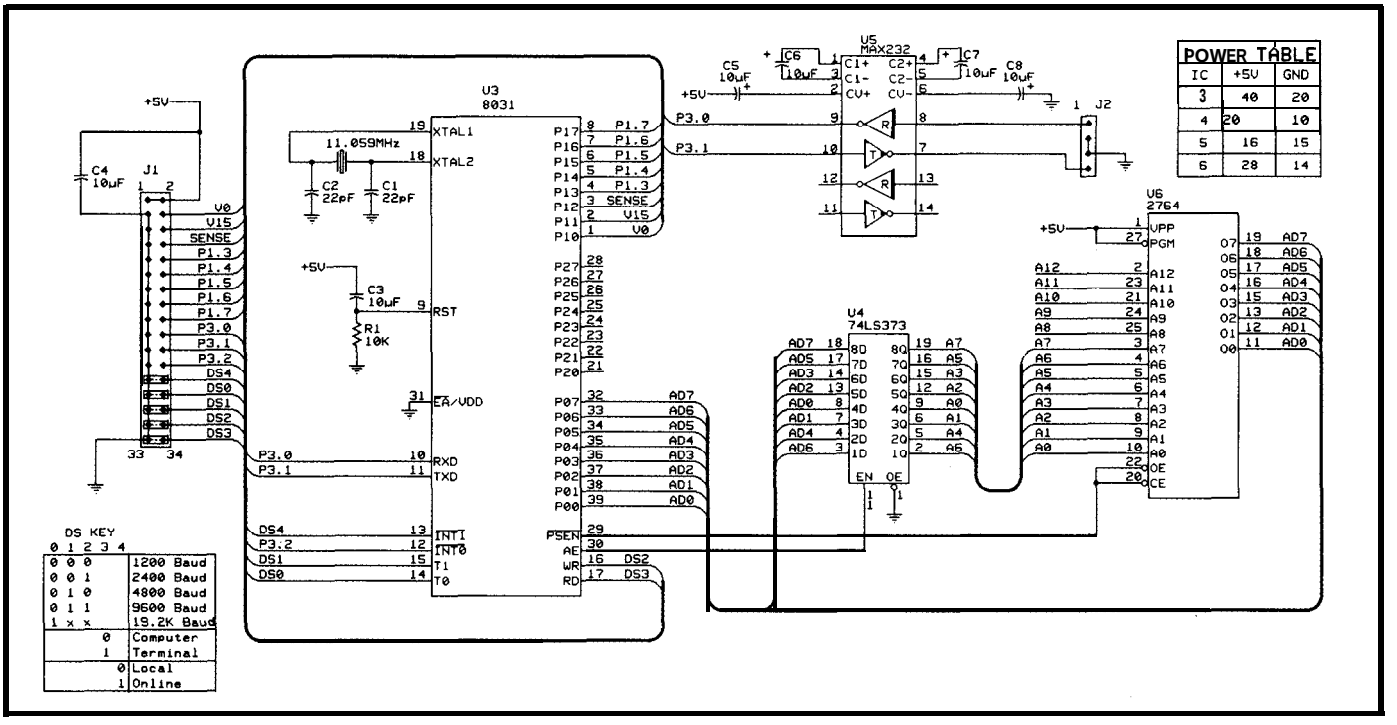


Figure 1 -The *Control-R I processor board provides the brains for the network controller.*

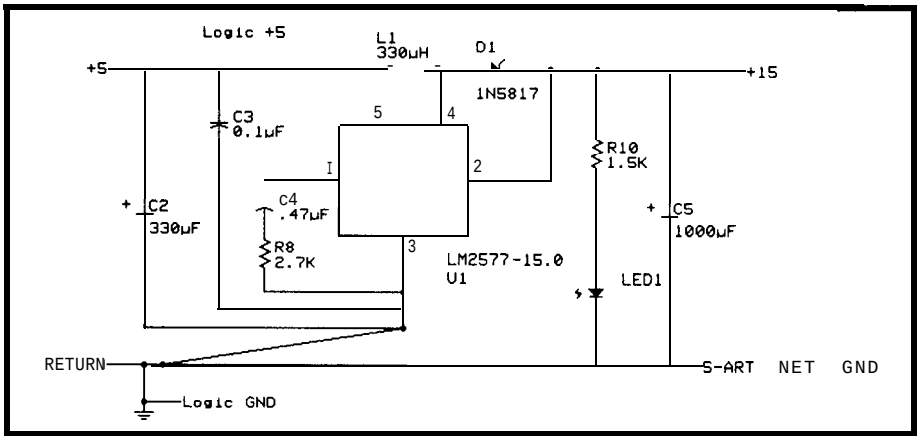


Figure *Pa-An LM2577-* **15.0** *provides power for the network*

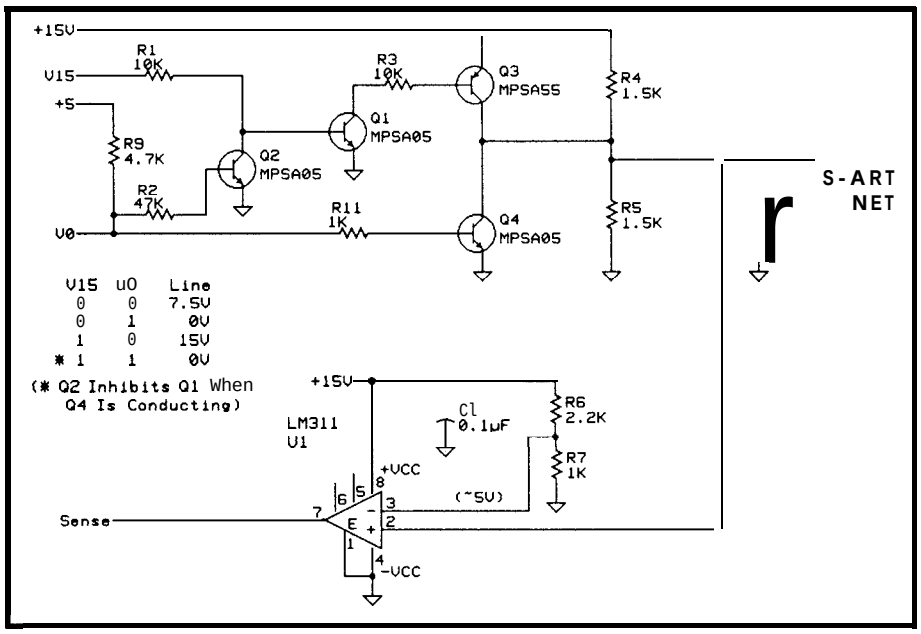


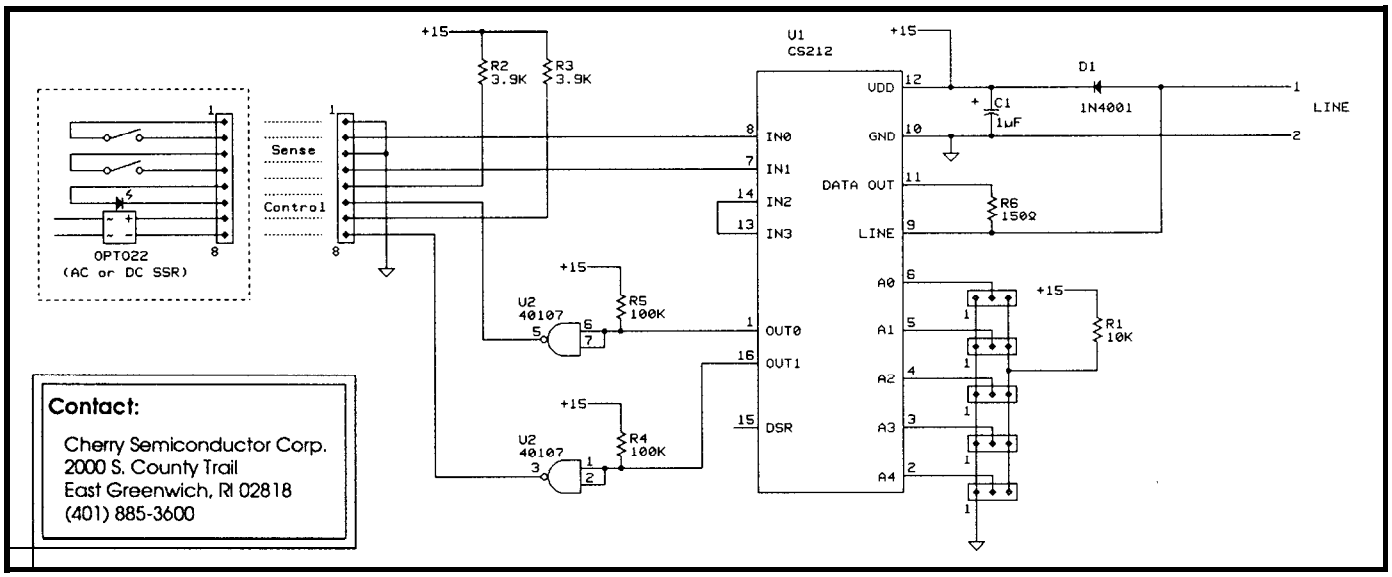
Figure 2b—*The* **network** *interface is capable of asserting 0 V, 7.5 V, or 15 V.*

the full rail 15 volts, 0 volts, or 7.5 volts. In this arrangement, Ql provides the level shifting for Q3, the 15-volt switching transistor; Q4 is the 0-volt switch. If neither Q3 nor Q4 is conducting, the line is pulled to 7.5 volts via a resistor divider consisting of R4 and R5. Q2 clamps the base drive to Q1 when Q4 is on to safeguard the drivers during power up initialization and in the event of loss of software control. This inhibits power from being applied to the network until the controller is ready to initialize the satellites to their default states. Line monitoring is accomplished via an LM311 comparator wired to trip at approximately 5 volts.

The controller interface uses three processor pins: two for signal switching and one for line monitoring.

S-ART SATELLITES

The two-lead bidirectional cable that carries that signaling to the S-ARTs also provides power to the sat- **(shown** in Figure 3). The line signal is rectified and filtered via a blocking diode, D1, and capacitor, Cl, and is used for the power supply for the chip. Pin 12 is the power supply pin; 10 is the ground return. The IC will function with a supply voltage of between 10 and 18 volts. The S-ART also decodes the line signal into clock and data signals used inside the IC. Clock and data discrimination is accomplished in the S-ART using a dual comparator scheme, where the two trip points are set at approximately 6 and 12 volts with about 0.7 volts of hysteresis. The upper trip point derives the clock and the lower derives the data. Therefore, if the signal swings from 7.5 to 15 volts, a "1" is recorded; a transition from 0 to 15



gure 3—*Each satellite consists of nothing more than the S-ART chip and a handful of support parts.*

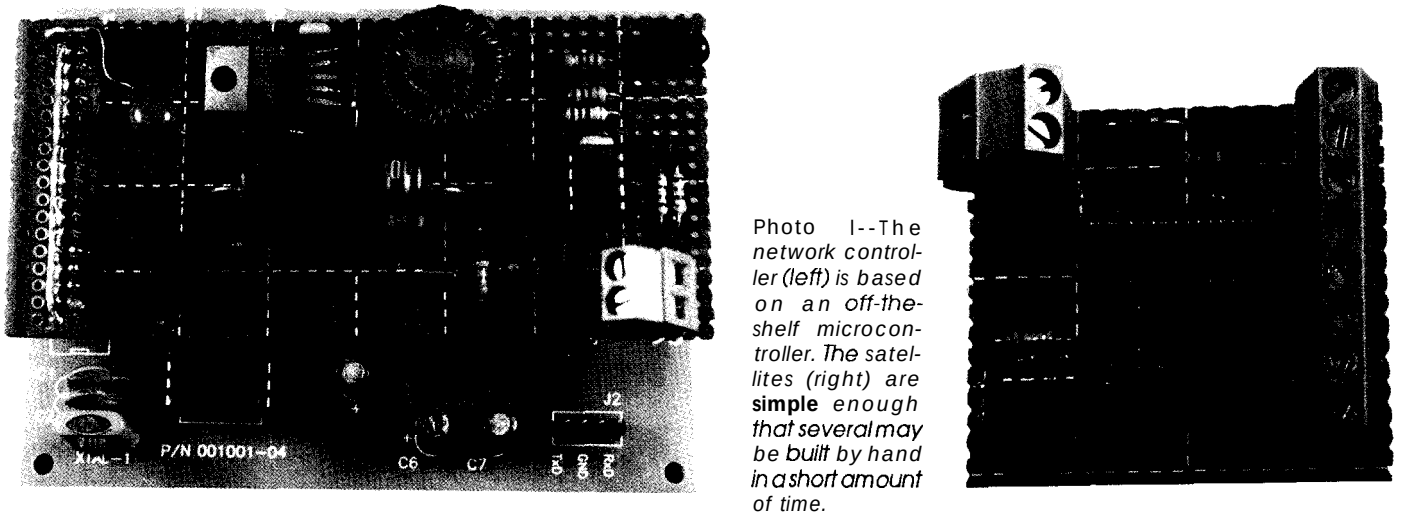


Photo I--The network controller (left) is based on an off-the-shelf microcontroller. The satellites (right) are simple enough that several may be built by hand in a short amount of time.

volts registers a "0." Data is received on pin 9. Figure 4 shows a typical line signal and how it is decoded.

The address for the satellite is set by connecting pins 2 through 6 to either the supply voltage through a 10k resistor, RI, to select a logic one, or to ground for a logic zero.

The S-ART accepts addresses and commands in 10-bit words. Three types of words may be generated: Sync, Read, and Write.

Synchronization is obtained by providing the S-ART with eight or more ones followed by a zero. Figure 5a shows what a sync word looks like.

To check the status of an S-ART, a "read" word must be sent as in Figure 5b. The first 5 bits must correspond to the address of the S-ART to be interrogated. Bit 6 is the address parity bit and must ensure that the first bits have an even number of ones. The next bit is the command-zero indicates a read operation. If the parity is correct, the S-ART will transmit its status with an internally generated parity bit. To receive the status, the controller must pull the line to approximately 7.5 volts, then the S-ART will transmit. If a one is transmitted, no change will occur on the line. If a zero is to be transmitted, the S-ART will pull the line down. In either case, the controller must now pull the line back to 15 volts in order to continue.

The inclusion of the data parity bit allows the detection of malfunctioning or nonexistent S-ARTs. Data is output on pin 11. This output is a saturated switch capable of sinking 10 mA at 0.4 V and 50 mA at 1 V on a transient basis. The 50-mA capability is needed to discharge the line capacitance. The 150-ohm resistor, R6, limits the current into this pin when the driver is on.

In order to update OUT0 and OUT1, a "write" word must be sent to the S-ART (see Figure 5c). The first five bits are the address, followed by the address parity. The next bit is the command, and is set to a one to denote a write operation. The next two bits are the actual output states for OUT0 and OUT1, respectively. Finally, a data parity bit is transmitted. An even data parity bit must be received by the S-ART in order for the update to OUT0 and OUT1 to take effect. The output signals, OUT0 and OUT1, are buffered by a CD40107 dual NAND driver with open-drain outputs. The CD40107's outputs will typically be used to drive low-current LEDs or solid-state relays at about 2 mA; R2 and R3 limit the source current to the external loads. OUT0 and OUT1 are Darlington-type open-collector outputs capable of sinking 1 mA at 1.2 V.

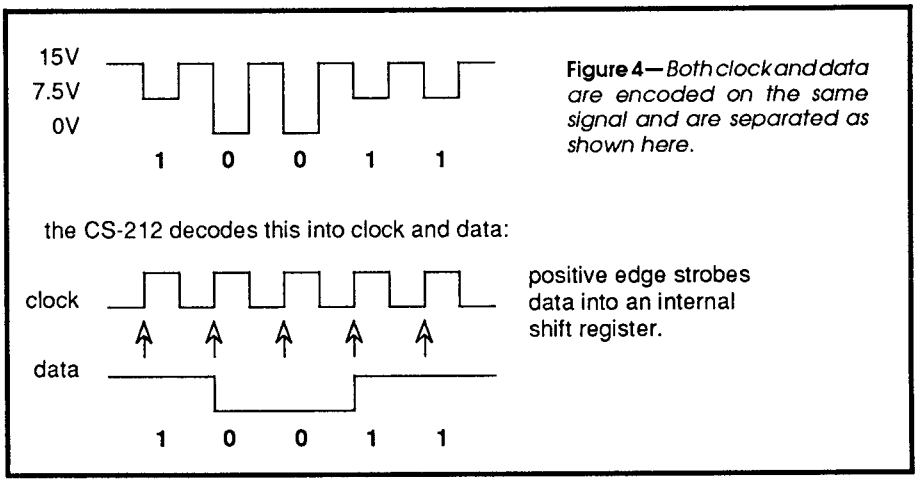


Figure 4—*Both clock and data are encoded on the same signal and are separated as shown here.*

THE S-ARTNET TRANSLATOR

In the role of an S-ARTnet translator, the controller functions as a gateway between the S-ARTnet and the host computer. The execution vehicle for the S-ARTnet controller is, as previously mentioned, an 8031. The 8031 has been pressed into service in a variety of applications over the years; everything from interpreting high-level languages and database manipulation to communication protocol converters. Although satisfactory results can be attained in these applications, the 8031 is, in fact, a control-oriented processor. I'm pleased to

S-ARTNET SCANNING

Communication with the S-ARTs consistsof bit patterns that can indicate one of three types of operations: sync, read, or write. First, we will define the routines to actually perform the signaling required to issue a one and zero bit.

A one bit consists of taking the line from 7.5 volts to 15 vol ts:

```
XMIT_ONE PROC
        CLR   LO        ; 0 volts off
        CLR   L15       ; 15 volts off
        DELAY_ 50US  ;  delay
        SETB  L15       ; 15 volts on
        DELAY_ 50US  ;  delay
        RET
        ENDPROC
```

When the line transits from 0 to 15 volts, a zero bit is indicated:

```
XMIT_ZERO PROC
        SETB  LO        ; 0 volts on
        CLR   L15       ; 15 volts off
        DELAY_ 50US  ;  delay
        SETB  L15       ; 15 volts on
        CLR   LO        ; 0 volts off
        DELAY_50US      ; delay
        RET
        ENDPROC
```

Now, a flexible routine that will transmit a bit pattern that we input via ACC, the bit count is in RO:

```
XMIT_TEMPLATE PROC
L?XT0:
        RRC   A
        JC    L?XT2
L?XT1:
        CALL XMIT_ZERO
        SJMP  L?XT3
L?XT2:
        CALL  XMIT_ONE
L?XT3:
        DJNZ  RO,L?XTO
        RET
        ENDPROC
```

With a little more processing we can generate an S-ART address sequence: input is in ACC:

```
XMIT_ADDRESS PROC
        ANL   A,#11111B ;  5-bit address
        MOV   C,P
        MOV   ACC.5,C  ;  even parity
        MOV   RO,#6    ;  6 bits to transmit
        CALL  XMIT_TEMPLATE ; go send
        RET
        ENDPROC
```

Using the routines we have devised, we can now read the status of an S-ART. On entry, ACC contains the S-ART's binary address. On exit, ACC contains PD, D1, and DO in the three least-significant bits.

```
READ_SART PROC
        CALL  XMIT_ADDRESS ; xmit address

        CALL  XMIT_ZERO ; the read command
        MOV   RO,#3    : 3 bits to read
L?RS0:
        CLR   LO
        CLR   L15       ; idle the line
        DELAY_50US
        MOV   C,SENSE  ; read the input bit
        RRC   A         ; position bit
        SETB  L15       ; drive line high
        DELAY_50US
        DJNZ  RO,L?RS0 ;  loop until done
        SWAP  A         ; put bits in low nibble
        RR    A         ; right justify
        ANL   A,#111B  ; return 3 bits
        RET
        ENDPROC
```

Finally, a command to control the S-ART outputs is easily implemented. ACC contains the S-ART address with the output bit pattern in the two least-significant bits of B:

```
WRITE_SART PROC
        CALL  XMIT_ADDRESS   ; xmit address
        CALL  XMIT-ONE ; the write command
        MOV  A,B        ; get output bits
        ANL   A,#11B   ; 2-bit command
        MOV   C,P
        MOV   ACC.2.C  ; even parity
        MOV   RO,#3    ; 3 bits to transmit
        CALL  XMIT_TEMPLATE ; go send
        RET
        ENDPROC
```

report that the S-ARTnet application uses the 8031 in its "native" mode that makes use of the 8031's best features. We are primarily interested in manipulating the internal RAM and bits. Good performance is assured since, after all, the 8031 approaches 1 MIPS when operating at this level.

The functions the controller must perform may be thought of as existing at several layers of a virtual process that we can logically partition, including physical scanning of the S-ARTnet, network status block management, system timcbase, and host communications. It has been said that the currency of a controller is time. How this time is budgeted can have an impact on the success or failure of a design. With this in mind, we will proceed to define how the functional blocks will be implemen ted.

Two levels of processing will be broadly defined: foreground and background, where the background tasks will be those that execute from within interrupt handlers. S-ARTnet scanning will be performed entirely as a foreground function. The system timcbase and host communications will run in the background. Status block maintenance will be split betwccn the foreground and the background. Input status will be logged as a part of the S-ARTnet scanning loop; the output states will update the control block directly out of the host command interpreter which is invoked from the SIO interrupt service routine.

The entire network is scanned approximately six to seven times per second, depending on the amount of background processing performed. The actual signaling to the S-ARTs is clocked at 10 kHz; the overhead of consulting and maintaining the control blocks accounts for the remainder of the time.

The control block for the S-ARTnet is composed of 30 bytes of RAM.

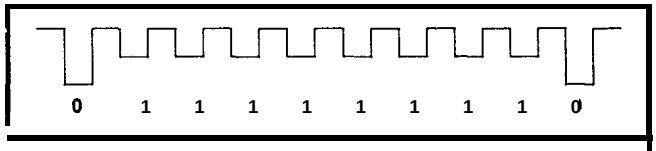


Figure 5a—*A standard S-ART synchronization sequence.*

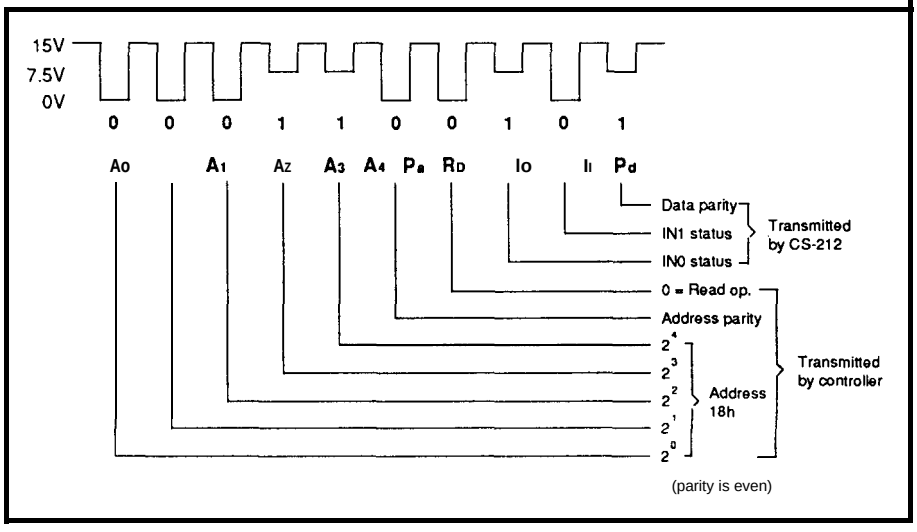


Figure 5b—*To check the status of an S-ART, a "read" word must be sent.*

Bit fields are defined for input status two bits), output status (two bits), and fault status (one bit). It can be seen that even with only the use of the 803l'sinternal RAM, up to three separate networks can be scanned with our simple microcontroller module while maintaining acceptable performance.

HOST COMMUNICATIONS

The SIO interrupt handler is coded on the premise that we can expect error-free communications. In other words, no error detection mechanisms are involved. This approach is reasonable if the communications line between the host and the controller is kept short.

An interesting component of this code is the SIO interrupt service routine that is structured as a multiple-entry-point vectored state machine. This approach provides for fast processing of interrupts by storing the address for the next section of process code prior to exiting the interrupt service routine. On entry, this vector is picked up and program execution

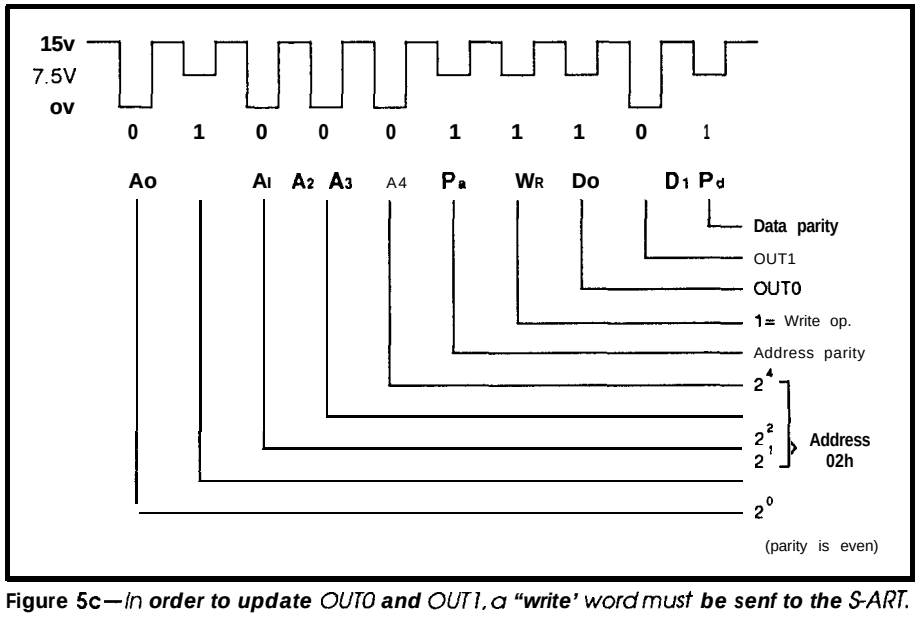


Figure 5c—*In order to update OUT0 and OUT1, a "write' word must be senf to the S-ART.*

Standard Read

R	Read the status of the entire S-ART network.
Rnn	Read the status of *nn*.
Rssee	Read the status of *ss* through ee. inclusive.

Continuous Read

The requested operation will autorepeat at the rate set via the "D" command. Continuous mode operation terminates on execution of any Read command or the Stop command.

C	Read the status of the entire S-ART network.
Cnn	Read the status of nn.
Cssee	Read the status of ss through ee, inclusive. (nn=ss=ee=OO-29, ss<=ee)
S	Stop continuous read operation.

Write Command

Wnnxx	Write the pattern xx to OUT1/OUT0 of S-ART *nn*. (nn=00–29, x=0 or 1)

Set Delay

This is the interblock delay for continuous mode operation.

D	Set delay time to default value of 2 seconds.
Dtt	Set delay time to *tt*. (n=00–99, in .25 second increments)

Select Transmission mode
(terminal or computer mode)

PO	Select computer mode.
P1	Select terminal mode.

Select SIO envelope characters
(computer mode)

E	Default the leading and trailing envelope characters to linefeed and carriage return respectively.
EO	Default the leading character to linefeed.
EOx	Set the leading character to the value of x.
EI	Default the trailing character to carriage return.
EOx	Set the trailing character to the value of *x*.

If the value NUL is loaded for either character, the transmission of the character is suppressed.

Table 1 -The *list* ***of supported commands includes everything necessary to implement a simple distributed*** *control system.*

continues at the vector address. One of the benefits of this approach beyond fast execution and clear program structure is that it is easy to install an entirely different service routine if desired. This feature is utilized to provide the two transmission modes described below.

COMMAND SET

The command line format consists of a linefeed, a command followed by its (optional) argument, terminatcd by a carriagereturn. The linefeed can be omitted. (If used, it resets the receive routine to a known state.) On receipt of the carriage return, the command string is sent to the parser for interpretation and execution. Command execution from within the interrupt service routine is feasible, since in this application we simply update some RAM and bit variables. Table 1 shows a list of supported commands.

VECTORED COMMUNICATIONS

By implementing the communications interrupt handler as a vectored state machine. the address of the next state is kept in internal RAM in an address vector location. On entry, the interrupt handlerpicksup thevectoraddressand program execution continues at that address.

For example, the transmitter interrupt entry and exit code:

```
TRAN_INT PROC
        PUSH   PSW
        PUSH   ACC
        PUSH   B
        PUSH   0
        PUSH   D
;
; PICK UP TRANSMIT VECTOR
,
        PUSH   TRAN_VECTOR
        PUSH   TRAN_VECTOR+1
        RET
,
; RETURN TO MAINLINE
,
TRAN_EXIT:
        CLR    TI
        POP    D
        POP    0
        POP    B
        POP    ACC
        POP    PSW
        RET1
        ENDPROC
```

The following fragment illustrates how the states progress using this method. Here we are finishing up the **status dump** and are proceeding from the SO status to the trailing envelope character.

```
; SO STATUS

L?XP6:
        MOV    A,STAT_STORE ; qet S-ART status
        MOV    C,ACC.0      ,. isolate stat bit
        JNB    ACC.2,L?XP6A ; jump if no error
        MOV    SBUF,#'X'    ; node error
        SJMP   L?XP6B
L?XP6A:
        MOV    A,#'0'       ; initial ASCII code
        MOV    ACC.0,C      ; now ASCII 1 or 0
        MOV    SBUF,A       ; go transmit
L?XP6B:
        MOV    TRAN VECTOR,#LOW L?XP7 ; nxt state
        MOV    TRAN-VECTORtl,#HIGH L?XP7
        JMP    TRAN_EXIT
,
; leave interrupt level
; TRAILING ENVELOPE CHARACTER
,
L?XP7:
        MOV    A,END_ CODE  ; trailing envelope
        JZ     L?XP8        ; jump if disabled
        MOV    SBUF,A.      , go transmit
        MOV    TRAN_VECTOR,#LOW L?XP8 ; nxt state
        MOV    TRAN_VECTOR+1,#HIGH L?XP8
        JMP    TRAN-EXIT
; leave interrupt level
```

S-ART Network Status

00: Cl =On C0=On S1=Off SO=Off
01: Cl=On C0=On Sl=Off S0=On
02: Cl=Off CO=Off -Node Fault-
03: Cl=Off CO=Off -Node Fault-
04: Cl=Off C0=Off -Node Fault-
05: Cl=Off CO=Off -Node Fault-
06: Cl=Off C0=Off -Node Fault-
07: Cl=Off CO=Off -Node Fault-
08: Cl=Off CO=Off -Node Fault-
09: Cl =On CO=Off S1=On S0=Off
10: C1=Off C0=Off -Node Fault-
11: Cl=Off C0=Off -Node Fault-
12: Cl=Off CO=Off -Node Fault-
13: Cl=Off CO=Off -Node Fault-
14: Cl=Off C0=Off -Node Fault-
15: Cl =Off C0=Off -Node Fault-
16: C1=Off C0=Off -Node Fault-
17: Cl =Off CO=Off -Node Fault-
18: Cl=Off CO=Off -Node Fault-
19: C1=Off CO=Off -Node Fault-
20: C1=Off CO=Off -Node Fault-
21: Cl =Off C0=Off -Node Fault-
22: C1=Off C0=Off -Node Fault-
23: C1=Off CO=Off -Node Fault-
24: C1=Off C0=Off -Node Fault-
25: C1=Off C0=Off -Node Fault-
26: C1=Off CO=Off -Node Fault-
27: C1=Off CO=Off -Node Fault-
28: C1=Off C0=Off -Node Fault-
29: C1=Off C0=Off -Node Fault-

able P-ANSI *terminal mode shows the status of the network.*

TRANSMISSION MODES

Two modes of transmission are provided: a mode suitable for interface with a communications program running on a computer, and a mode that provides a readable screen output on an ANSI terminal. The terminal mode contains information that would be superfluous to a comm program, but provides intelligible screen output and is useful for configuring and checking the system.

The computer interface format is:

e0 nn o1 00 i1 i0 e1

where:

e0 is the leading envelope character (default to linefeed)
nn is the S-ART node address, 00 through 29
o1 is the binary state of OUT1 (ASCII "0" or "1")
o0 is the state of OUT0
i1 is the binary state of IN1 (ASCII "0" or "1")
i0 is the state of IN0 (If the requested nodeisnot functional, the states of sl and s0 are reported as ASCII X.)
e1 is the trailing envelope character (dcfaul t to carriage return)

ANSI terminal mode is always initiated with the clear screen character (0Ch) and appears as in Table 2.

In the case of the outputs, "Off" indicates a logic 1 state, "On" denotes logic 0. For the inputs, "Off" is reported if the sense input is open, "On" is returned if it is pulled to ground.

Next time, we'll examine S-ARTboy, a PC program that rounds out our computer-based control center. We'll also look at using our controller as a traffic cop that will link S-ART satclli tcs and permit stand-alone operation of the S-ARTnet.

John Dybowski has been involved in thedesign and manufacture of hardware and software for industrialdatacollectionandcommunications equipment.

IRS

401 Very Useful
402 Moderately Useful
403 Not Useful

FEATURE ARTICLE

Software at the Hardware Level

Programming TSRs for Interrupt Handling

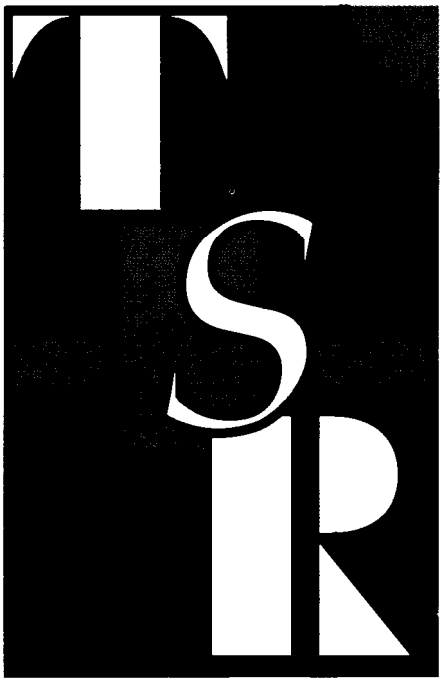
Chris Ciarcia

Most of us who push our computers to the limit are always in need of something new, easy, fast, and specially tailored. This, of course, requires us to keep learning **new** ways of doing things. And, like the old hot-rod mechanics of the '50s, we keep getting our hands dirty doing customization of this sort. So, let's pop up the hood and soup up the engine. Let's apply some of the very powerful low-level "greasy" tools available to us and learn new things about system performance improvement features and how to implement them using DOS system-level programming.

In any article like this, it's tempting to just leap into the code and hope that most readers will catch up and fill in the details. It's usually true that we experienced users maintain a basic understanding of some application languages. But most useful **procedures** require new adventures. Most of us have fun sitting in front of our keyboard, drinking too much coffee, swearing at the typos (yours and any code of mine here), and wondering why the machine crashed again.. .and again, and again. So let's try something new: a little show-and-tell, and walk each other through some basic simple applications which demonstrate what and how memory-resident programming and interrupt handling are and how to use them.

THE TOOLS OF THE TRADE

In the most ancient of times, some dumb, poor, overworked programmer actually learned to write his programs in bits. The hand of eternal light then gifted him with some intelligence and he decided to write a program that did this for him. This program was called an assembler, and it enabled the automatic encryption of such simple English-like commands as `MOV ah, 0EH` into those shifty bits. At first these programs were primitive and riddled with bugs, but these **days** they have become powerful tools supporting a wide variety of functions.



With such a lead-in, I obviously plan to center our code development around assembly language. Why? Isn't C a reasonably functional language for work of this type? Well yes and no. In some applications, C and other high-level languages are very appropriate, but there is always some tradeoff or compromise. In our case, assembly language is the best choice for the type of programming we will be doing. It's flexible enough to allow us to use our machine to its greatest potential, readable enough that even I can follow basic procedures without making a career out of bit mapping, and fast enough and economical in memory usage so that we can get our job done without wasting resources or time. Besides, I've never used assembler on a PC. So without any more fanfare, let's define what **we** call memory-resident programming and how interrupt handling becomes involved.

TSR'S AND INTERRUPTS

memory-program, the
resident application is de
encapsulate the basic features of its design function or purpose. How it is written depends on the scope of those design parameters. Its un however, lies in its method of operation. Programs of this sort typically remain resident within e memory space until they a system-y an operator or a generated interrupt. As such, they are called TSRs (terminate-and-dent programs). In general they come in two categories: active and inactive.

Active programs typica spond to a specific keystroke called a "hot key." When they are they take over the computer and perform their function before they return control to the program or system that originally "owned" the ma

Inactive TSRs are designed to respond to a calling program through a designated interrupt. When activated they perform their designed function (not unlike a subroutine) and control is returned to the calling

TSRs must monitor what both the operator and DOS do. When a TSR is activated, it sets up memory tables and readies itself for execution by connecting to a DOS interrupt. When everything is ready, the program determines its memory requirements, then sets the terminate-and-stay-resident-the "keep process"-DOS interrupt 21h, function 31h. This function allows the TSR to pass return codes while enabling it to use more than the 64K memory limit. When the programexits, the amount of memory available for executing programs is reduced by the amount assigned to the TSR, and the exit code reverts back to the parent process.

It's obvious that the idea of a memory resident program is tightly coupled to PC interrupt handling. Hard and soft interrupts are the main mechanism of communication and control between TSR applications and the operating system and as such are fundamental to system operation. As is implied by their name, an interrupt generates a short-term distraction, asking the computer for attention. The system processor then suspends its current process, retains restart information, and transfers control to a program called an interrupt handler. This handler then does its thing and returns control to the interrupted program. Therefore, for each interrupt the standard interrupt structure must handle the following situations:

- •save everything that the processor didn't save automatically when the interrupt occurred
- •block any interrupts which might interfere with the interrupt handler's operation
- •enable interrupts that can occur safely during the handler's operation
- *handle the interrupt
- •restore the processor registers saved in the first step
- •reenable interrupts
- *return to normal processing.

What is not commonly realized is that this interrupt process is constantly occurring while you are running your computer, many times per second.

We'll get back to interrupt handling later, but first lets start by getting our hands dirty and building the coded shell of our TSR.

A MINIMUM TSR SHELL

Most memory resident programs are designed to optimize execution speed while minimizing memory allocation overhead. As such, they are typically small (less than 64K) and can be loaded into one segment. They are usually compiled and linked into .COM files instead of the more general, but slower loading, .EXE file. The basic envelope for such a workable resident application can be segmented into four parts, shown in Listing 1.

```
_TSR    segment         ; part 1
        assume cs:_TSR, ds:_TSR
        org     100h
start:                  ;.part 2
        nop
done:                   ; part 3
        mov     dx,offset done
        int     27h
_TSR    ends            ; part 4
        end     start
```

Listing *1-A basic TSR shell.*

Part 1 defines our code segment. All segment registers are initialized to the same segment with the ASSUME directive. This tells the assembler which segment to associate with each segment register. The use of ORG makes assembly start at byte 256 (100h). This leaves room for the DOS Program Segment Prefix, which is loaded into memory at run time. Part 2 is the actual code, in this case just a "no-op."

Part 3 is the segment of the code which terminates the TSR, leaving it resident. In this case, I've used INT 27h, which causes the program to exit withoutretumingallthememoryused to the system pool. Using INT 27h instead of INT 21h Function 31h forces us to a maximum of 64 kilobytes of memory. If we had used INT 21h, we could specify any amount of memory instead of being restricted to 64K. Using INT 27h requires only the passing of a pointer to our TSR while INT 21h would require setting register DX to the number of paragraphs of memory reserved (in 16-byte chunks) with a return code sent in AL. Remember, memory overhead isof paramount importance when it "sticks" in your system executable space. Minimizing thedrainonyourprocessingresources is therefore of absolute importance. I like using INT 27h for short applications where the .COM type executable is used. It is faster and more streamline. Keep in mind, INT 27h should only be called from .COM files since the load allocation of .EXE files must be managed by the programmer.

Of course, the code described above does nothing but take up memory. As a matter of fact, if you placed code into part 2 (start :)it would only execute once and then remain as the proverbial system memory hog. So it behooves us to define an application function and then slightly reconstruct the above code to make it a true resident. Let's construct a very simple but basic utility that's truly useful.

```
;
; Suppose we want to reset the CTRL-C interrupt vector.
;
; To get the CTRL-C vector,
;
    mov   ah,35h        ; set get vector function
    mov   al,23h        ; define CTRL-C function
    int   21h           ; call DOS
                        ; results in es:bx = seqment:offset
    mov   old_seg,es    ; save the segment address to old_seq
    mov   old_off,bx    ; save the offset address to old_off
;
; and to set a new CTRL-C vector,
;
    mov   ah,25h        ; set vector function
    mov   al,23h        ; define CTRL-C function
    mov   dx,new_seq    ; copy new segment address to dx
    mov   ds,dx         ; mov dx to ds
    mov   dx,new_off    ; copy new offset address  to dx
    int   21            ; call DOS
```

Listing P-Using INT 21 to *remap a keyboard input is relatively straightforward.*

How many times have you wished you had a simple procedure enabling you to use those wonderful function keys? To be able to program them with a set of customized instructions that would execute, at the touch of a key, *n* other words, expanding their capability. Let's create a TSR that acts as a basic key expander, so that when DOS or an application wants a character from the keyboard (usually calling `INT 16h`) it will activate a piece of our code which has been placed between that application and the generic operating system. In effect, we will insert a layer of code, under our control, in between the keyboard interrupt and its associated handler. To do this, we must examine how the keyboard handles input.

```
C:\> mem /program | more          . . mem execution command
                                  .. results
Address    Name       Size       Type
000000                000400       Interrupt  Vectors
000400                000100       ROM Communication

-------------------------------------------------------

C:\> debug                        . . DEBUG execution command
                                  .. debug command prompt
-d0:0                             .. command dump starting at 0:0

0000:0000 2B 68.                  .. word output
```

Listing 3— MSDOS supplies *two* tools, *MEM* and *DEBUG*, that can help software development by showing exactly what's going on in memory.

INTERRUPT VECTORS AND THE KEYBOARD

The most common activation of a TSR takes place through the keyboard interface with thecomputer. You could attach a TSR to the timer interrupt and activate the TSR's operation every few seconds. But more often it is attached to the keyboard service interrupt and executes whenever a certain keystroke is pressed. The best employable procedure is to change the keyboard interrupt vector **address** (the pointer that tells the system where to go when a key is pressed). Substitute a vector to your own code and then test each key as it is entered. If it's the keystroke you want, then execute your desired operation and pass control back to the computer. Otherwise, pass control on to the original keyboard interrupt vector address. Simple, right?

System services are linked to specific interrupt handlers through a table of interrupt vectors. These vectorscan be changed at will without affecting application programs that call them. This table is stored within the bottom 1,024 bytes of your system memory using four bytes per interrupt. This makes a maximum of 256 distinct interrupt vectors with each of these 4-byte entries being composed of the segment number and offset of the interrupt handler for that function or, in some cases, the address of a table.

There are two ways to modify interrupt vectors. You can set an inter-

```
;-------------------------------------------------------------
; goto keyboard interrupt vector and read it
;
          TITLE    addr
prgm      segment                  ; SECTION 1
          assume   cs:prgm,ds:prgm
          org      100h
start:
          jmp      begin           ; jump over data

keyboard  dw       ?
          dw       ?
hextab    db       '0123456789ABCDEF',0
message1  db       'Keyboard Interrupt Vector Address: '
lmessage1 EQU      $- message1     ; length
message2  db       ' segment:offset, location  058-05BH'
lmessage2 EQU      $- message2
;
; SECTION 2

begin:
          mov      bx,1            ; load 1 - file handle
                                   ; for standard output
          mov      cx,lmessage1    ; load message length
          mov      dx,offset message1 ; load message address
          mov      ah,40h          ; load number for DOS write
          int      21h             ; call DOS
;
; SECTION 3

          mov      bx,cs           ; make ds point to segment
          mov      ds,bx
          mov      al,16h          ; set interrupt number
          mov      ah,35h          ; set read interrupt func
          int      21h             ; call DOS
          mov      keyboard[0], bx ; keyboard vector offset
          mov      keyboard[2],es  ; keyboard vector segment
;
; SECTION 4
; set up tor output
;
          mov      dx,keyboard[21  ; segment
          push     dx              ; save dx
          mov      dl,dh           ; shift high byte down
          call     sbyte           ; output high byte to mon
          pop      dx              ; restore dx
          call     sbyte           ; output low byte to mon
;
; separate with a colon
;
          mov      al,':'          ; load in character ":"
          call     pchar           ; print ":" to monitor
          mov      dx,keyboard[0]  ; offset
          push     dx
          mov      dl,dh
          call     sbyte
          pop      dx
          call     sbyte
;
; SECTION 5

          mov      bx,1            ; send second message
          mov      cx,lmessage2
          mov      dx,offset message2
          mov      ah,40h
          int      21h
          mov      ax,4C00h        ; end procedure
          int      21h


sbyte     proc     near
          push     ax
          push     dx
          push     si
          push     dx
          push     cx
          mov      cl,4
          shr      dx,cl           ; shift register right
          and      dx,000Fh        ; bitwise logical AND
          mov      si,dx
          mov      al,hextab[si]   ; get hex char
          call     pchar           ; print character
          pop      cx
```

(continued)

listing 4—*ADDR.ASM is* ***a simple utility that demonstrates the 'get interrupt vector' function 35h INT 21h and several of the general INT 21h services.***

```
            pop     dx
            and     dx,000Fh
            mov     si,dx
            mov     al,hextab[si]
            call    pchar
            pop     si
            pop     dx
            pop     ax
            ret
sbyte       endp
;
;
pchar       proc    near
            push    ax
            push    bx
            mov     bh,l
            mov     ah,0Eh          ; set write char in
                                    ;  TTY mode (video)
            int     10h             ; write char and advance
                                    ;  1 cursor position. AL =
                                    ;  character
            pop     bx
            pop     ax
            ret
pchar       endp
;
prgm        ENDS
            END     start
```

listing 4—*continued*

rupt vector directly by changing the contents of its memory location, or you can call the DOS service designed to set them. I prefer the latter since system-generated interrupts may interfere with your process during the change procedure. For example, suppose we employ "`mov word ptr es:24,offset keyboard`" to install a handler offset. If a key was typed "exactly" as this code was executing, the keyboard interrupt address would be meaningless and cause a machine crash. Turning off all interrupts with the "*CLI*" instruction would fix this problem, but CLI can't suspend nonmaskable interrupts. There are ways to handle even the NMIs, but why bother to rewrite DOS? And, although direct modification may work now, it definitely won't work on a multitasking system. The DOS INT 21h service handler, however, provides a safe way to change the interrupt vectors using functions 25h (set interrupt vector) and 35h (get interrupt vector). An example of how this DOS service can be applied is shown in Listing 2.

If you want to examine this interrupt vector structure first hand, this can be accomplished by using two utilities provided within DOS (at least within 4.01). These are the "MEM" and "DEBUG" functions. MEM can be used to reassure yourself that DOS does indeed contain an interrupt vector table within the first 1024 bytes of your 640K DOS memory map. To examine your base DOS memory allocation, follow the sequence shown in Listing 3.

Several high-level languages provide functions similar to the "get and set" functions described above. Turbo C provides the getvect and setvect procedures which do the same thing without actually calling the DOS functions. Microsoft C 5.0 includes the dos_getvect and dos_setvect functions. And, Turbo Pascal 4.0 incorporates the function GetIntVec and SetIntVec. To start writing our planned TSR, let's briefly examine a short utility, shown in Listing 4. Here we use the DOS 35h INT 21h to retrieve the 16h (keyboard) interrupt vector and write the results to the monitor.

Section1 is used to define the code segment and set the location counter so that ADDR will start code assembly at byte 257 in order to leave room for the DOS segment prefix. The code's data structures are then defined. These include the DD (double word) keyboard storage location where we will put the keyboard interrupt vector, two messages for output to the video monitor, and the hex character table hextab. Section 1 then passes control to the procedure BEGIN.

In Section 2 the first message is sent to the monitor. This is achieved by first loading the length and then the message pointer to message1 into

```
if (the hot-key has been pressed){
    discard keystroke
    check DOS - since DOS is not reentrant - so use 34H, 21H
    if (in DOS) {
        set a hot-key flag
        return from interrupt
    }
    else {
        activate the TSR
        when the TSR is completed, return to the calling system
    }
    else {
        move to the next handler on INT 09H
    }
```

Listing *5-A procedure monitors keyboard activity by keeping track of INT 09h.*

		34	ALT-G	67	F9	100	CONTROL-F7
2		35	ALT-H	66	F10	101	CONTROL-F8
3	Pseudo-NULL	36	ALT-J	69		102	CONTROL-F9
4		37	ALT-K	70		103	CONTROL-F10
5		36	ALT-L	71	Home	104	ALT-F 1
6		39		72	UpArrow	105	ALT-F2
7		40		73	PgUp	106	ALT-F3
8		41		74		107	ALT-F4
9		42		75	LeftArrow	106	ALT-F5
10		43		76		109	ALT-F6
11		44	ALT-Z	77	RightArrow	110	ALT-F7
12		45	ALT-X	78		111	ALT-F8
13		46	ALT-C	79	End	112	ALT-F9
14		47	ALT-V	80	DownArrow	113	ALT-F10
15	SHIFT-TAB	48	ALT-B	81	PgDn	114	CONTROL-PrtSc
16	ALT-Q	49	ALT-N	82	Insert	115	CONTROL-LeftArrow
17	ALT-W	50	ALT-M	83	Delete	116	CONTROL-RightArrow
16	ALT-E	51		04	SHIFT-F1	117	CONTROL-End
19	ALT-R	52		85	SHIFT-F2	118	CONTROL-PgDn
20	ALT-T	53		86	SHIFT-F3	119	CONTROL-Home
21	ALT-Y	54		87	SHIFT-F4	120	ALT-1
22	ALT-U	55		88	SHIFT-F5	121	ALT-2
23	ALT-I	56		89	SHIFT-F6	122	ALT-3
24	ALT-0	57		90	SHIFT-F7	123	ALT-4
25	ALT-P	58		91	SHIFT-F8	124	ALT-5
26		59	FI	92	SHIFT-F9	125	ALT-6
27		60	F2	93	SHIFT-F1 0	126	ALT-7
26		61	F3	94	CONTROL-F1	127	ALT-6
29		62	F4	95	CONTROL-F2	128	ALT-9
30	ALT-A	63	F5	96	CONTROL-F3	129	ALT-0
31	ALT-S	64	F6	97	CONTROL-F4	130	ALT-Hyphen
32	ALT-D	65	F7	98	CONTROL-F5	131	ALT- =
33	ALT-F	66	F6	99	CONTROL-F6	132	CONTROL-PgUp

Table 1 — *The expanded character set*

```
;
; SECTION 1

            TITLE   expand
_TSR        segment
            assume cs: _TSR,ds: _TSR ; selects cs to be the
                                     ; default segment, assigns
                                     ; default values for seg reg
            org     100h             ; set location cntr to 256 bytes
start:      jmp     init             ; transfer execution
                                     ; unconditionally to init
old kb vec  dd      ?
current     dw      0                ; pointer to where we're located
                                     ; within the expansion string
                                     ; set to 0  if=0 no expansion
                                     ; if not 0 then expansion
string      db      'dir',0dh,0      ; defined string
            assume  ds:nothing       ; cancels current seg selection
;
; SECTION 2
;-------------------------------------------------------------------
new_kb_vec  proc    far
;-------------------------------------------------------------------
            sti                      ; set int flag on (recognizes)
                                     ;
                                     ; if (a READ request ?){
                                     ;    goto read char
                                     ;} else {
                                     ;    if (a STATUS request ?){
                                     ;       goto status
                                     ; } else {
                                     ;    let original routine finish
                                     ; }
            mov     al,0
            mov     dx,ax
            cmp     ah,0             ; to detect whether operation Oh
                                     ; conditional jump
                                     ; INT 16h,keyboard
                                     ;  Oh read a char from input que
                                     ;  1h test char ready
                                     ;  ah = scan code
                                     ;  al = char code
                                     ;  2h which shift keys depressed
                                     ; je = unsigned compare
            je      read_req         ; if ah=0 then do read_req
            cmp     ah,1             ; if ah=1 then do status_req
            je      status_req
            jmp     old_kb_vec ; do remaining subfunction
read_req:
            call    keyread          ; call a procedure, after ret,
                                     ; continues following call
            iret                     ; returns control from an int
                                     ; procedure to the interrupted
code status req:
            call    keystat          ; get status
            ret     2                ;
;
;-------------------------------------------------------------------
new_kb_vec  endp
;
;SECTION 3
;-------------------------------------------------------------------
keyread     proc    near             ; read a character
;-------------------------------------------------------------------
            cmp     cs:current,0; input queue. If current NOT 0
            jne     expandchar  ; then expansion is being done
;
readchar:
            mov     cs: current.0
            pushf
            call    old_kb_vec ; get keyboard status
            cmp     al,0             ; if al == 0 goto extended
            je      extended
readdone:
            ret
extended:
            cmp     ah,59            ; if the key is not = F1 (59)
            jne     readdone         ; return
            mov     cs:current,offset string ; offset of string
expandchar:
            push    si               ; expand F1
            mov     si,cs:current
            mov     al,cs:[si]
            inc     cs:current
            pop     si                                   (continued)
```

listing **6-The** *"EXPAND"* **keystroke** *TSR.*

```
            cmp     al,0        ; check for end of string
            je      readchar    ;  if YES read from keyboard
            ret
;-------------------------------------------------------------
keyread     endp
;
; SECTION 4
;-------------------------------------------------------------
;
;  if F1 expansion is NOT happening, then the status routine
;   simply calls the old keyboard routine to determine the
;   state of the keyboard input queue
;
;  if expanding F1 then ZF is cleared indicating that a
;   character is available
;
;  if the F1 expansion is in progress, then return a fake
;   status ZF=0, suggesting that a character is ready to be
;   read.  If NOT, then return the actual status
;
keystat     proc    near
;-------------------------------------------------------------
            cmp     cs:current,0
            jne     fakestat
            pushf               ; original routine
            call    old_kb_vec ;  get keyboard status
            ret
fakestat:
            mov     bx,1        ;set a "character ready"
            cmp     bx,0        ; by clearing ZF
            ret
;-------------------------------------------------------------
keystat     endp
;
; SECTION 5
;-------------------------------------------------------------
init:
;-------------------------------------------------------------
            assume cs: TSR,ds: _TSR
            mov     bx,cs       ; move cs into bx,  req to req
            mov     ds,bx       ; move bx into ds,
            mov     al,16h      ;  interrupt number (keyboard)
            mov     ah,35h      ; determine address of int
            int     21h         ; execute DOS function
            mov     old_kb_vec[0],bx ; ES:BX = interrupt vect
            mov     old_kb_vec[2],es ; at loc 058h-05Bh
                                ; es = seg, bx = offset
                                ; replace int vector for ROM
                                ; call with pointer to ours
                                ; set interrupt vector
                                ; ds:dx = ptr to interrupt
                                ; handler.. ds contains cs
            mov     dx,offset new_kb_vec ;offset, returns the
                                ; offset addr of expression
                                ; load offset of variable/
                                ; label
            mov     al,16h      ; keyboard interrupt vector
            mov     ah,25h      ; set new
            int     21h
            mov     dx,offset init ;load offset of init label
            int     27h         ; terminate and stay resident
                                ; dx = offset of last byte+1
                                ; of memory to remain
                                ; cs = seq of memory to stay
 TSR        ends
            end     start
```

listing 6—*continued*

registers CX and DS:DX respectively and then calling the DOS INT 21h function 40h. This is a DOS universal function designed to write a stream of bytes to a file or device. Section 3 then calls to the "get interrupt vector' function 35h and loads the keyboard interrupt vector (BS:ES) into the double word (32-bit) keyboard.

Section 4 outputs the keyboard interrupt address to the monitor. Each byte in the 4-byte address is tested (using sbyte or "show byte") and its equivalent character (using hextab) is thendisplayed (with pchar). Once the address output is completed, a final message is sent to the monitor and the program is terminated using the 4Ch INT 21h DOS exit function.

Keep in mind many of the available DOS interrupts are documented and some are not, and there is some

address space allocated "just" for application user areas. Don't be surprised if some of your applications develop conflicts when running together. Each application designer allocates these spaces as the spirit moves him. I recently built a device driver that used the last available address location within the vector table. I'm keeping my fingers crossed...

HOT-KEY PROGRAMMING

If you are programming an active TSR you will probably use the keyboard interrupt (INT 09h) as a way to initiate your TSR's control of the system. By monitoring what happens at the keyboard, your TSR can tell when a hot key has been pressed and then take control. An outline of this basic procedure is shown in Listing 5.

The INT 09h interrupt allows a certain amount of type-ahead lead time. Whenever a key is pressed, the character is read and placed into an input queue which ROM interrupts examine. The above procedure is successful because of this decoupling of the interface between the keyboard and queue. As such, it permits greater flexibility within the interface between the keyboard and applications.

OUR TSR: A KEYSTROKE EXPANDER

We should now have enough information about TSRs and interrupt handling to build a simple resident application. We'll build our interrupt routine like any general procedure using the PROC and ENDP directives. The only differences are that the new routines will always be defined as FAR and they will be terminated by an IRET instructioninstead of the regular RET instruction. Our procedure will replace a single keystroke, in this case F1 (059), with a series of characters. Its design is to eliminate the need to retype the same commands over and over. You can of course modify the code to handleanyof the 132 expanded keys shown in Table 1, or to contain a table of multiple key redefinitions.

The "key expander" we will be discussing is shown in Listing6. It has been broken into five sections to facilitate design parameters.

I'll not bore you by trying to describe each line in the listing. I think I've put enough comments within the listing to fill in most of the important steps. Instead, I'd like to just add a few brief **summary comments** here. Within Section 1, the program segment is defined and coding again starts at 0100h to leave room for the DOS Segment Prefix. Control execution then transfers unconditionally to INIT for initializationandsettinguptheproper interrupt vector addressing scheme. The .DATA declarations are also contained within this section. It consists of three different entries. The first is the double word, old_kb_vec (32 bits) which is designed to hold the old keyboard interrupt vector. The second is the expansion string pointer, current.Thispointerisofparamount importance to our routine. It points to the next character to be returned to DOS. If it is zero, then no expansion

takes place. If it is not zero then that character is returned to DOS. However, if that character is a zero byte (a character string terminator), then the expansion is discontinued and a new character from the keyboard is required. current is also used by the key status and the key read routines. It's used to determine the state of current operations. And finally, the expansion st ring is defined. For our purposes, it is defined to be "dir" followed by a carriage return (0dh).

Section 2 is in many ways the "decision-making brain" of the utility, but Section 5 is its "heart and soul." Here our code becomes a truly memory resident program. The interrupt vector (16h) of the keyboard is retrieved by a call to INT 2 1h function 35h and stored within old_kb_vec. The offset address is stored in the first two bytes ([0]) and the segment address is stored in the last two bytes ([2]). The address within theinterrupt descriptor table is then replaced with the address of our new kb routine. This is achieved by a call to DOS INT 21h function 25h. A pointer to our procedure is then loaded and a call to DOS INT 27 h places our program in a terminate-and-stay-resident mode. Now, each time a key is pressed, new_kb will capture it and examine it to determine if it is the F1 key. If not, it will allow the system to continue its procedure. If it is F1, then it will discard that strokeand put our predefined st ring into the keyboard queue.

Sections 2, 3, and 4 are used to determine how our inserted code will react. Polling of the results of INT 16h to see whether the keyboard interrupt is processing a read (AH=0h) request or a status (01h) request is done each time an interrupt is generated. If a status check is requested, we need to remember that the INT 16h 001h "test for characterready" doesn't return via the IRET since it must pass back ZF (zero flag) as an indicator. It does a quick check of the keyboard and returns immediately. If a keystroke is ready, the zero flag is cleared and it returns the keystroke's ASCII code and keyboard scan code. If there is no keystroke to process, a value of 0 is returned for the ASCII code.

So for ZF = queue, set means empty and clear means a character is available. This means that for a status ZFresponse, we are required to return ZF to the value it contained before the interrupt. This is done by using the RET feature for specifying the number of bytes to pop off the stack. In this case we need to flush the altered flags from the stack so that the original set of flags will be returned instead.

In Section 3 a valid keystroke is tested to determine if it is our defined "expand" key F1. If it is, then the contents of the queue are stuffed with our string. But also keep in mind that the character input routine returns its results in the low byte of AX. However, if an extended key like our function key is typed, then a two-byte sequence is returned. AL is set to zero and AH will contain the extended character code. If this code is 59 (F1), then we will expand the key. This is done by fetching the byte defined by the current pointer and puting it in

the queue. If AL becomes zero, we've reached the end of our expansion string and we transfer control to the keyboard to fetch another character.

ONWARD AND UPWARD

Serious assembly language programmers cringe at the thought of what I'm going to suggest next, but you can create a programlike **EXPAND** by first writing it in C and then compiling it with the "output assembly source code" option set. Then just edit and streamline the assembly output. I often find myself doing this when I want to optimize a program for speed and loading time. And if I'm honest I'll admit that I'm not that great with assembler. At least this way I get everything there in some semblance of order; it's my editing that screws everything up. But isn't that the fun of programming, biting your nails, guzzling the coffee during those late nights, and the not so occasionally screaming and cursing in despair as you reboot your system for the thousandth time?

REFERENCES:

1. Dettmann, T.R., 'DOS Programmer's Reference.' Que Corp., Carmel, Indiana, 1988.
2. Wyatt, A., 'Using Assembly Language.' Que Corp., Carmel, Indiana, 1987.
3. Norton, Peter, "Inside the IBM PC.' Brady Communications Co., Inc.. NY, 1986.
4. Wadlow, T.A., 'Memory Resident Programming.' Addison-Wesley, NY, 1987.
5. Murray, W.H. and Pappas, C.H., "80386/80286 Assembly Language Programmina.' McGraw-Hill, Berkeley, CA.? 986.

Chris Ciarcia has a Ph.D. in experimental physics and is currently working as a staff physicist at a national lab. He has extensive experience in computer modeling of experimental systems, image processing, and artificial intelligence. Chris is also a principal in Tardis Systems.

IRS

404 Very Useful
405 Moderately Useful
406 Not Useful

FEATURE ARTICLE

Jim Butler

A Simple RS-485 Network:

Exploit the Nine-Bit Serial Communication Modes of the 8051, 8096BH, 68HC71, 68HC05, and 2780 Microcontroller Families

As often happens, a project gets started because existing products were not adequate; such was the case with our RS485 network project. A couple of years ago, two of us wanted to acquire **data** from several industrial Programmable Logic Controllers (PLCs) which were being used for process control. These PLCs used 8052 microcontrollers. We purchased a 9600-bps RS485 network from the manufacturer of the PLCs, but the system was not satisfactory. The main problem was that running the network required a significant number of CPU cycles from the processcontrol PLCs: the installation of the network caused the PLC scan rate to drop by over 50%; in other words, network overhead consumed over half of each PLC's time!

Based on that experience, we decided to develop a simple network for limited data acquisition and control which could connect a PC and several embedded controllers. Our design had to have the following characteristics:

*usable in an industrial environment

•low resource requirements from the embedded controllers, both in terms of CPU time and memory

•low cost per node

•adaptable to popular microcontrollers

•can be installed in existing embedded controllers with little modification to the hardware and control software

We have since named thenetwork protocol tiny-NSP (Nine-bit Serial Protocol).

NETWORK HARDWARE—THE PHYSICAL LAYER

We chose shielded twisted-pair RS485 because of its low cost and simplicity. Our data requirements were relatively low, so a network with high bandwidth over long distances was not necessary.

RS-485 allows up to 32 transmitters and 32 receivers to be connected by a single pair of wires; thus, up to 32 network nodes are possible in the simplest configuration. Only one transmitter can be active at a time on any given pair. Shielded twisted-pair wiring does not pick up much noise, important in an industrial environment. The maximum bandwidth depends on the length of the twisted pair. (For a discussion of RS485, see references [1] and [2].)

Our network topology is simple: a single twisted-pair of wires goes from node to node, connecting all of them. The pair is terminated by a 120-ohm resistor at each end. When any transmitter is active, the data it sends is received by all receivers on the pair. This topology is often referred to as a bus or multidrop network.

An embedded controller is interfaced to the twisted pair using an RS-485 receiver and transmitter. Since our network uses a single twisted pair, we often use an RS485 transceiver such as the 75176.

In the process of designing our hardware, we had to contend with a little-discussed problem with RS485. Since there will be times when no transmitters are active, we need to ensure that the RS-485 receivers will indicate to the CPUs that there is no data to receive. In order to do this, we must hold the twisted pair in the mark (off) state by applying a differential bias to the lines. Otherwise, random noise on the pair could be falsely interpreted as data.

PROTOCOL FRAME AND MESSAGE FORMAT

Because of our desire to support existing embedded controllers with minimum hardware modification, we chose asynchronous serial communication for our network. In addition, we designed the data format to match the capabilities of the UARTs built into several popular embedded controllers, so in many cases, the only hardware enhancement required in order to turn an embedded controller into a network node is the addition of an RS485 transceiver (see Figure 1).

By designing the frame and message formats so as to exploit certain features of popular microcontrollers, we **can** significantly **reduce** the amount of time that the embedded controller spends dealing with the network. For best performance, a particular embedded controller should only be interrupted when it is receiving a message it should act upon or when it is transmitting a message.

Microcontroller families such as the 8051, 8096BH, 68HC05, 68HC11, and 2180 all have a nine-data-bit serial asynchronous communication mode and a "wake-up" mode, which together were specifically designed for interprocessor communication. (For more information on these mi-

crocontrollers, see 141, [5], [6], [7], and [8].) These modes suggest a network protocol in which data is packaged into messages, each of which consists of a series of frames.

Certain frame and message formats make best use of the nine-data-bit and wake-up modes. There are two kinds of frames (address frames and data frames), each of which has a start bit, nine data bits, and a stop bit. Each message consists of one address frame followed by one or more data frames. Address frames and data frames are distinguished by the ninth data bit, which is 1 for address frames and 0 for data frames. The other eight bits of an address frame specify the address of the nodewhichshouldact on the message. Data frames contain commands, status information, error control, or data, as needed by the application.

When the wake-up mode is enabled, the microcontroller's UART will ignoredata frames, but will generate an interrupt when an address frame is received. This is how we use it: Upon initialization, the embedded controller goes into nine-data-bit mode with wake-up enabled. The microcontroller's UART will not trigger an interrupt until an address frame is received, signaling the beginning of a message. When an address frame is received, a node's network code examines the lower eight bits, which represent the address of the target node. If that address matches the node's address, then the code will disable wake-up in order to allow the reception of the data frames that follow, then reenable wake-up after the entire message has been received. Otherwise, wake-up will remain on, so that the remainder of the message will be ignored; in fact, the microcontroller will not be interrupted by the UART until the next message begins. All of this means that when wake-up mode is used, each embedded controller will be interrupted at the beginning of each message, but the remainder of the message can be ignored if appropriate.

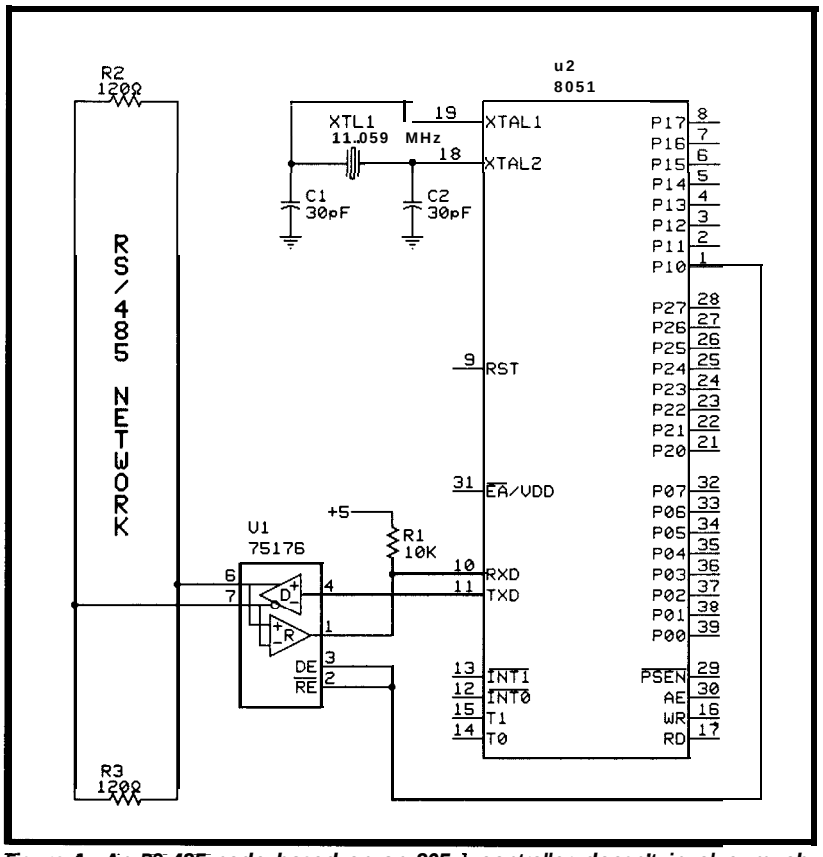


Figure 1 *-An RS-485 node based on an 805 1 controller doesn't involve much more than the addition of a transceiver to the processor.*

See the sidebar for information on the 83C51FA microcontroller, which is especially well suited for nine-bit asynchronous serial communication.

NETWORK ACCESS CONTROL

Everynetworkprotocolmusthave a method of allocating available bandwidth to the various nodes. When RS-485 is used, only one node should be transmitting at any given time. Of course, we want each node to have the opportunity to transmit messages every once in a while.

Keeping in mind that we wanted the embedded controllers which are actually doing useful work to have to do as little communication processing as possible, it made sense to use a simple polled master-slave scheme in which the master node is dedicated to controlling the network, thus allowing the slaves to concentrate on process control or whatever. Periodically, themasterpollseach slave to get status information, collect data, or issue control commands. In our current protocol, slaves only respond to commands from themaster; thus, the protocol is an order/reply message protocol.

Because data quantity requirements were modest in the applications we were targeting, a polled master-slave scheme is adequate despite the fact that it is not a particularly efficient method for utilizing network bandwidth. Polled protocols also have the disadvantage of preventing slaves from transmitting datauntil they are addressed, but this is only important if the data must get to the master very quickly.

Alternative protocols which do not have these two drawbacks, such as CSMA/CD (used in Ethernet), are usually significantly more complicated, and thus require more communication processing time and **hardware resources.** For example, in most contention network protocols (e.g., CSMA/CD), all nodes are required to have a timer for collision recovery and time-out detection, whereaspollednetworksusuallyonly require the master **node** to have a timer (for timeout detection).

In our prototype network, the master node is a PC and the slave nodes are embedded controllers. The PC is a good choice for the network master when data must be collected or when a system must be monitored and controlled by a person. PC hard-

ware requirements are discussed in a later section.

MESSAGE STRUCTURE

The message structure design was heavily influenced by our desire to minimize the slaves' need for computational resources. At the same time, we wanted to have some confidence that corrupted messages could be detected. The structure we are currently using is shown in Figure 2.

The master node always has address 0. Slave addresses can range from 1 to 254. The master may broadcast a command to all slaves by using address 255.

Messagescan be anywhere from5 to 255 bytes long, depending on the operation. Each node must have enough RAM to buffer the entire message so that error checking can occur before themessage isacted upon. Since some slave embedded controllers may not have enough RAM to buffer long messages, they may choose to restrict the maximum message length that they will handle.

Error checking is done using an eight-bit checksum or CRC. We chose eight-bit error checking over sixteen-bit checking because the eight-bit microcontrollers we primarily use can do the necessary computations much faster. Checksum is actually rather weak (it is only guaranteed to detect a single bit-error in a message), but it is fast and requires little memory. By using eight-bit CRC, it is much more likely that multiple bit-errors will be detected (particularly in messages of 15 bytes or fewer); however, CRC computation requires significantly more time (XOR and shift method) or more memory (table lookup method) than checksum computation. A 16-bit CRC is often used in commercial communication protocols. For more information on CRCs, see [11] and [12].

The address of the source node is not actually necessary for our current protocol. This address could be useful if we were to allow slaves to communicate directly with each other.

For examples of other message formats and network access control methods, see [1], [9], and [10].

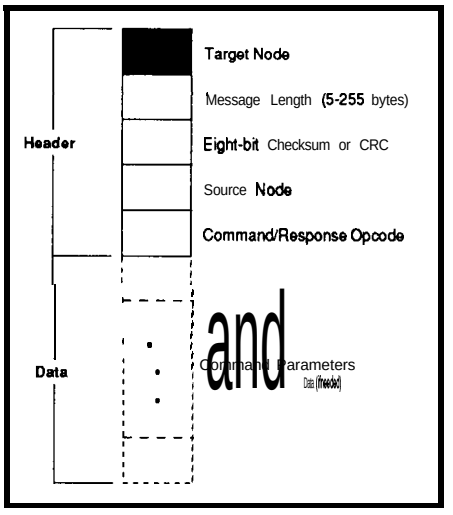


Figure 2—*The* tiny-NSP ***message*** structure ***includes source and targetnoae, message length, opcode, checksum, and optional data.***

MESSAGES

We have started with a small set of commands and responses, which are listed in Figure 3. The master expects a prompt response to all messages except broadcast messages.

Slaves must at least support the "status query" and "reset network" commands. If a slave receives an unsupported command, it responds with a "command not supported" message.

Slaves may support a particular command while not allowing certain kinds of access (e.g., master may not be allowed to write to some memory locations). The "accessviolation" reply is useful in those cases.

The main limitation of the core group of data transfer commands is that themaster must know the **precise** memory or I/O port address of a desired piece of data. For a specific application, commands can be added which refer to data objects symbolically instead of by physical address.

ERROR RECOVERY

All network protocols must include procedures for handling errors. Here are some of the circumstances which can cause network errors:

*the addressed slave cannot handle a specific command

•the addressed slave is busy doing something which temporarily prevents the slave from responding to a command

*the addressed slave is not connected to the network

•hardware has malfunctioned

• noise was picked up by the transmission medium, corrupting a message

In our protocol, error recovery is the responsibility of the master. This helps to minimize the computational resources required of the slave. Here are four error conditions that the master could detect:

•a valid response from a slave indicates that the slave could not execute a command

•a slave response is invalid

•a slave did not respond to a message within a certain period of time (detection requires a timer)

•a slave transmitted a message without being polled

Master Commands	Slave Responses
reset network [B]	my status (includes maximum message length handled) [D]
status query	acknowledge (response to write command)
write to slave internal RAM [PD]	data (response to read command) [D]
write to slave external RAM [PD]	
write to slave I/O port [PD]	command not supported
read from slave internal RAM [P]	parameters out of range
read from slave external RAM [P]	access violation
read from slave I/O port [P]	message is too long to buffer

Figure 3—*This is the current tiny-NSP command and response set. All nonbroadcast commands should trigger a slave response.*

[B] - broadcast command
[D] - message includes data
[P] - message includes command parameters

If the first condition occurs (the slave could not execute the command), then the master will report the error to the operator. If any other error condition is detected by the master following the transmission of a message to a slave, then the message will be retransmitted up to a fixed number of times; if the master does not receive a valid response. then the slave is removed from the master's list of active slaves and the operator is notified. If an error condition is detected at any other time, then the network reset procedure is initiated.

A slave is still responsible for detecting errors in messages addressed to that slave. If such an error is detected, the slave will simply ignore the message. The lack of a slave response (to a nonbroadcast message) should result in a time-out condition, triggering the master's error recovery procedure.

Consider the following scenario: The master sends a command message to a slave, the slave acts upon the command, and the slave replies. If the reply is corrupted, the master will likely retransmit the command to the slave, causing the slave to act upon the command a second time. There are two ways to get around this problem:

- do not use any commands which if executed more than once could cause undesirable behavior (e.g., XOR operations)
- add message numbers to the protocol, allowing the slave to recognize a repeated command; one protocol which uses such a scheme is BITBUS [10]

For now, we have taken the first approach.

The Intel 83651 FA

Intel's 83C51FA subfamily is particularly appropriate for use in a nine-bit serial asynchronous network protocol. Microcontrollers In this family are enhanced versions of the 8051. The good folks at Intel added a feature just for nine-data-bit networking: Automatic Address Recognition, which when enabled prevents the microcontroller from being interrupted by received data until it receives a message addressed to itself. Thus, microcontroller network processing overhead is reduced even more.

It Is possible to program these chips to wake-up when any one of several addressesare received. These controllers also have framing error detection. For more information, see (4) and (8).)

PC HARDWARE REQUIREMENTS

Our nine-bit network was designed to make the best use of the capabilities of popular microcontrollers, but outfitting a PC to be a nine-bit network node requires hardware with specific capabilities. Most RS-485 cards for the PC have 8250, 16450, or 16550 UARTs; these cards are capable of being used in some nine-bit networks, but special programming is required (for example, see [3]). Of those UARTs, the NS16550AF is the best choice for reasons of performance and communica-

References

(1) Nisley, Ed, 'A Network for Distributed Control, Part 1,' Circuit Cellar INK, Aug/Sep 1989. pp. 32-39.

(2) 'Standard for Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems- (EIA-485). Electronic Industries Association, 1983.

(3) Woehr. Jack, 'Multidrop Processing,' Embedded Systems Programming, March 1990, pp. 58-67. Nine-data-blt communication using the 6868 1 DUART.

(4) 8-Bit Embedded Controller Handbook, Intel Corp. 8051 information.

(5) 16-Bit Embedded Controller Handbook, Intel Corp. 8096 Information.

(6) Motorola Microprocessor, Microcontroller and Peripheral Data (Volumes 1 and 2). Motorola, Inc. 68HC05 and 68HC 11 information.

(7) 280 180/Z180 MPU Microprocessor Unit, Zilog Inc., 1989.

(8) Embedded Control Applications, Intel Corp.. 1988. includes 'An Introduction to the Intel MCS-5 1 Single-Chip Microcomputer Family' (AP-69) and 'Enhanced Serial Port on the 83C51FA" (AP-410).

(9) Gandy, Mar-tin, Microcomputer Communications, NCC Publications: Manchester (U.K.), 1985.

(10) The BITBUS Interconnect Serial Control Bus Specifications, Intel Corp., 1988.

(111 Perez, Aram, 'Byte-wise CRC Calculations," IEEE Micro, June 1983.

(12) Ritter, Terry, 'The Great CRC Mystery," Dr. Dobb's Journal, February 1986, pp. 26-34. Includes Pascal listings and a good set of references.

(13) McGraw-Hill's Compilation of Data Communications Standards, Edition II. Includes the ISO basic mode protocol and ANSI standard 3.28-1976.

(141 Gee, K. C. E., Introduction to Local Area Networks, NCC Publications: Manchester (U.K.), 1983.

(151 Hardy, Peter, Introducing Data Communication Protocols, NCC Publications: Manchester (U.K.), 1985. Error recovery procedures.

(161 Flint, David C., The Data Ring Main: An Introduction to Local Area Networks, Wiley, 1983.

(17) McNamara, John E., Technical Aspects of Data Communication, Digital Press, 1988 (third edition).

tion reliability. However, the best UART for nine-bit networking is the 82510.

MASTERS AND SLAVES

By **using the interprocessor asynchronous serial communication features of popular microcontrollers, we have designed a low-cost network appropriate for industrial environments which has low communication processing overhead for slave nodes.** A PC **can be either a master or slave node in such a network.** In the future, we hope to develop other hardware and software that will help engineers work with distributed data collection and control applications. ✣

Jim Butler is a software engineer at Cimetrics *Technology. He received B.S. and M.S. degrees in* engineering **from** *M.I.T.*

IRS

407 Very Useful
408 Moderately Useful
409 Not Useful

FEATURE ARTICLE

Interfacing Microsoft's Flash File System

Using Flash Memory Under MS-DOS

Markus Levy

Back in issue 18, I discussed the design for a flash memory array based upon a PC/AT add-in board employing SIMMs or IC memory cards. Now we are ready to turn your flash memory platform into a DOS-compatible solid-state disk. In this article I'll show you how to interface Microsoft's special Flash File System (FFS) so you can store and retrieve files on this non-volatile solid-state disk.

Before discussing the structural elements and benefits of the FFS, let's review some basic standard MSDOS concepts. This will provide the motivation for implementing a special file system for flash memory.

A LOOK AT STANDARD MS-DOS

At the highest level, applications make requests to the MS-DOS function dispatcher through interrupt 21h. Arguments identifying the service desired and its options are typically passed in registers. To relieve application programs of the necessity of managing disk storage space, MS-DOS provides a file system manager. This series of MS-DOS services keeps track of disk storage using file and directory structures.

All block-device accesses by MS-DOS are made through a standardized device driver. MS-DOS makes requests to a block device by passing a data structure, called a request header, to the device driver. The request header contains the desired command and required arguments and is located by the device driver using a special routine, typically named "Strategy." For a read or write request, the logical sector to start the access and the number of logical sectors to transfer are provided. The device driver performs translations from the requested logical sector to the physical location on the device.

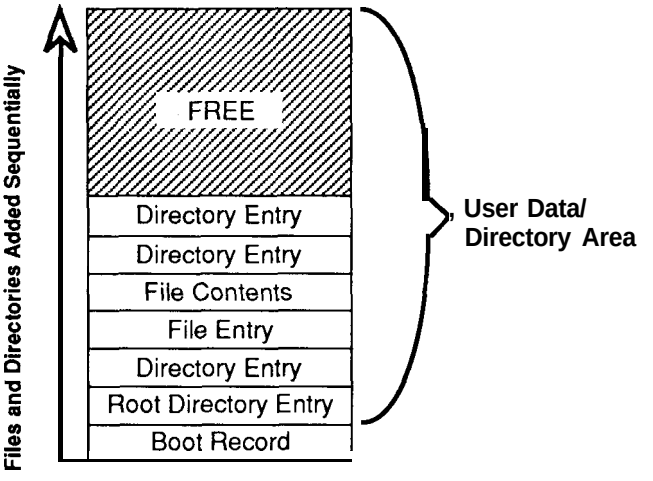


Figure 1 — *Flash* **files are** *located by following linked-list pointers within each file and directory entry.*

For each block device, MS-DOS maintains four areas: a boot record, a File Allocation Table (FAT), a root directory, and a data area. The boot record is the first section on the disk. It contains a structure known as the BIOS Parameter Block (BPB) supplying MS-DOS with information about the disk, including sector size, sectors per cluster, number of FATs, directory size (number of files), and so on.

MS-DOS requires each disk to have a FAT to keep track of sector/cluster allocation. The FAT contains information on all the clusters that are allocated, free or unusable. It is an array of cluster pointers and each entry has a one-to-one correspondence with a cluster on the disk.

Grouping sectors to form clusters increases efficiency in terms of the memory required to manage the FAT. A cluster can consist of a different number of sectors. Four sectors (or 2048 bytes) per cluster is typical on a hard disk. The larger the cluster or allocation unit, the more potential of wasting space for files not sized to a multiple of the number of bytes in a cluster. For instance, a 20-byte file stored on a disk with a cluster size of 2048 bytes, wastes 2028 bytes.

Following the FAT is the root directory. Directory entries consist of the file name, attributes, time, date, file size, initial cluster number, and reserved bytes. When allocating space to a file, the initial cluster number is updated to point to the initial cluster number used by the file. The value at that location in the FAT points to the next cluster used by the file or contains an end-of-file marker. Thus, the allocation chain is a forward linked list. When extending a file and another cluster is required, MS-DOS replaces the end-of-file marker with a pointer to the next cluster, which is set to an end- of-file marker.

An increase in file size, or any type of file revision, results in a byte-alterable modification to the disk di-

rectory. In addition to file size change, modifications include time of last change, date, attributes, and file renaming.

THE NEED FOR AN ALTERNATIVE FILE SYSTEM

Recall that flash memory is a bulk-erase memory. The FAT and directory structures created for the byte-alterable magnetic disk are not ideal for a flash memory solid-state disk (SSD). Updating a FAT or directory entry requires complete erasure of the flash memory components containing the changing bytes. It is possible to implement a flash memory disk based on this approach, but the write latency times are unacceptable for general use. When a file is added, deleted, or modified, the directory could be copied to a RAM buffer and modified to reflect the change. After the flash memory device that contains this directory is erased, the modified directory is copied back.

Disk imaging is another method of implementing the standard MS-DOS FAT scheme on a flash memory SSD. Using this method, files are first copied to a floppy disk. Then a special utility performs a disk copy transferring the FAT, directory, and all files to the flash memory SSD. This approach is useful for building an application cache that is FAT file system compatible, but all flexibility is lost.

Microsoft has developed a special file system utilizing the attributes of flash memory. To minimize fragmentation losses and allow arbitrary extension of files, the flash memory file system uses variable-sized blocks rather than the sector/cluster method of standard MS-DOS file systems. This flexibility is provided by employing a linked-list structure; that is, chaining files together using address pointers located within directory entries for each file.

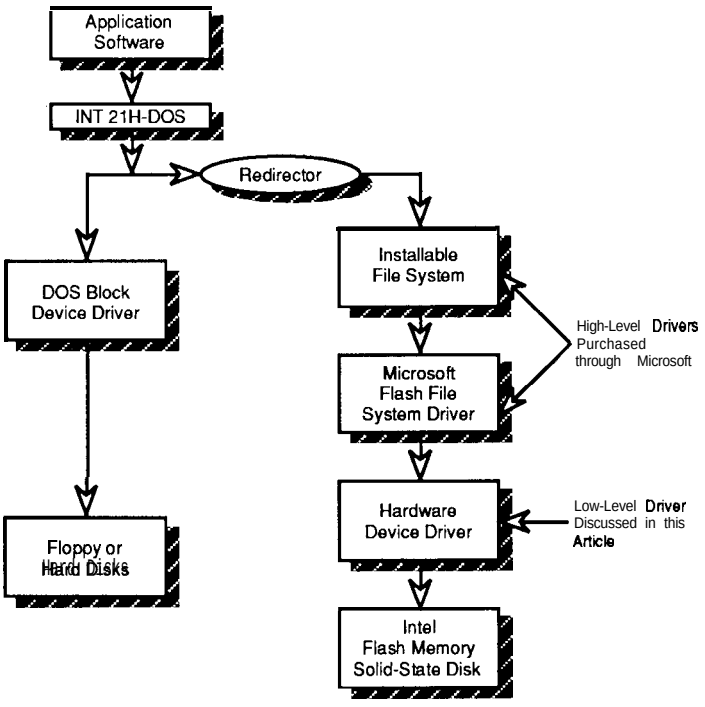


Figure *2-A two-level architecture provides a consistent application interface while allowing for a variety of flash-memory hardware platforms.*

Files and directories are written to the flash memory SSD using sequentially free memory locationsa stack-like operation (Figure 1). When the "stack" is full, the desired files are copied to another disk and the current disk is erased for reuse.

File and subdirectory information is attached to the beginning of each file, unlike the standard MS-DOS approach of directory and FAT placement. As directory and file entries are added, they are located by building a linked list. Besidescontaining the standard fields (e.g., name, extension, time, date of creation), a directory or file entry contains a status byte and various pointers used for the linked-list structure. The status byte, besides indicating whether a file/subdirectory exists or is deleted, also signifies valid sibling and/or child pointers and if a directory entry pertains to a file or a directory.

When a directory or file is requested, the flash memory SSD is searched beginning at the head of the linked list. The chain is followed from pointer to pointer until the correct entry is found. If the search arrives at the chain's end (an FNULL identifier is encountered), the system responds analogously to MS-DOS with a "File not found" message.

When deleted versions of a file appear on the flash memory SSD, the file system finds the most recent version. The status byte contains bit fields that indicate whether a particular file is valid or deleted. The directory information of a deleted file is still used for pointers of the linked list and the search proceeds until it finds the most recent and valid version.

FLASH FILE SYSTEM: ARCHITECTURE OVERVIEW

The Flash File System consists of two components: `IFS .SYS` (Installable File System) and `FEFS .SYS` (Flash EPROM File System). When an application accesses a disk through interrupt 21h, the MS-DOS kernel checks the drive letter. If the drive has been declared as a flash memory SSD, `IFS.SYS` intercepts the request through a proprietary interrupt 2Fh redirector interface and passes it to

Character Device Header

```
DW      Block Header            ; Pointer to Next Device Driver
DW      0
DW      11000000000000008       ; Attributes (CHAR, IOCTL control)
DW      Strategy                ; Offset of Strategy Procedure
DW      Interrupt               ; Offset of Interrupt Procedure
DB      'FIFIDEV£'              ; Device Name Used by FEFS to Locate the LLD
```

Figure J-The *Flash File System doesn't use a FAT and directory structure, but is uses a character device driver which must contain the proper header.*

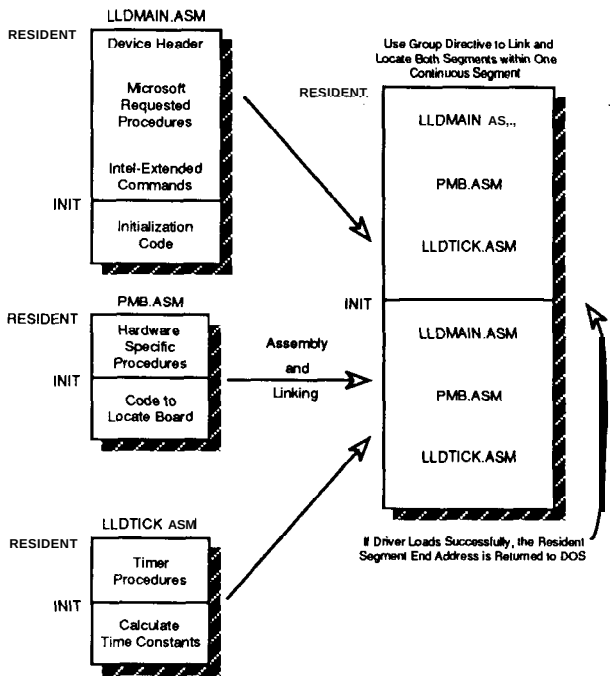


Figure 4—*The INIT portion of the device driver is used only once, at initialization, then 'discarded."*

the FEFS.SYS driver. FEFS.SYS implements the FFS logic, developing and maintaining the linked-list structures.

The Microsoft FFS is implemented as a two-level architecture, where **IFS.SYS** and **FEFS.SYS** represent the high-level driver communicating with a low-level driver (LLD) that is hardware specific (Figure 2). This architecture provides a consistent application program interface while allowing for a variety of flash memory hardware platforms.

The LLD implements a set of device primitives for use by the high-level driver. This is not that different from the FAT file system as we know it. In that environment, MS-DOS implements a high-level, FAT file system driver interacting with a set of device primitives implemented as interrupt 13h.

WRITING THE LOW-LEVEL DRIVER

The remainder of this article will explain how to write this low-level driver providing the bridge between Microsoft's Flash File System and the page-memory board described in the December issue. While I would need forty pages to completely explain all the details of this implementation, there is enough here for you to understand the LLD software that can be obtained from the Circuit Cellar BBS. Also, if you have no intentions of implementing a DOS-compatible SSD, many of the device primitives from the low-level driver canbeextracted for any type of flash memory application.

The LLD consists of three components: an MS-DOS device driver, the procedures called by Microsoft's FFS, and the page-memory board hardware-specific procedures. I have written the LLD in several modules tosimplifyanymodifications necessary to accommodate hardware variations. Themodule LLDMAIN . ASM contains the MS-DOS interface routines, the Microsoft-requested procedures, and special Intel-extended procedures. PMB . ASMcontains procedures specific to the page-memory board hardware, such as setting the page number and turning on V_{pp}. All the functions contained in this module control the I/O functions on the board. LLDTICK . ASM contains procedures associated with the timing requirements of the flash **memory components, as** well as routines used to provide a V_{pp} turn-off delay.

The programming and erase voltage, V_{pp}, is generated on the page-memory board using a DC-DC converter. When this converterisswitched on, it takes anywhere from 20 to 100 milliseconds for V_{pp} to arrive at a stable voltage. This time depends on the amount of capacitive loading and the circuit used, as some have faster start times. If you are designing your hardware for a desktop system, V_{pp} can remain switched on. However, in battery-powered systems, V_{pp} should be switched off when not in use to conserve power. To accommodate these applications, I have written a procedure to generate a V_{pp} turn-off delay. This is similar to that for a floppy disk in that after two seconds of nonuse, V_{pp} is switched off. If several blocks of data are being written consecutively, your software will not have to delay waiting for V_{pp} to ramp up every time a new block is written.

The V_{pp} turn-off delay is calculated by installing an interrupt 1Ch (time-of-day clock) filter. This interrupt is generated every 18.2 milliseconds. Before servicing the original interrupt, our filter increments a count value. When that count value reaches 36, the procedure to turn V_{pp} off is called.

Since the LLD is an installable device driver, it requires a standard

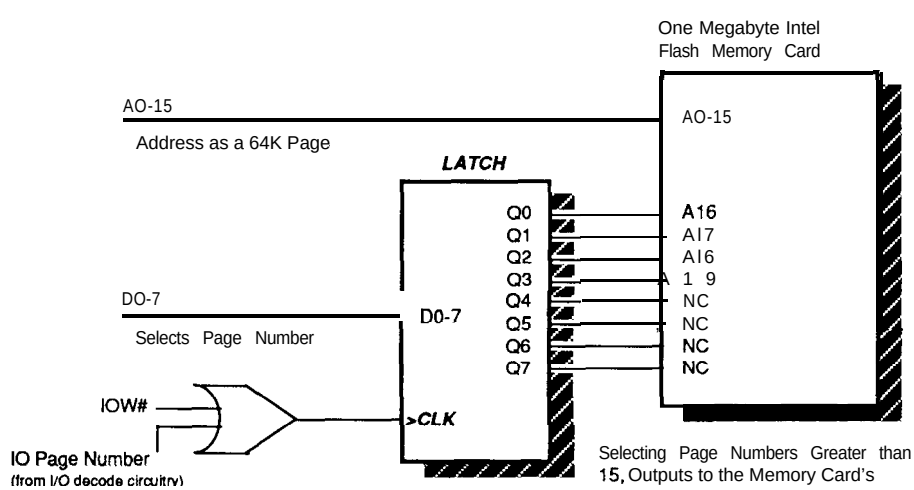


Figure 5—*Device density is dynamically determined, with page numbers beyond the valid size of the device handled by page wrapping.*

MS-DOS–compatible interface that provides a request header and entry points into the Strategy and Interrupt procedures. This portion of code primarily performs the initialization of the device driver.

Unlike the magnetic disk, the FFS does not implement a FAT and directory structure with sectors. Therefore, it is designed as a character device driverratherthanablockdevicedriver and must contain a character device driver header (Figure 3). During installation of the character device, MS-DOS passes a command number in a request header to the Interrupt procedure which dispatches a call to the `InitDriver` routine (command zero). This procedure is performed only once, immediately after the device driver is loaded into memory. The initialization procedure of the character device, `InitDriver`, is primarily responsible for locating the page-memory board, computing the time constants for the flash memory program and erase algorithms, and determining the quantity of flash memory available.

INITIALIZATION

Recall from the page-memory board design that the first four I/O ports are read to obtain the board's signature. A procedure, `SearchPMB`, called by `InitDriver`, reads sequential I/O ports until finding this signature. When the signature is found, the base port address is stored in a memory location to be used for future I/O port access. If the board is not present, the driver aborts its installation and returns the system memory to MS-DOS. This is done by passing an offset of zero back to MS-DOS in the INIT request header. Each of the three modules (`LLDMAIN.ASM`, `PMB.ASM`, `LLDTICK.ASM`) are divided into a RESIDENT and an `INIT` segment. These segments are joined using the group directive to ensure they are linked and loaded consecutively in memory. Using this technique, if the page-memory board is found, the ending offset of the resident portion of the device driver is passed back to MS-DOS (Figure 4). This "throws

away" the INIT portion which is not needed after initialization.

The page-memory board has four sockets for Intel Flash Memory single in-linememory modules (SIMMs). The hardware functions with one or more sockets populated, so during initialization the available memory must be determined. Similarly, with the Intel Flash Memory Card, an interesting situation is faced in accommodating density variations. This is best understood using the memory card as an example.

One-and four-megabyte card densities are available today, but the page-memory board's card socket handles up to 64 megabytes (based on the Personal Computer Memory Card International Association, PCMCIA, specifications). Assume that a one-megabyte card has been installed (although our software doesn't know this yet) containing eight one-megabit components (28F010). The first step in determining the module's density is reading the flash memory device ID from offset zero of page zero to obtain the device size. Comparison against a table of IDs versus densities allows the calculation of the page number needed to access the next component.

Theoretically, this process would continue, adding up the total number of components and multiplying that number by the component size to calculate the total module density. However, the page-memory board responds to setting the page register beyond the valid page range of the flash memory installed (Figure 5). For example, the one-megabyte memory card accommodates sixteen (numbered O-15) 64K-byte pages. Pages 0, 16, 32, and so on, access the same memory location because of the wrap-around phenomenon. This inaccurately determines an infinite module size. How does our software know when to stop? Notice in the code (Listing 1) that the first device (page zero) is left in the READ_ID mode. Looping through the rest of memory, a wrap-around is detected when the device ID is already present in the first location of the device. This condition is used to terminate the loop.

Besidesinitialization,theMS-DOS device driver for FFS supports IOCTL reads and writes. The device driver uses the IOCTL commands to return control information to the program regarding the device. FFS issues the IOCTL commands to get and set the entry point for the low-level driver. Unlike block devices, character devices are located with a file open request. When FEFS.SYS installs,itlinks into the low-level driver by opening the file FIFIDEVE (£ is the British Pound Sterling or IBM extended ASCII 9Ch), and performing an IOCTL read to obtain the pointer to the entry of the low-level driver.

Character devices do not support drive letters. You must use a "pseudo-block" device header during driver installation to reserve DOS letters for use by the FFS (Listing 2). After the character device installs, the block-device driver links the device driver into the device driver chain. Drive letters are established by providing fake BPB information in the block device driver header to pass back to MS-DOS.

```
GetPhyChar PROC NEAR
        ASSUME  CS:PROG,  DS:PROG, ES:NOTHING

        CALL    TurnVppON     ; Turn Vpp ON for READID_CMD
        XOR     AX,AX         ; AX contains page number
        CALL    SetPage       ; Routine to set page

        MOV     ES,FrameSeg   ; Point to page frame segment
        XOR     DI,DI         ; Point to start of device

        MOV     ES:WORD  PTR[DI], (READID_CMD SHL 8) OR READID_CMD

        MOV     AX,ES:[DI]    ; Read manufacturer's ID
        CMP     AX, (INTEL_ID'SHL 8) OR INTEL_ID ; Intel Devices?
        JE      GPC2          ; Yes, continue
GPC1:
        MOV     ES:WORD PTR[DI], (READ CMD SHL 8) OR READ_CMD
        STC                   ; Indicate-error and exit
        JMP     GPCExit
GPC2:
        MOV     BX,2          ; num of 64K pages/device
                              ;  for 1 Meg devices

; -----Compute total size of media

        XOR     AX,AX         ; Clear reqs for accumulation
        MOV     CX,AX         ; Count num of device pairs
        SHL     BX,1          ; 2 Devices per 16 bits
GPC3:
        INC     CX            ; Count num of device pairs
        ADD     AX,BX         ; Point to next device pair
        CALL    SetPage       ; Select page for next device
                              ; Already in READ ID mode?
        CMP     ES:WORD PTR [DI],(INTEL_ID SHL 8) OR INTEL-ID
        JNE     GPC4          ; No, Continue
        JM!?    GPC5          ; Yes, We're done
GPC4:
        MOV     ES:WORD PTR [DI],(READID CMD SHL 8) OR READID_CMD
                              ; Intel Devices?
        CMP     ES:WROD  PTR [DI],(INTEL_ID SHL 8) OR INTEL-ID
        JNE     GPC5

; Set READ mode and check next device
        MOV     ES:WORD PTR [DI],(READ_CMD SHL 8) OR READ_CMD
        JMP     GPC3
GPC5:
        MOV     ES:WORD  PTR [DI],(READ_CMD SHL 9) OR READ_CMD
        MOV     AX,CX         ; Get num of device pairs
        SHL     AX,1          ; Compute number of devices
        MOV     BX,2          ; Multiply by number of 64k pages
        MUL     BX
        MOV     TotalSizeHi,AX ; Set total size
        MOV     TotalSizeLo,0 ; Size is always a multiple of 64k
GPCExit:
        CALL    TurnVppOFF ; Turn off Vpp

        RET
GetPhyChar ENDP
```

Listing 1—*Page wrap-around is detected by leaving the first device encountered in READ_ID* mode. When a preexisting ID is found, the program knows to stop looping.

By issuing the IOCTL read command, FEFS . SYS obtained the

which has been named FlashEntry. The FlashEntry procedure is to the FFS what the Interrupt procedure is to an MS-DOS device driver. It determines a command's validity before dispatching. FlashEntry handles calls from FEFS . SYS and applications that communicate directly with the flash memory, such as a formatter.

When writing the procedures to handle the FEFS . SYS commands, it is important to follow the entry and exit protocols. This is analogous to interrupt routines expecting parameters and status information to be passed within certain registers. Take for example the procedure ReadLogBlock that reads a block of data from the flash memory SSD into a buffer. FFS is informed by MS-DOS that the destination buffer is located at ES:BX. The CX register contains the number of bytes to read. A 32-bit pointer into flash memory is supplied by DI:DX. The unit number is passed in the AL register. Upon return from this procedure, the carry flag is expected to have the status of the operation, whether it was successful or unsuccessful.

There are thirteen procedures, including ReadLogBlock, that are defined by the Microsoft FFS technical specification. To more fully comprehend the file system, a brief discussion of each is helpful:

GetMediaCheck-Determines media status (same, changed, missing, unknown).

ReadLogBlock—Read a block of data at a logical address (which must be converted to a physical address).

WriteLogBlock—Writes a block of data from a buffer to the flash memory SSD.

CheckWrite—Checks a block of data for writability. If a byte is already programmed, but still has "one" bits, that byte can be rewritten to change the "ones" to a zero. This is useful for modifying the status byte.

EraseSSD-Erases data on a selected unit.

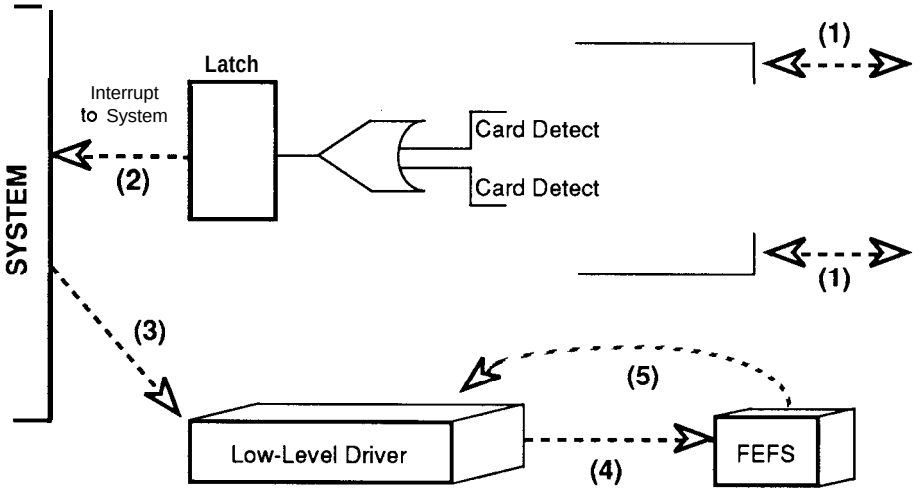
GetMediaInfo-Returns the description (type and size) of the installed media.

```
; Block Device Header use to set up drive letters with MS-DOS

Blockheader  DD  -1                  ; Last device in file
             DW  0000000000000000B ; Block device
             DW  Strategy
             DW  BlockInt
bBlkUnits    DB  8 DUP (0)           ; Initialized in BlockInt

; These structures are passed back to MS-DOS to simulate
;  BIOS Parameter Block values:

BPB1         BPB <512,1,1,2,128,1024,0F8H,2> ; Dummy values
BPB2         BPB <512,1,1,2,128,1024,0F8H,2> ; Dummy values
```

Listing 2—*To reserve an MS-DOS drive letter, a different header must be usedduring driver installation. The new header acts like a block device. rather than a character device.*

FirstFree—Finds the first non-FFh byte from the end of the SSD and, as a result, finds the first available space.

SetMediaCB—Registers a procedure as the call-back procedure which is called by the LLD when a flash memory card is inserted or removed



(1) - Card is Removed or Reinserted
(2) - System is Notified of Interrupt
(3) - System Vectors to Interrupt Routine in LLD
(4) - LLD Notifies FEFS Through Callback interface that Media Has Changed
(5) - FEFS Requests LLD to Perform a Media Interrogation

Figure 6—*The Call-back Interface supports changeable media in a system using Flash Cards.*

(see the discussion on Removable Media).

GetMediaCB-Returns the present media call-back procedure pointer. If zero is returned, no media call-back procedure exists.

WriteLogByte—Write a byte of data.

ReadLogByte-Read a byte of data.

SetEraseCB—Registers a procedure address as the call-back procedure called while erasing an SSD. It is used to display progress information.

GetEraseCB—Gets the address of the procedure called while erasing the SSD. A zero value returned indicates no erase call-back procedure has been registered.

```
BOOT_RCD        STRUC           ; Flash File System Boot Record
                                ;  (and root dir) copied in
                                ; with formatter.
wStdID          DW 0F1A5H       ; Flash Media ID (Spells FLASH)
wUniqueIDLo     DW ?            ; Uniques ID for this Flash Memory
wUniqueIDHi     DW ?
WWriteVersion   DW ?            ; Flash version required to write
wReadVersion    DW ?            ; Flash version required to read
bPointerSize    DB 0            ; Num of bits in link list ptrs
pRoot           DB 3 DUP (0) ; Pointer to root directory
bVolumeLabel    DB 11 DUP (oFFH) ; Volume label
wEraseCyclesLo  DW OFFFFH       ; Num times device erased (low)
wEraseCyclesHi  DW OFFFFH       ; Num times device erased (high)

bManfID         DB ?            ; Manufacturer's ID
bDeviceID       DB ?            ; Device ID
wDeviceSize     DW ?            ; # of 64k pages in each device
wNumDevices     DW ?            ; # dev in SIMM or memory card
wTotalSizeLo    DW ?            ; Size of media (low word)
wTotalSizeHi    DW ?            ; Size of media (high word)
wTotalPages     DW ?            ; Total number of 64k pages
bIntelID        DB  '1.0'       ; Indicates Intel format

pSibling        DB 3 DUP (0FFH) ; Start offset of data area
bRootName       DB   'ROOT   '; First entry must have this name
bRootExt        DB'      '
bStatus         DB 0FBH         ; Directory entry
pPrimary        DB 3 DUP (0FFH)
pSecondary      DB 3 DUP (0FFH)
bAttributes     DB OFFH
wTime           DW ?
wDate           DW ?

ROOT_RCD        ENDS
```

Listing 3—*Any formatting utility must be hardware dependent. This listing shows the boot record copied into the SSD at the beginning of the FFS partition.*

SETTING UP THE PAGE NUMBER

Duringinitialization,oursoftware calculated the total flash memory available, and this information is passed back to FEFS.SYS (which knows nothing about the actual hardware used to access this memory). Because a page-memory scheme is used, the linear address supplied in DI:DX must be converted to a page number and offset within that page. A procedure, called SetupRW, performs thisconversion and sets the hardware accordingly. In reality, if you are using a 64K-byte page size, this conversion is very simple: [DI] contains the page number and [DX] contains the offset. This is only tricky if you have designed your hardware with a smaller page size. In this case you would take DI and DX, move them into DX and AX, respectively, and divide DX:AX by the page size. AX would now contain the page number and DX the offset.

REMOVABLE MEDIA

If you have designed your page-memory board for the Intel Flash Memory card instead of SIMMs, you can include support for media change. The PCMCIA specifies theuseof card-detect pins on the memory card. These pins can be wired to a latch that is set whenever the card is removed or inserted. This card-change line can be read before every access or can be tied to aninterrupt line to allow instant acknowledgement of the card change. Forbothcases,Microsofthasprovided a call-back interface.

A call-back interface allows the LLD to call a procedure internal to FEFS.SYS when the media changes. This function is similar to a software interrupt except it is private and cannot be accessed by other software. At initialization, FEFS.SYS requests the LLD to perform a "Set Media Callback." In this procedure, the LLD stores the value in ES:BX which has been set up by FEFS.SYS as a pointer

to its internal media status call-back procedure. Now when the memory card is changed, the low-level driver, initiated by polling or interrupt, calls this procedure to let the FFS know that it needs to perform a media interrogation (Figure 6). This is analogous to removing a floppy disk in that MS-DOS must read the BPB and related information.

FORMATTING

A standard format utility is not incorporated in Microsoft's FFS because a formatter is hardware dependent. As such, a formatter was written that communicates directly with the LLD through `FlashEntry`. The formatter starts off by using the `EraseSSD` procedure. After erasing the SSD, a boot record (Listing 3) is copied to the SSD at the beginning of the FFS partition. During initialization, `wStdID` in the boot record is read to determine if the flash memory is formatted for the flash file system (notice that "F1A5h" spells Flash). If the SSD is FFS-compatible, the information in the remainder of the boot record describes the characteristics of the flash memory as well as the starting point for the FFS data area.

DEBUGGING

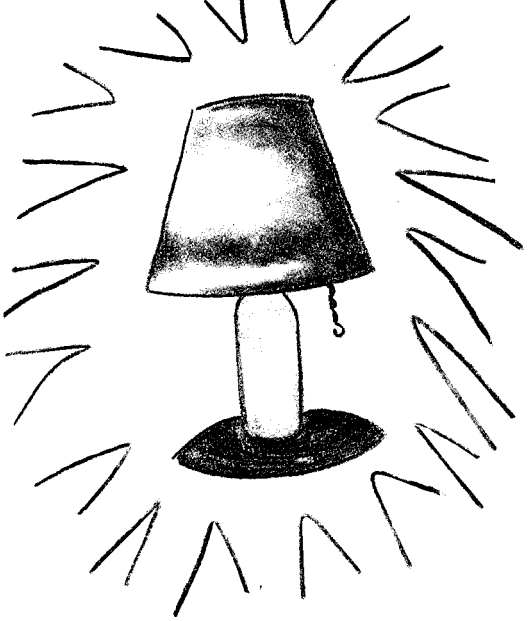
DOS Debug can be used for simple debugging purposes such as reading and writing I/O ports to test basic hardware functionality. Device drivers are difficult to debug because they are loaded during the DOS boot process. However, System Debugger by Sandpaper. Software works well for this purpose. To use System Debugger to debug this device driver, the system must be set up so that a warm reboot may be taken at any time. After the debugger is loaded, breakpoints are set and a program called `BOOT.COM` is executed. This reboots the system without removing the debugger. When the breakpoint is reached, the debugger pops up and displays the information you need. One thing to note is that this debugger is designed for '386 systems because it uses the CPU's internal break registers and extended memory.

This flash file system project will provide a great education into the world of device drivers. Some of the concepts may seem a little tricky at first, but all of a sudden, a light bulb will go on, and you will be impressed by the functionality of this unique file system. Furthermore, once the project is complete, you will be impressed by the performance of the flash memory solid-state disk. This technology is rapidly gaining market acceptance and you have become a part of the secondary storage revolution.

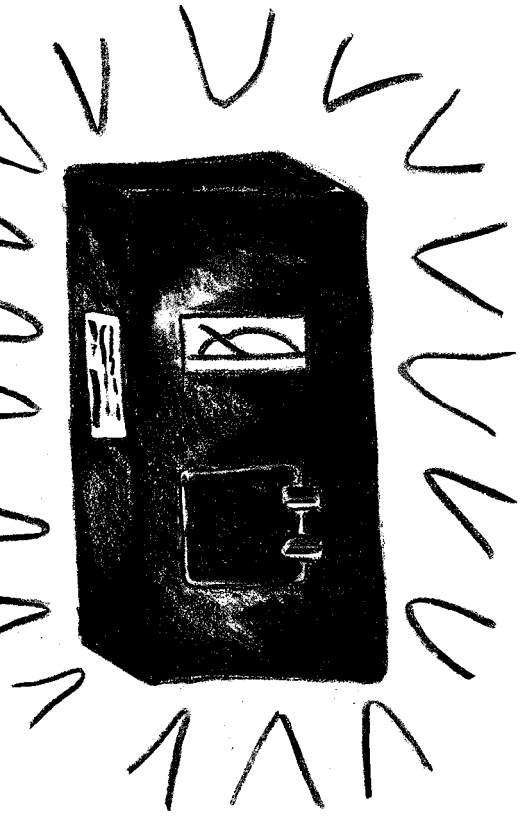
Markus Levy is an application engineer at Intel Corporation in Folsom, California, and holds a B.S.E.E. from California State University. His specialties include software and hardware implementations of solid-state disks in portable computers.

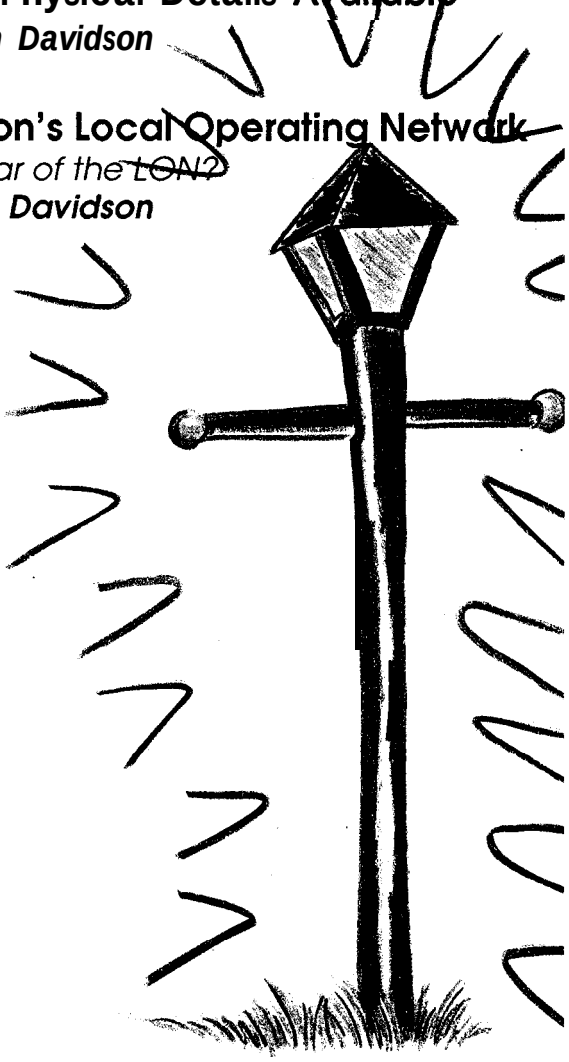
IRS

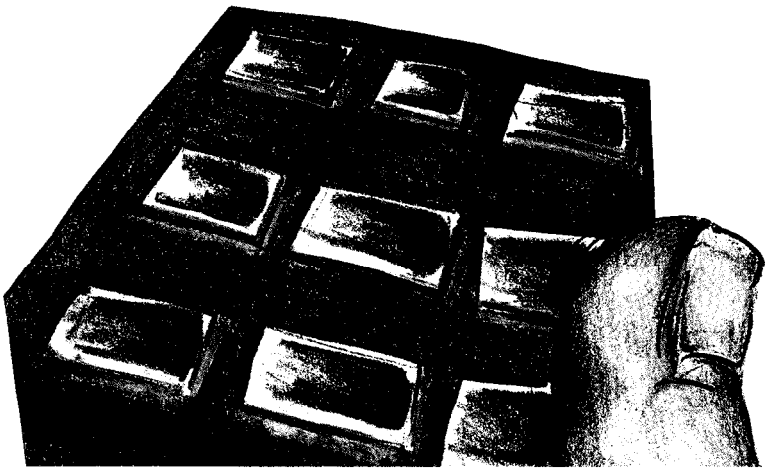
410 Very Useful
411 Moderately Useful
412 Not Useful



Home Automation







SPECIAL SECTION

Touch-Tone Interactive Monitor

A Watchdog in Every Home

Steve Ciarcia

As many of you already know, home control is a pet project of mine. Over the years I've presented at least 50 articles relating to electronically enhancing the home environment. You'd think with all this junk I'd be happy to leave well enough alone. Unfortunately, the down side of having so much electronic living is that you become used to it. When I walk into a bathroom I expect the lights to go on automatically. The unenlightened majority, of course, looks for the light switch on the wall before closing the door rather than standing in the dark waiting for divine electronic intervention.

And, heaven forbid, should I ever lose the capability of sitting in my office across town,

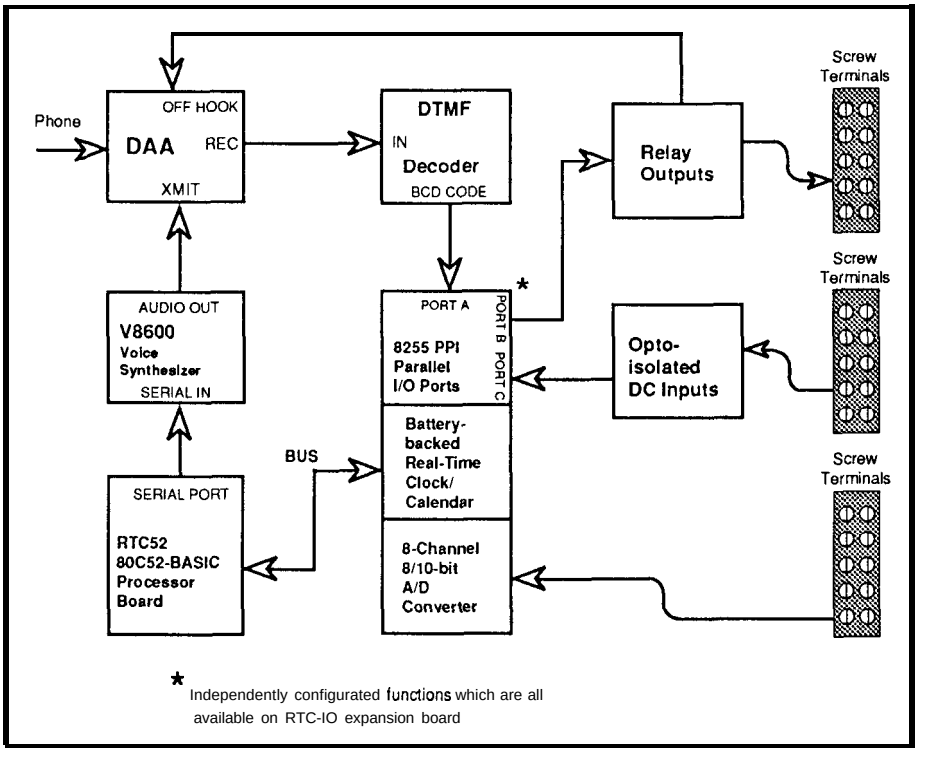


Figure 1 -The block *diagram of the Touch-tone Interactive Monitor. a data collector/ controller specifically configured to respond to touch-tone-coded commands with vocal replies.*

benefits of designing and presenting independent projects is that 1 can usually incorporate a new subsystem, such as a weather or lightning monitor, without disrupting the integrity of the current home control system, and remove it if it doesn't perform properly. By designing all system-to-

to fifteen volts on those terminals will automatically execute a preprogrammed sequence of equipment shutdowns and protective reactions. I've yet to build a lightning detector, and certainly it will have to contain sufficient computer smarts to minimize false alarms, but I **know that the rest of** my system needs only a simple "yes or no" contact closure output from that detector when it is attached.

SPREADING IT AROUND

The upside of a distributed-type home control system is that, in theory, it should not all crash at the same time. While I can be standing in a dark bathroom if the house lighting HCS controller takes a "hit," it will not necessarily affect light-

sys-

monitoring

backed over rose e commu-

pro-

The reality of life is that compo-

mat-

reserved e
for a "Lightning Detector" signal. Five

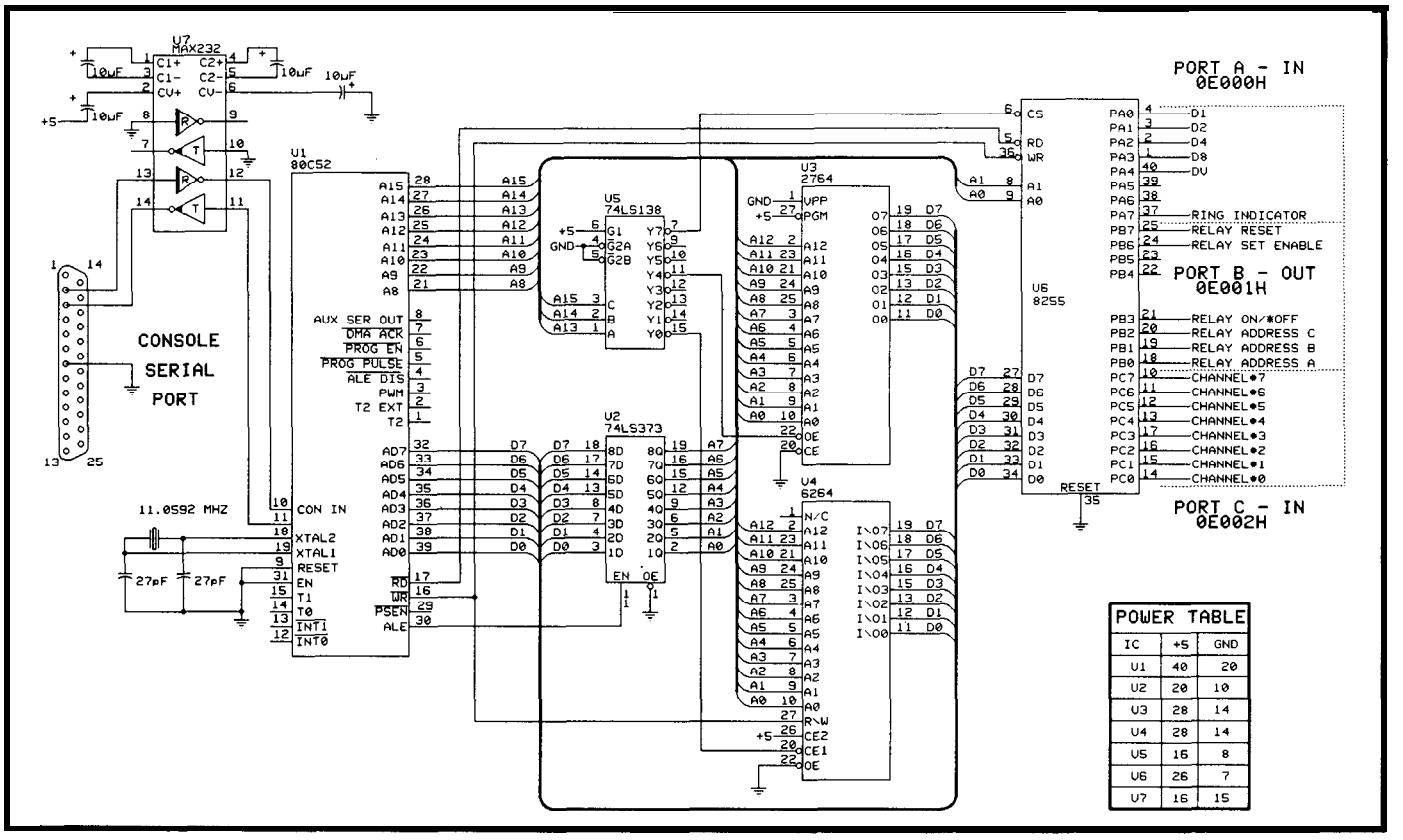


Figure 2—*The* **minimal circuitry** *schematic of a typical three-parallel-port 80C52-BASIC system capable of creating the basic TIM. These ports connect to the relays, optoisolators, and DTMF circuitry.*

signal appropriately called "HCS Crashed," will be active!

The down side of implementing a totally distributed control system is keeping track of all the independent activities and calculating some basis for total system integrity. Someone or something has to look at all the watchdog timers, battery charger levels, system monitors, and other critical parameters to make a judgement of system soundness. Simply monitoring watchdog timers for an errant event only involves a few logic gates. Second-guessing theactivitiesof a PID control loop on a solar heating system, on the other hand, involves a supervisory system probably as complicated as the systems being monitored. The space shuttle, for example, uses five computers that "take a vote" to decide who's right.

I certainly don't want to suggest that I'm proposing anything tha t complicated. Quite thecontrary! I've found that because I amcontinually revising and modifying my home control system, thereprobablyisno well-defined functional description that is valid for any length of time. The end result is that, other than monitoring the obvious crash and watchdog timer outputs, the only central supervisory judge I value is me! A more effective answer is for mc to build a supervisory system that provides data on the collectivcactivitiesand statusesof the other systems and lets mc decide if they are operating correctly. Rather than just automatically triggering an alarm at some preset battery charger voltage limit (remember, computers only do what you tell them to do), I'd prefer to know what that voltage is, continue to monitor it as long as I determine, and have the capability, even remotely, torcdcfine the trigger limits or take alternative corrective action.

Getting from here to there is what brings us to the project at hand. Remotclymonitoringandcontrollingmy HCS system is nothing unique--there **has** always been a modem port. Using a computer and a modem Icancall my HCS from any telephone and control it as if I wcreatthcterminaldirectlyon the HCS. In reality, however, this is a rarely used feature. Any of you who read my editorial from the last issue knows I'm not about to drag around a portable computer for anything, even my HCS.

TOUCH-TONE INTERACTIVE MONITOR

Still, it's hard to ignore the benefits of using something as universally available as the telephone. The intelligent alternative was to design a supervisory system that could communicate by telephone in a more traditional manner with no added hardware requirements. A quick analysis of the average phone suggests that tradition is spelled only as either touch-tone (dual tone multifrequency, DTMF) or voice, however!

Without too much cliché, let me just say that this, fortunately, is one instance where something could be as easily done as said. The solution was TIM.

Figure 1 is the block diagram of my Touch-tone Interactive Monitor (TIM, for short). TIM is a data collector/controller specifically configured to respond to touch-tone-coded commands with vocal replies.

To hear TIM, I just call my house from any touch-tone telephone. When TIM answers, he (it's a male voice) states that I've called the Circuit Cellar HCS and asks that I enter an access

code. The access code is a 4-digit number that TIM is preprogrammed to accept. He also has codes for various friends who might substitute "house sit." I get 45 seconds to do it right or he hangs up.

Pressing the proper four digits merits a "Hello, Steve," as the system recognizes my code. From there I can ask for directions by pressing "0" (for operator) or enter a specific code to tell me things like whether the outside lights are on (obviously I know when they are supposed to be and such a failure would indicate some other system fault), alarm status, AC power status, whether anyone has entered the property perimeter within the last x hours, current temperatures as well as other distinctly analog measurements, and specific control settings among other things. Any or all of these statuses and readings are spoken vocally as they are requested.

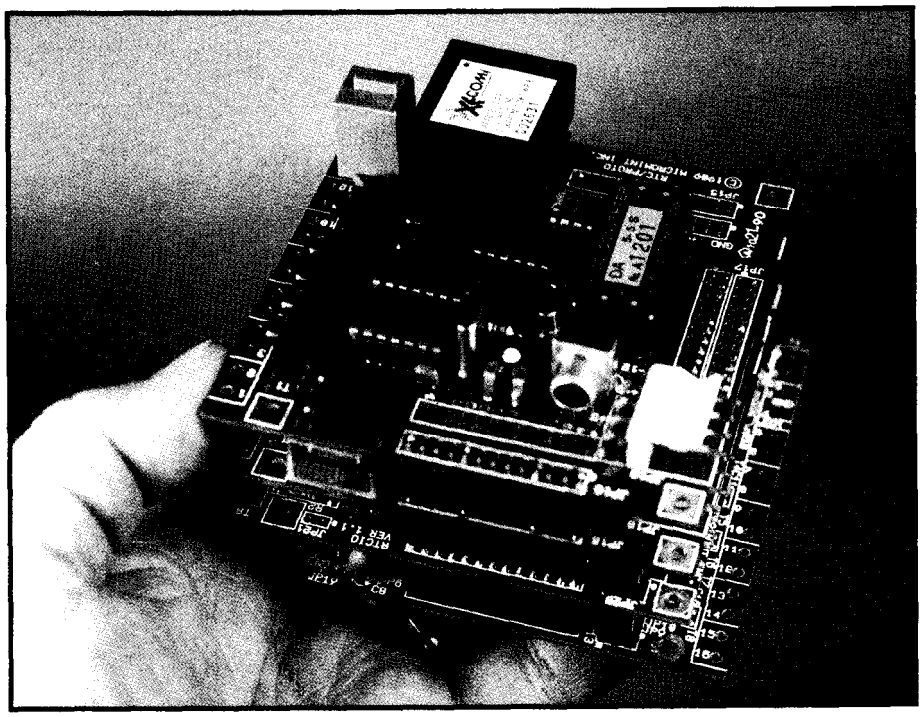


Photo 1 — The *complete prototype for TIM. Several design decisions were made based on the space available for components.*

What separates this unit from being just a message system is that TIM has a functional architecture that facilitatesintelligent control as well as reporting. TIM incorporates six relays of its own that can be used for independent control or system-to-system signaling. Using DTMF codes I can conceivably close or open those relays on command. 1 could use one of them to operate a battery charger, for example, and manually turn it on or off after hearing a report of the battery voltage level. Or, more appropriately, until I actually build a lightning detector, if I suspect a thunderstorm is brewing I can call from my office and activate a relay from the human lightning detector-me-to the HCS. How one

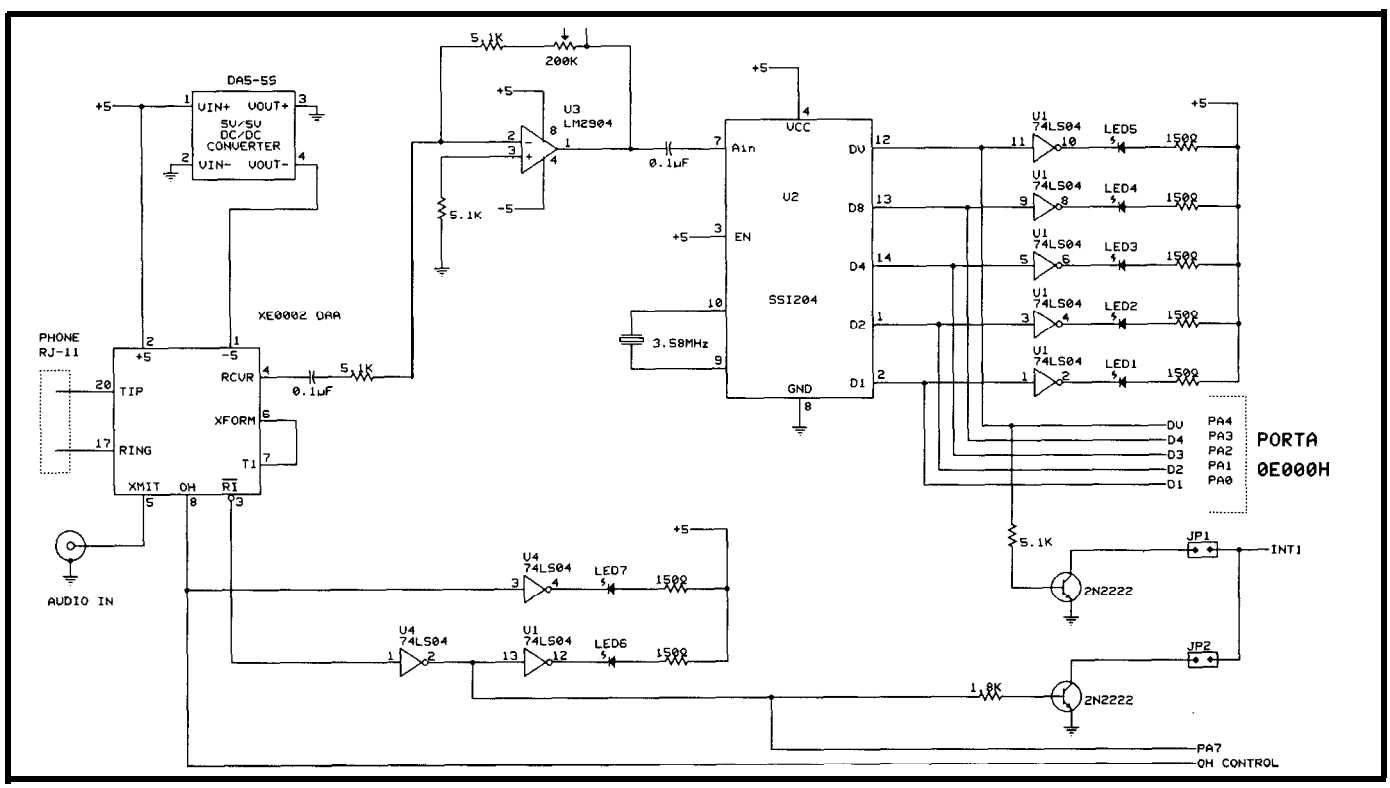


Figure **3**—*The TIM DAA/DTMF interface. The* only *modification to the description of the DAA and SSI204 chip is the necessity of an amplifier between them. The LM2902 is a dual op-amp. just in case I have to reconfigure the circuit as an automatic gain-controlled amplifier.*

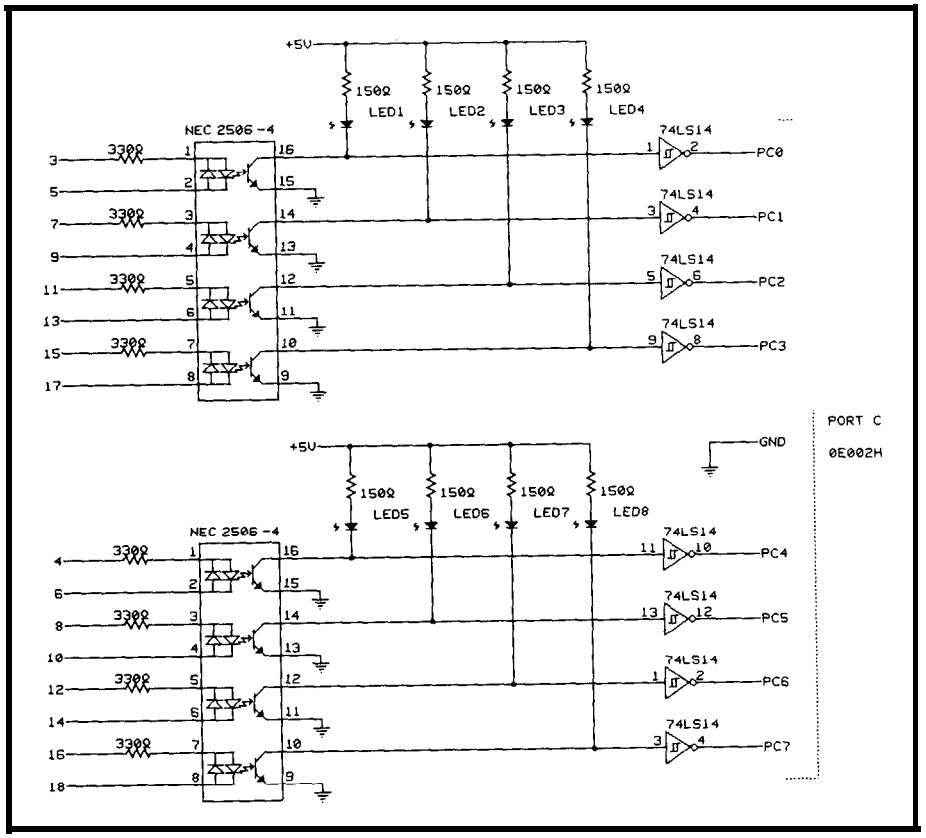


Figure 4—The d-input DC-level optoisolated input inferface utilizes NEC2506-type optoisolators with opposed-parallel-connected LEDs to eliminate input polarity sensitivity and provide 1500V of electrical isolation.

might use these relays or any other I/O from TIM is purely dependent upon the application and your programming ability.

UNIQUE PERIPHERALS ON A STANDARD CONTROLLER

TIM's brain is a MicromintRTC52 80C52-BASIC controller; his ears arc a DTMF decoder chip; and, his voice is an RC Systems V8600 text-to-speech voice synthesizer.

The minimal required expansion to the basic RTC52 is an 8255 three-port peripheral interface chip addressed at EOOOH. Port A is connected to the DTMF decoder, Port B controls the output relays, and Port C connects to the optoisolated input section.

Only three parallel ports are necessary to create the basic TIM configuration. However, by adding a clock/calendar and ADC chip, TIM's functionality is significantly expanded. If you use an RTCIO expansion board to provide the 8255 PPI as I did, the clock/calendar and A/Dcomponents are on the same board. *[Editor's Note: See "Creating a Netwurk-based Embedded Controller," in CIRCUIT CELLAR INK #8, for details of the RTC52 und RTCIO.]*

As an alternative for illustration here, Figure 2 outlines the minimal circuitry schematic of a typical three-parallel-port 80C52-BASIC system capable of creating the basic TIM (because of the reduced expandability, there are significant differences from the schematic of the RTC52). These ports connect to the relays, optoisolators, and DTMF circuitry.

THE TELEPHONE INTERFACE

The following are two abbreviations that are essential to TIM's operation: DAA and DTMF. DAA stands for Data Access Arrangement and DTMF stands for Dual Tone MultiFrequency (as I've already mentioned). Touch-Tone, which we normally call the latter, is actually an AT&T trade name for the DTMF frequencies specifically used for phones.

The Xecom XE0002 DAA is a module that provides a direct connect telephone interface. It is registered under part 68 and FCC recertification is not required when integrated into systems, providing that the label lists the registration number and ringer equivalence. The DAA module is directly PC board mountable and telephone line connection is made via an external cable with a RJ11C or equivalent mating plug.

A registered DAA is more than a 600-ohm transformer. Among its functions are ring detection, on/off hook control circuitry, modemcontrol logic, and analog transmit/receivelogic.The telephone is attached to the Tip and Ring side of the DAA and anything

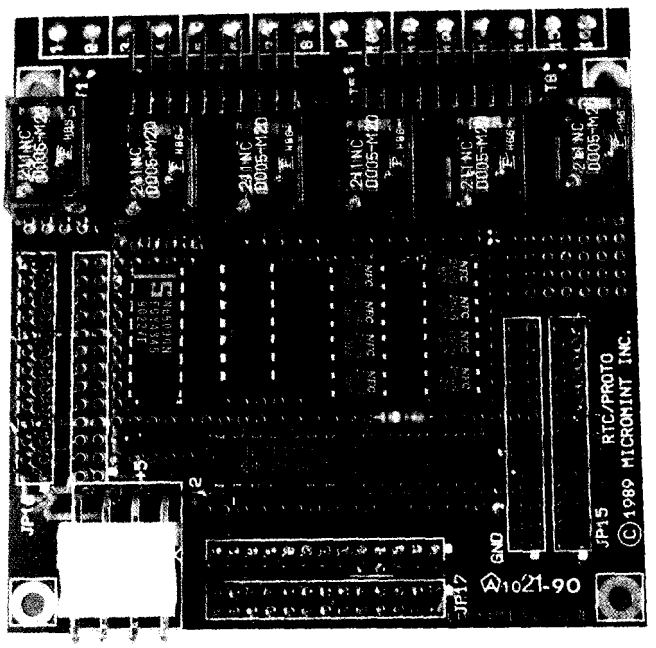


Photo 2—The optoisolation and relay circuitry (shown schematically in figures 4 and 5) let TIM do the work and talk about the results.

```
100  REM DTMF/Voice interactive Security System Rev 0.05
105  REM With RTCIO time/date display
110  DIM T(15)
120  XBY(0E03FH)=4
130  PRINT
440  BAUD 4800
450  PRINT CR, CR, "Ready"
500  REM Setup Routines
510  MODE=0E003H : REM 8255 PPI at address 0E000H
520  DTMF=0E000H  : LATCH=0E001H : CHANL=0E002H
530  XBY(LATCH)=255  : REM clear relay outputs
540  XBY(MODE)=099H : REM Set ports A and C input, B output
550  CLOCK 0 : TIME=0
560  KEY=0 : CODE=0  : FLAG=0
580  REM
590  REM
600  GOTO 1000
610  REM
700  REM Read Clock/Calendar Chip
710  FOR Z=0 TO 15
720  T(Z)=(XBY(0E030H+Z).AND.15)
730  IF (T(Z)<>(XBY(0E030H+Z).AND.15)) THEN 720
740  NEXT Z
750  PRINT 10*T(5)+T(4)," Hours ",
760  PRINT 10*T(3)+T(2)," Minutes"
765  RETURN
770  REM
1000 ONEX1 2000
1010 ONTIME 45,2700
1020 IF FLAG=0 THEN GOSUB 4000  : REM Get access code first
1050 IF CODE=10 THEN CODE=0 : GOSUB 2550 : REM List directions
1060 IF CODE=1 THEN CODE=0: GOSUB 3000 : REM List inp status
1070 IF CODE=2 THEN CODE=0 : GOSUB 4700 : REM State cur Time
1080 IF CODE=3 THEN CODE=0 : GOSUB 4800  : REM Read inside temp
1090 IF CODE=4 THEN CODE=0 : GOSUB 5000
1100 IF CODE=5 THEN CODE=0 : GOSUB 5000
1110 IF CODE=6 THEN CODE=0 : GOSUB 5000
1120 IF CODE=7 THEN CODE=0 : GOSUB 5000
1130 IF CODE=8 THEN CODE=0 : GOSUB 5000
1140 IF CODE=9 THEN CODE=0 : GOSUB 5000
1300 GOT0 1000
1490 REM
1500 REM
2000 X=XBY(DTMF)  : REM read DTMF decoder
2010 IF (X.AND.128)=128 THEN GOSUB 2500
2020 IF (X.AND.16)=16 THEN GOSUB 2600
2100 RET1
2200 STOP
2500 REM: autoanswer
2510 CLOCK 1 : XBY(LATCH)=8 : XBY(LATCH)=9: REM strt call tmr
2515 FOR Q=1 TO 1000 : NEXT Q
2520 PRINT CR, CR, CR,
2525 XBY(LATCH)=15  : REM go off hook
2530 PRINT "Hello you have called the Circuit Cellar"
2535 PRINT "Home control system"
2540 PRINT "Please enter your access code now": PRINT : X-0
2543 XBY(LATCH)=0  : XBY(LATCH)=1  : REM reset call init relay
2545 RETURN
2550 PRINT "press the pound sign to stop talking"
2555 PRINT "press the star to hang up"
2560 PRINT "press a number for specific reports"
2562 PRINT "press one for input status"
2564 PRINT "press two for time"
2566 PRINT "press three for inside temperature'*
2568 PRINT "press four for last sensor activated"
2578 PRINT "press zero for directions"
2580 PRINT
2590 RETURN
2594 REM
2600 REM read DTMF input code
2610 CODE=X.AND.15
2620 IF CODE=11 THEN PRINT "bye" : XBY (LATCH)=7 : GOSUB 4500 :
       RETURN
2630 IF CODE=12 THEN PRINT CHR(24): RETURN : REM voice off
2640 KEY=1
2650 RETURN
2700 REM Call timeout
2710 PRINT "bye" : XBY(LATCH)=7: REM Go on hook
2730 GOSUB 4500
2740 RET1
2750 REM
3000 REM Input channel reports -- Active high
3010 Y=XBY(CHANL)
3020 IF (Y .AND.1)=1 THEN PRINT  "Alarm System is activated"
```

(continued)

sting 1—*TIM is programmed in BASIC, which provides all of the features needed for the project.*

you build is attached to the other side. The DAA serves to protect your circuitry from line transients and the phone line from your circuit.

Audio that is destined for the telephone line is called transmit (XMIT) audio. This audio, in our case, comes from the voice synthesizer. It is applied at the XE0002's XMIT signal input. The synthesizer's output volume should be adjusted to keep the audio level, heard by the caller, at the appropriate volume. The DTMF tones pressed by thecaller are output on the XE0002's RCVR pin.

The device is powered from ±5-volt supplies, but logic control inputs and all status outputs are CMOS zero to +5 volts level compatible. Since the RTC systemoperateson +5 voltsonly, a separate -5-V DC-to-DC converter was added to power the DAA.

DTMF DECODING

The full DTMFencoding standard defines four rows and four columns for a total of 16 two-tone combinations. Standard telephones use only 12 of these combinations. Depending upon your application, the extra codes may or may not be useful.

The 12 keys are arranged in four rows and three columns, **as shown** in Table 1. All the keys in a given row or column **have** one tone in common. For example, pressing the digit "9" (row 2 and column 2) produces an 852-Hz and a 1477-Hz tone simultaneously. Similarly, pressing "4" (row 1 and column 0) produces 770-Hz and 1209-Hz tones simultaneously.

The eight frequencies associated with the rows and columns are separated into two groups. The low group, containing row information, has a range of 697 Hz to 941 Hz. The high group, containing column information, covers 1209 Hz to 1633 Hz.

The SSI204 (renumbered and most recently sold as part number SSI 75T204), is a 14-pin 5-volt chip that detects all 16 DTMF tone pairs. It uses an inexpensive 3.58-MHz "colorburst' crystal and requires no front end prefiltering. The SSI204 incorporates "switched-capacitor" filtering to separate the high- and low-frequency

```
3030 IF (Y.AND.2)=2 THEN PRINT "Outside lights are on"
3040 IF (Y.AND.4)=4 THEN PRINT '*Temperature limits are OK"
3050 IF (Y.AND.8)=8 THEN PRINT "Power levels OK"
3060 IF (Y.AND.16)=16 THEN PRINT "Driveway sensor activated'*
3070 IF (Y.AND.32)=32 THEN PRINT "Water Sensor activated*'
3080 IF (Y.AND.64)=64 THEN PRINT "System problem 1 has occurred"
3090 IF (Y.AND.128)=128 THEN PRINT "System problem 2 has occurred
3095 IF Y=0 THEN PRINT "no inputs activated"
3100 RETURN
4000 REM decipher access code
4010 N=O
4040 IF KEY=1 THEN NUM(N)=CODE : N=N+1 : KEY=0
4050 IF N<4 THEN 4040
4060 REM
4070 ENTRY=(NUM(0)*1000+NUM(1)*100+NUM(2)*10+NUM(3))
4080 CODE-O
4090 IF ENTRY=4177 THEN PRINT "Hello Dan" : GOT0 4150
4100 IF ENTRY=5273 THEN PRINT "Hello Ed" : GOTO 4150
4110 IF ENTRY=6264 THEN PRINT "Hello Steve" : GOTO 4150
4120 IF ENTRY=3665 THEN PRINT "Hello Jeff" :GOTO 4150
4130 IF ENTRY=6116 THEN PRINT "Hello Jeannette" : GOTO 4150
4140 PRINT "Please enter your access code again now": GOT0 4010
4150 PRINT "What can I do for you today" : KEY=0
4210 TIME=0 : FLAG=1 : RETURN
4250 REM
4500 CODE=0 : CLOCK 0 : TIME-O : KEY-O : FLAG=0
4510 RETURN
4520 REM
4700 REM Print current time from clock/calendar chip
4710 PRINT "The time is ":GOSUB 700 : RETURN
4800 REM Compute inside temperature
4810 XBY(0E013H)=0 : REM Start conversion on A/D channel #3
4815 V=XBY(0E010H)
4820 TEMP=INT((100*(V/256)*5))
4830 PRINT "Inside temperature is ",TEMP," degrees"
4850 RETURN
5000 PRINT "Function not working yet": RETURN
```

Listing 1 -continued

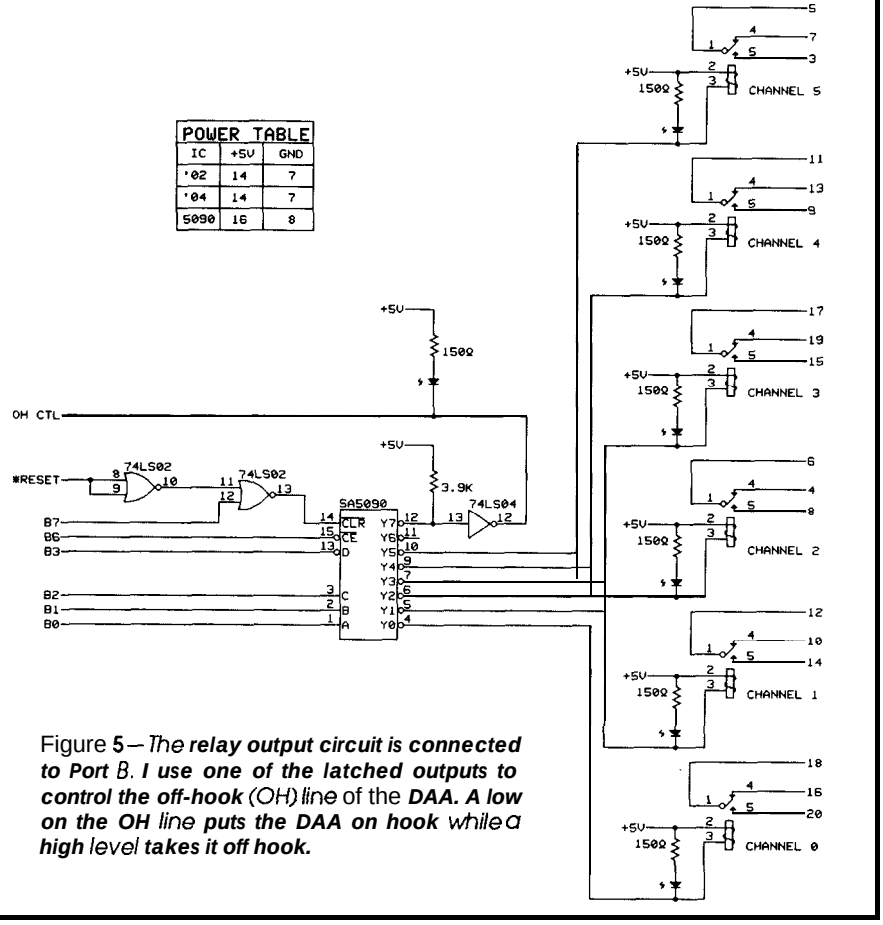


Figure 5—*The relay output circuit is connected to Port B. I use one of the latched outputs to control the off-hook (OH) line* of the *DAA. A low on the OH line puts the DAA on hook while a high level takes it off hook.*

bands as well as detect the individual tones. The output is hexadecimal 4-bit CMOS tristate logic with a data available strobe.

Figure 3 is the schematic of the TIM DAA/DTMF interface, and the completed prototype is shown in Photo 1. The only modification to the description of the DAA and SSI204 chip is the necessity of an amplifier between them. This amplifier gain is adjustable and, so far, a single setting seems to have worked with all the phones I have tried. The LM2902 designated in the schematic is a dual op-amp, however, just in case I have to reconfigurethecircuitasanautomatic gain-controlled amplifier later. Board space is already at a premium.

When a DTMF signal is received, the particular code for that tone pair (row-column) is presented on the D1–D8 lines (D8 is MSB) and the data available (DV) line goes high. The DV line stays high until the input signal is released. These lines are connected to the Port A connections of the 8255. One further point to note is that both the ring indicator (RI) on the DAA and the data available (DV) line on the SSI204 are connected to the interrupt (INT1) line of the processor through a pair of open-collector transistor drivers. Because these events are short lived and time critical, they should be processed under interrupt programmingrather than a scanned input port. The LEDs are more or less ornamental. I like visual indicators to let me know circuits are working, but none of the LEDs are required.

ISOLATED I/O INTERFACING IS FOR MUTUAL PROTECTION

The whole concept behind TIM is tofurnishatelephonecallerwithstatus information provided from connections to its relays and input interface. However, the cardinal rule of distributed processing intersystem connections is: "Thou shalt not crash the rest of the system when thee takes a dive!"

Justasyou would not havea single backup power supply cover all the processors in a distributed control system, neither would you make TTL or common-ground connections over hundreds of feet of wire between systems. A single over-voltaged power supply or inadvertent connection to the AC power line can be coupled through the entire system from unit to unit. For maximum safety in embedded applications, independent units should be electrically isolated from one another. If they have a serial connection, it too should be isolated.

Digit	Output Code D8	D4	D2	D1	Digit	Output Code D8	D4	D2	D1
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	0	1	0	1	0
3	0	0	1	1	*	1	0	1	1
4	0	1	0	0	#	1	1	0	0
5	0	1	0	1	A	1	1	0	1
6	0	1	1	0	B	1	1	1	0
7	0	1	1	1	C	1	1	1	1
8	1	0	0	0	D	0	0	0	0

Table 1—*DTMF coding means that every location in a column and everylocation in a row share a common tone. 'Touch-tone' is a trademarked implementation of DTMF encoding.*

		High Group			
		Column 0 1209 Hz	Column 1 1336 Hz	Column 2 1477 Hz	Column 3 1633 Hz
Low Group	Row 0,697 Hz	1	2	3	A
	Row 1, 770 Hz	4	5	6	B
	Row 2,852 Hz	7	8	9	C
	Row 3,941 Hz	*	0	#	D

My simulated "lightning detection" signal is a connection between TIM and the HCS. Since the inputs of my HCS are not truly isolated (only current limited) and any input signal is common ground with the processor, it is up to me now to make connections to the HCS only from isolated "contact closures" rather than direct application of voltage levels which are common ground with the processor in the next system. Of course, if you feel lucky.. .

ISOLATED INPUTS

Figure 4 is the circuit for an 8-input DC-level optoisolatcd input interface (pictured in Photo 2). It utilizes NEC2506-type optoisolators with opposed-parallel-connected LEDs to eliminate input polarity sensitivity as well as provide 1500 V of electrical isolation. A **330-ohm** series resistor allows an input range of abou t 3-20 V (I suggest that you limit the top to 12 V).

The outputs of the optoisolators are connected to directly drive eight LEDs. The LEDs present a visual indication of input status as well as operability. The typical technique of using a visual LED in series with the optoisolator LED would have polarized the input as well as raised the input voltage necessary to activate the coupler. Because the LEDs attached to the output transistor collectors raise the relative V_{ce} saturation voltage, however, it is suggested that the chips used to invert the signals between the optoisolators and the 8255 be 74LS14 Schmitt-triggered devices.

The relay output circuit of Figure 5 (also pictured in Photo 2) is similarly connected to Port B. The only reason for there being six relays rather than four or eight is that was all I could fit on theprotocard. I suppose you could make it eight, but note that I am utilizing one of the latched outputs to control theoff-hook (OH) line of the DAA. A low on the OH line puts the DAA on hook while a high level takes it off hook.

Generally speaking, thcSA5090 is a bus-connected device ra ther than a port-connected element. It was specifically chosen because of a peculiarity of the 8255. When first powered up, the 8255 sets itself to all input ports. Given the typical relay circuit configuration that would have a noninvertingdriverand a relay connected to each parallel port pin, the high impedance level of an output bit suddenly reconfigured as an input would momentarily turn on all the output relays. This is something unforgivable in a controller.

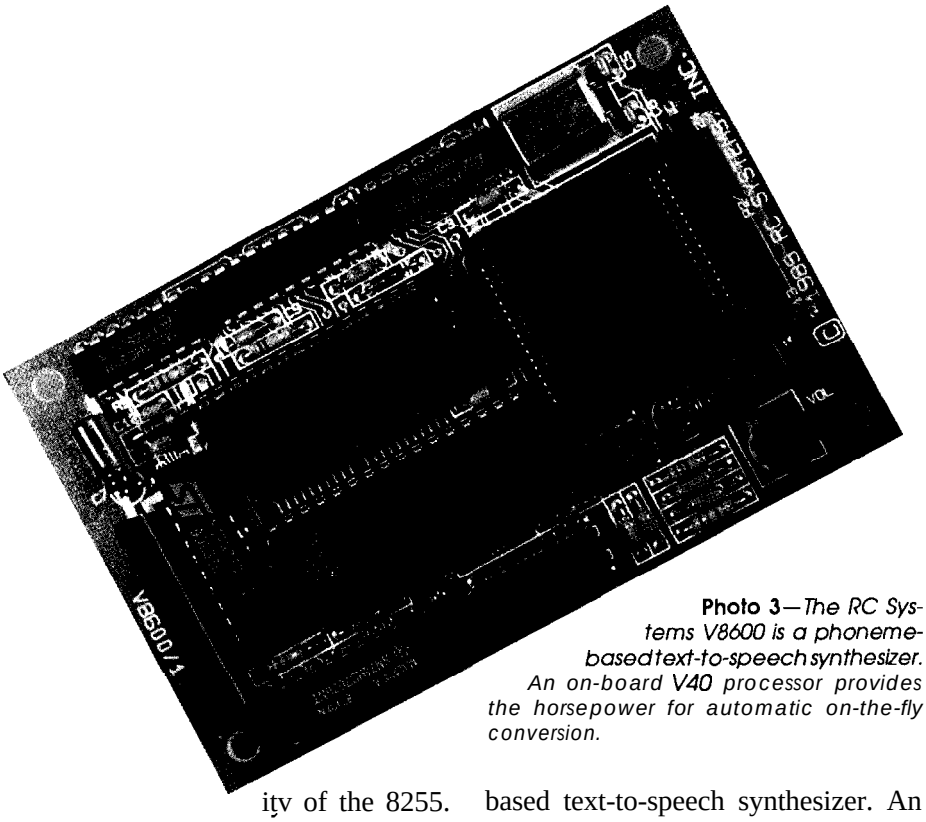


Photo 3—*The RC Systems V8600 is a phoneme-based text-to-speech synthesizer. An on-board V40 processor provides the horsepower for automatic on-the-fly conversion.*

To alleviate that situation, Port B is connected to simulate a bus interface to the SA5090 instead. A high level on B7, such as happenson startup, or a system reset, will clear all relays. Addressing specific relays consists of setting a 3-bit address on AO-A2 and a relay on/off (1 or 0) level on D while chip enable (pin 14) isbrought low. To turn on channel 2, for example, you would output OA hex to Port B. To turn off channel 2 you would send 02 hex. Again, I put LEDs across therclay coils for monitoring purposes only.

TALK TO ME!

TIM depends upon a DTMF decoder to understand a caller's commands and uses a voice synthesizer to respond. The voice synthesizer I selected is the RC Systems V8600 shown in Photo 3. The V8600 is a phoneme-based text-to-speech synthesizer. An on-board V40 microprocessor automatically converts plain ASCII text presented to its input pins into a high-quality male voice at the speaker output. Measuring 2.75"×4.25" and operating on only 5 volts, the V8600 was ideal for this project (I had considered using my original Microvox, which is also a text-to-speech unit, but it has no "stop talking" command other than a hard-wired reset).

The V8600 is a stand-alone unit that is intended to piggyback onto the host processor. It can receive ASCII via a serial or parallel printer port interface. I connected the V8600 to the serial port so the BASIC PRINT statements could either be spoken or displayed on the console terminal.

In addition to text-to-speech, the V8600 produces sound effects or tones (including DTMF), and can speak with varied pitch and intonation. Table 2 shows these various commands. My demonstration softwareonlyused the default command settings so that the PRINT statements weren't full of control characters. In my opinion, the V8600 is quite intelligible for a phoneme synthesizer and "tweaking it" may not be worth the effort given the small number of callers 1 have involved.

One further note about using the V8600: The V8600, unlike the Micro-VOX (which has a hardware phoneme chip), uses a software-generated phonetic synthesizer. The positive result is, of course, lower hardware cost, but the penalty for the designer is more complicated interrupt-intensive software. Even though the V8600 has an input buffer, the voice-output DAC has a higher interrupt priority than either theserial or parallel input ports, and the rate at which it accepts new data is affected by concurrent functions. Any PRINT routine should eithermonitor the"data terminal ready" line to send new charactersonly when allowed, or incorporate delay loops to wait for one line to be spoken before sending another. Given the "snail's pace" response times of any voice interactive system, however, neither of these "traffic monitor" approaches should even be noticeable.

Table 2-The *V8600's on-board V40 allows a relatively simple command set to provide for complex text-to-speech conversion.*

Command	Function	Range	Default*
nB	Punctuation Level	o-7	6
nC	Character mode/delay	o-31	0
D	Phoneme mode		
E	Enable intonation		
nF	Formant frequency	0-9	5
J	Tone generator		
L	Load exception dictionary		
M	Disable intonation		
nP	Pitch	0-99	50
Fi	Clear		
nS	Speed	0-9	5
nT	Text mode/delay	o-15	0
U	Enable exception dictionary		
nV	Volume	0-9	5
nX	Tone	0-1	1
nY	Timeout delay	o-1 5	0
Z	Zap commands		
@	Master reset		
n#	PCM mode	0-99	
	Store defaluts		

SOFTWARE

Fortunately for me it doesn't take a lot of software to make TIM function. BASIC works fine. Listing 1 is a program which demonstrates a first pass at some of TIM's features.

The program operation is sequential in nature with the exception of response to the ring indicator (RI) or DTMF data available line (DV) which are handled on interrupt. When an interrupt (INT1) occurs, the program first determines its source If it is the ring indicator (PA7 will be high), the program will command the SA5090 to set bit 8 to take the DAA off hook and starts a 45-second timer. If it is a DTMF tone (PA4 will be high) and depending upon where we are in the program, either a 4-digit access code must be entered or the individual tone will be interpreted as commands which TIM responds to. Of course, there is another timer to keep the phone line from being locked up forever.

Once in the command section we can either ask for directions (0), a list of input status (1), time (2), or analog input measurements (3). Speech is termina ted by pressing the # but ton (you can immediately select another command) or terminate the call by press-

ing the *button. This program is by no means all you can expect from TIM. All it is is my first pass at some demonstration software. In fact, I'm sure that as you read this, TIM will have considerably more capability than what I've demonstrated here. Nothing is static in the Circuit Cellar!

One consideration is to allow TIM to initiate a telephone call if he determines there is a problem (of course, by that time other systems in the Circuit Cellar will be jamming the other three phone lines). If I had used a combination DTMF encoder/decoderchip, this function would have been more obvious. However, using the V8600 synthesizer to produce DTMF codes eliminates the need to do it in hardware.

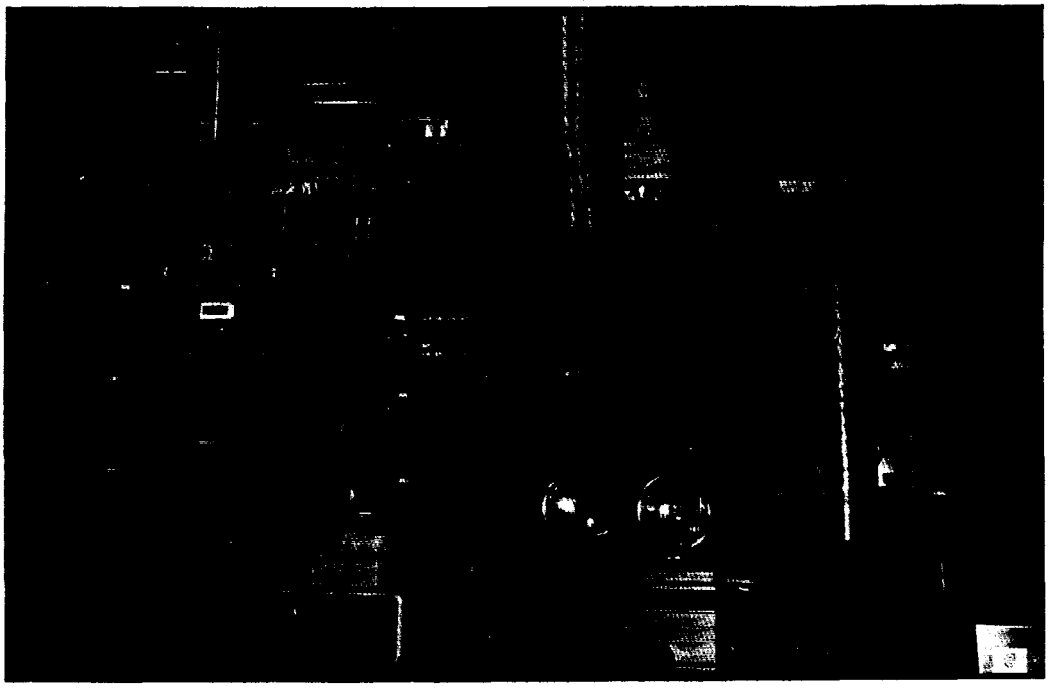


Photo 4—*TIM's* prototype *board isn't the only crowded* **piece** *of real estate in the Circuit Cellar! The next major project may* **involve sledgehammers as well as soldering** *irons.*

IN CONCLUSION

As you'll note from Photo 4, the control system area behind the Circuit Cellar is getting a little crowded. Soon I'll either have to take a sledge hammer to the water softener or the air conditioner to get more wall space. I used to have some of the audio-video controls on this wall too but they also expanded at the same rate. Directly opposite the home control system panel is a 6-foot high 19" rack full of security system and video controller stuff. Believe me, I'm never moving.

1 chose the RTC52 for its small form factor. The last time I built something like TIM it took up half the wall by i tsclf. This version of TIM ended up being a neat 3.5" x 3.5" 4-board (Photo 1) stack with the V8600 next to it. Unfortunately, the interconnection requirementsand vertical orientationof the rest of my control system creates another problem entirely. To facilitate mounting an unintcrruptable power supply (I used the 12-volt–to–5-volt DC-to-DC converter power supply from a previous article and a battery), the voice synthesizer, and screw terminal connections for the ADC, optoisolated input, and relays, I had to resort to mounting TIM in a 10" card cage on the wall (all the LEDs are toward the front edge of the protoboards for that reason).

So, where do we go from here? TIM works fine, but one really wouldn't want to listen to an encyclopedia being recited on a long-distance telephone line. So there is a limit to the ultimateusefulnessinitspresentform.

With the construction of TIM, however, I now have a central supervisor that is gathering lots of wonderful data for me. I've just restricted it to talking to me on a phone because I'm too lazy to carry a portable computer. So, what if I had this little box that could take a quick data burst from TIM directly and display.. .

Stare Ciarcia is an electronics engineer and computer consultant with experience in *process* control, *digital design, and product development.*

The XE0002 DAA is available from:
Xecom, Inc.
374 Turquoise st.
Milpitas, CA 95035
(408) 945-6640

The V8600 Voice Synthesizer is available from:
RC Systems, Inc.
121 West Winesap Rd.
Bothell, WA 98012
(206) 672-6909

The following is available from:
Circuit Cellar Kits
4 Park St.
Vernon, CI 06066
(203) 875-2751
Fax: (203) 872-2204

1. SSI204 DTMF decoder chip	$14.50
2. RTC52 BASIC-52 controller kit	$89.00
3. BASIC-52 Programmer's manual	$15.00
4. RTCIO-K1 Expansion board kit w/8255 PPI and 8-bit ADC	$79.00
5. RTCIO-K2 Expansion board kit w/8255 PPI, 8-bit ADC. DC-to-DC converter, and battery-backed clock/calendar	$119.00
6. RTC-PROTO Prototyping board	$39.00

Prices do not include shipping.

IRS

413 Very Useful
414 Moderately Useful
415 Not Useful

ing the * button. This program is by no means all you can expect from TIM. All it is is my first pass at some demonstration software. In fact, I'm sure that as you read this, TIM will have considerably more capability than what I've demonstrated here. Nothing is static in the Circuit Cellar!

One consideration is to allow TIM to initiate a telephone call if he determines there is a problem (of course, by that time other systems in the Circuit Cellar will be jamming the other three phone lines). If I had used a combination DTMF encoder/decoder chip, this function would have been more obvious. However, using the V8600 synthesizer to produce DTMF codes eliminates the need to do it in hardware.

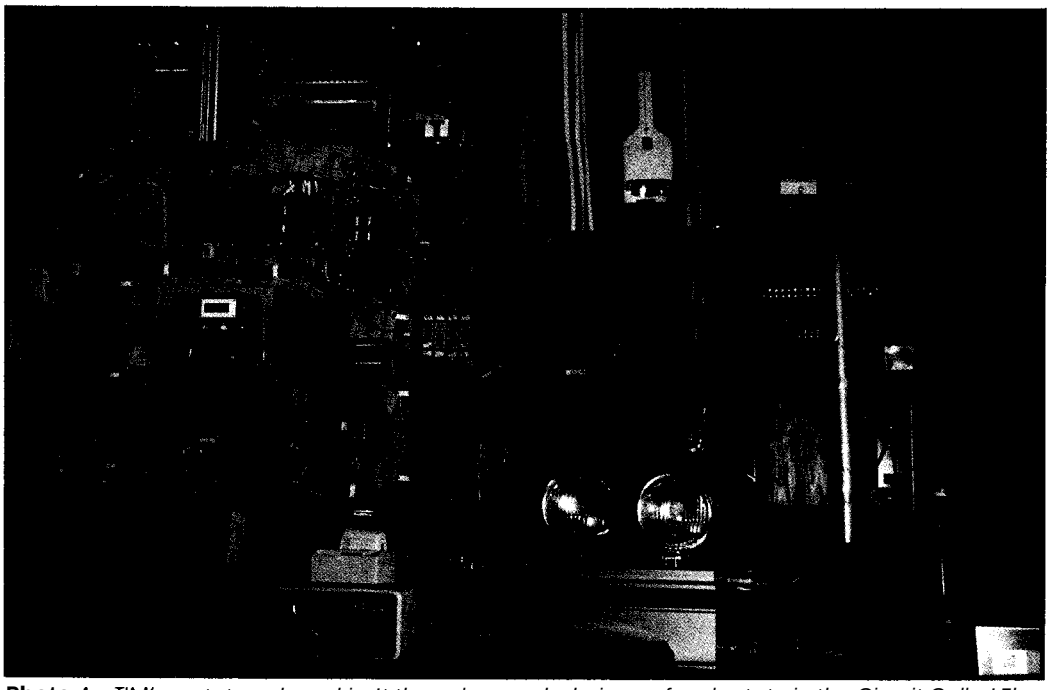


Photo 4—*TIM's prototype board isn't the only crowded piece of real estate in the Circuit Cellar! The next major project may involve sledgehammers as well as soldering irons.*

IN CONCLUSION

As you'll note from Photo 4, the control system area behind the Circuit Cellar is getting a little crowded. Soon I'll either have to take a sledge hammer to the water softener or the air conditioner to get more wall space. I used to have some of the audio-video controls on this wall too but they also expanded at the same rate. Directly opposite the home control system panel is a 6-foot high 19" rack full of security system and video controller stuff. Believe me, I'm never moving.

I chose the RTC52 for its small form factor. The last time I built something like TIM it took up half the wall byitsclf.ThisversionofTIM ended up being a neat 3.5" x 3.5" 4-board (Photo 1) stack with the V8600 next to it. Unfortunately, the interconnection requiremcntsand vertical oricntationof the rest of my control system creates another problem entirely. To facilitate mounting an unintcrruptable power (Lipplyed the 12-volt–to–5-volt optoi-

10"ort to mounting TIM in a LEDson the wall (all the proto-

.TIM directly and display..

Steve Ciarcia is an electronics engineer and computer consultant with experience in process control, digital design, and product development.

The XE0002 DAA is available from:
Xecom, Inc.
374 Turquoise St.
Milpitas, CA 95035
(408) 9456640

The V8600 Voice Synthesizer is available from:
RC Systems, Inc.
121 West Winesap Rd.
Bothell, WA 980 12
(206) 672-6909

The following is available from:
Circuit Cellar Kits
4 Park St.
Vernon, CT 06066
(203) 875-275 1
Fax: (203) 872-2204

1. SSI204 DTMF decoder chip $14.50
2. RTC52 BASIC-52 controller kit . 589.00
. . $15.00
4. RTCIO-KI Expansion board kit w/8255 PPI and b-bit ADC .. $79.00
5. RTCIO-K2 Expansion board kit w/8255 PPI, 8-bit ADC, DC-to-DC converter, and battery-backed clock/calendar $119.00
6. RTC-PROTO Prototyping board . . . $39.00

Prices do not include shipping.

IRS

413 Very Useful

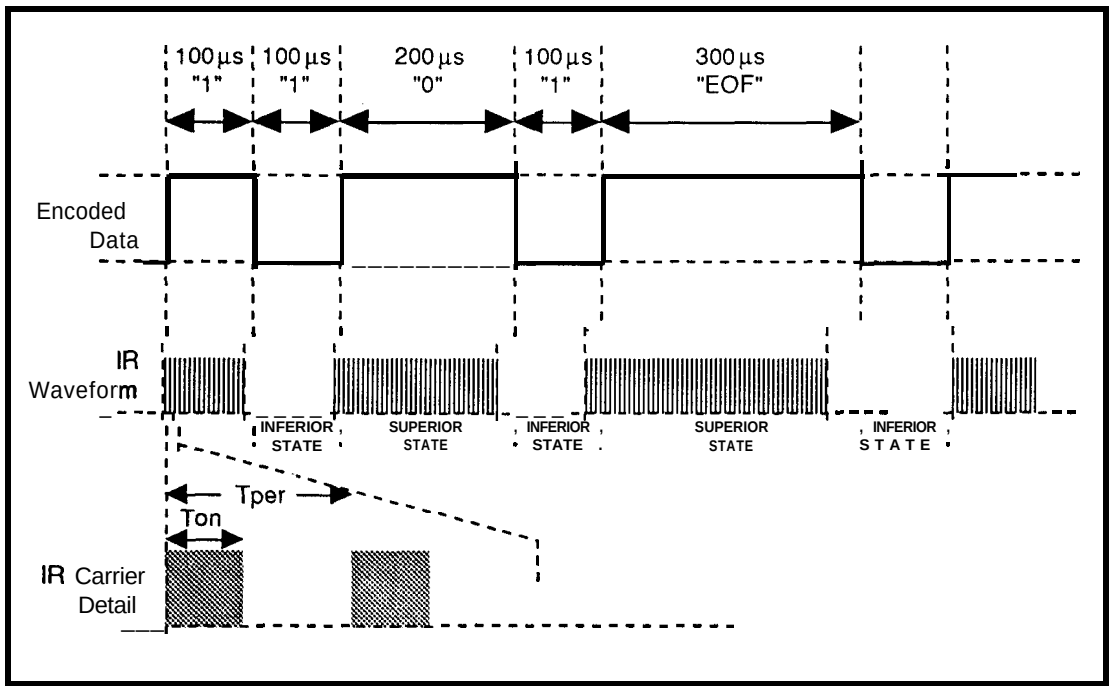


Figure 3—*The Infrared specification uses bursts of IR light modulated at 100 kHz for an effective data rate of 10,000 'one bits' per second.*

shades, check the security system, and lock the front door.

Likewise, "smart" light switches can be placed around the house that can have any light in the house assigned to them. Flip a switch next to the front door and you could be greeted with the outside lights plus the lights in the living room all coming on at once. If you would instead prefer to have the kitchen light come on when that same switch is flipped, it's a simple matter of retraining the switch.

CEBus devices communicate over one (or more) of several physical media, including power line, twisted pair, infrared, radio frequency, coax, and (eventually) fiber optics. Routers or bridges are used to usher messages from one medium to the next, so you don't have to worry about what device is plugged in where when you issue a command.

On the technical side, CEBus is modeled after the ISO/OSI seven-layer network definition. A number of the layers simply pass information untouched since the functions defined for those layers either don't apply or are performed to some degree by a different layer.

At the top is the Application Layer, where CAL (Common Application Language) is used to "program" devices to give them their individual personalities. Tables have been set up which contain entries for all common household devices including stereos, telephones, thermostats, clocks, and so on, plus common commands that might be given to those devices, such as volume adjust (which would equally apply to light level and temperature level), lock or unlock, on or off, and so forth. There is plenty of room for the tables to be expanded as new devices come to market.

Under the Application Layer are the Network, Data Link, and Physical Layers (Presentation, Session, and Transporta ren't used). As you go from the top down, each layer is responsible for one level of the network communication. Messages are passed between the layers using well-defined functions and are otherwise isolated from each other.

In general, the Network Layer is responsible for making sure packets are sent up to the Application Layer in the proper order. When they come down from the Application Layer, the Network Layer adds the proper routing information.

The Data Link Layer fills out the packet before it's sent with information such as source address, destination address, overall packet type, and a checksum for the whole thing. When going in reverse (a packet has been received and is going up the ladder), the Data Link Layer strips the same information, checking for packet integrity.

The Application, Network, and Data Link Layers were all described in detail in my article in the June/July 1990 (#15) issue. The CEBus committee has made a number of refinements based on comments received during the comment period and has re-released the spec, but none of the changes really impact what was presented a year ago.

At the lowest level is the Physical Layer. Up to this point, all the layers have worked identically regardless of which medium the device is connect to. The Physical Layer is where the actual signaling techniques come into play and is different for each medium. At this layer, "symbols" representing a "1" bit, "0" bit, end of frame (EOF), and end of packet (EOP) are defined in terms of "unit symbol times" (USTs). A "1" is one UST, a "0" is two USTs, an EOF is three USTs, and an EOP is four USTs. The length of one UST depends on the medium.

Each medium specification also defines a "superior" state and an "inferior" state. The symbols are encoded by alternating between the two states with the duration of each state depending on the symbol being encoded. For example, a "01" sequence would be encoded as a superior state lasting two USTs followed by an infe-

SPECIAL SECTION

CEBus Update: More Physical Details Available

Ken Davidson

It's that time of the year again. Flowers are blooming, people are sunning themselves at the shore, and CIRCUIT CELLAR INK is publishing its home automation special section. And what would that section be without a CEBus update? The CEBus committee has been hard at work over the last year revising preliminary specifications released for comment and putting the finishing touches on new specs. As a result, I'd like to bring you up to date on the latest in the CEBus arena.

I don't plan to cover in detail any portion of the spec which has gone largely unchanged since my last articles (CIRCUIT CELLAR INK #10 and #15). I'd start to sound like a broken record. Those who may have missed my previous articles should look them up for a complete background of CEBus. However, so as not to leave those unfamiliar with CEBus out in the cold, I'll give a brief overview of just what it is before getting to the good stuff.

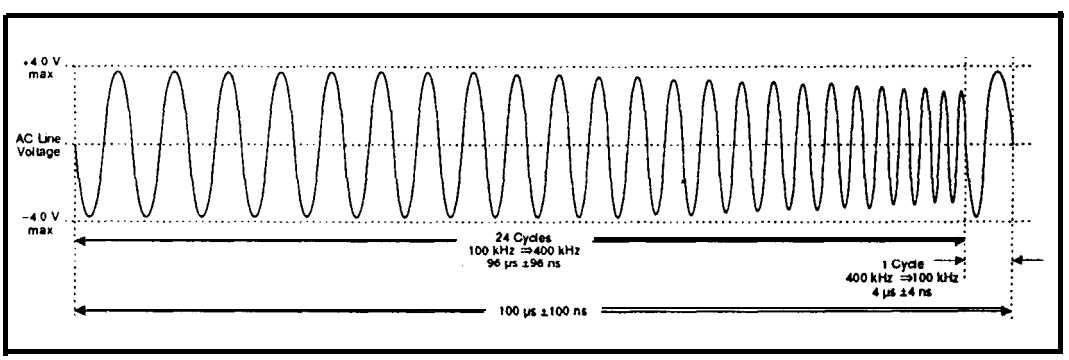


Figure 1 —*The waveform used by the new power line specification sweeps from 100 kHz to 400 kHz over a 100-microsecond unit symbol time.*

THE BASICS

CEBus is the Electronic Industries Association's (EIA) standard for home automation. It has been slowly (ever so slowly) evolving for about the last seven years as a result of the efforts of a committee made up of representatives from companies in the electronics industry. Its intention, once complete, is to allow a unified method of communication between virtually any electronic device found in a typical home.

Ideally, any CEBus-compatible product will be able to communicate with any other CEBus product regardless of who made them. The immediate advantages of such a system should be obvious: The herd of hand-held IR remotes clustered on the arm of your easy chair can be replaced by a single CEBus-compatible remote that not only controls your TV and VCR, but can also be used to adjust the room lighting and temperature, open the

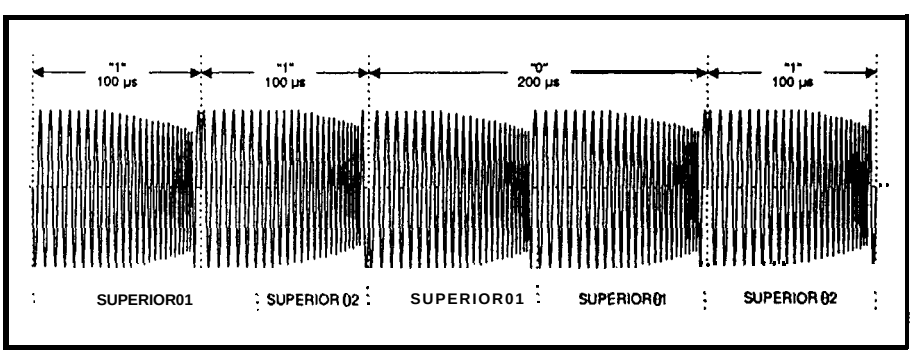


Figure *P-Data encoding is done by alternating between two states which differ only in phase (they are 180° out of phase).*

rior state lasting one UST. If this all sounds confusing, it will become clear when I get to the actual medium definitions and some examples.

The CEBus committee decided (wisely) that, rather than hold up the entire spec for all the Physical Layer definitions to be completed, they'd release the spec in stages as it was ready. The upper network layers plus the power line Physical Layer were released first, and a number of other Physical Layers have been released since. From this point on, I'm going to concentrate on the details of those Physical Layers.

To emphasize the same cautions I've expressed in previous articles, the details I present here are based on preliminary specifications which still must be finalized before they can be used for production designs. I also leave out a good deal of detail that, while is boring and useless in the context of a magazine article, is necessary when designing a product that conforms to the complete specification. If you have any plans for designing CEBus into a product, please contact EIA directly to get the complete text of the specs released so far and suggestions for conformance.

POWER LINE

If you've been following CEBus developments, you probably read about how the power line spec used a 120-kHz carrier and ASK signaling to achieve a paltry 1000 "one-bi t"-per-second data rate (a UST of 1 ms). Adding safeguards to prevent against false triggering of X-10 modules (which also use 120-kHz signals) reduced the effective data throughput even more.

Well, forget everything you've read. In last year's article, I made a comment that "perhaps a better method will be suggested during the comment period that will be enough of an improvement to prompt the committee to supplement or replace the proposed method." Fortunately, enough complaints were lodged during the comment period that the committee essentially threw away the proposed method and started again from scratch. What we've ended up with should assure quick acceptance of power-line-based CEBus products and will hopefully help stimulate its growth in the very lucrative retrofit market.

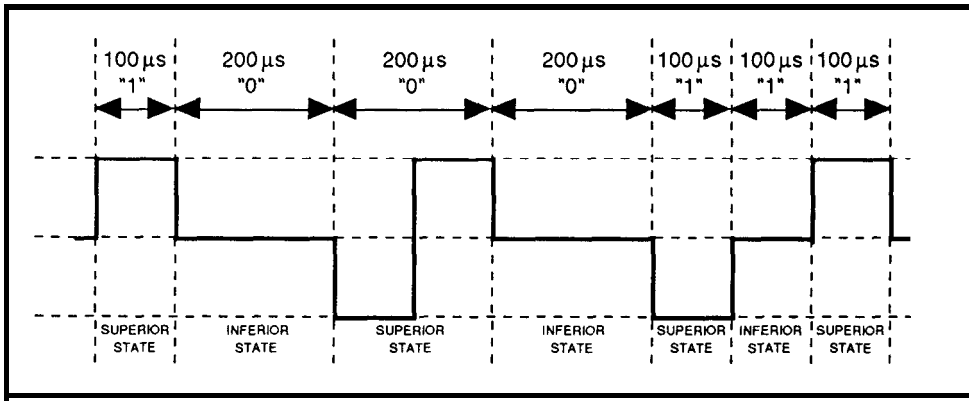


Figure 4-Twisted *pair uses a signaling scheme that can be confusing at first.*

The new method, originally proposed even before the first spec was released, uses a form of spread spectrum signaling. When dealing with a fixed-frequency carrier (e.g., 120 kHz), thatcarricrisvcrysusceptible to noise since a burst of garbage that happens to contain that particular frequency will mask out any real information. In the new spcc, the carrier is swept from 100 kHz to 400 kHz over a 100-µs UST. Since a broad spectrum of frequencies is being used, a random burst of noise is much less likely to cause errors. The shorter UST results in a data rate of 10,000 "one-bits" per second (which is consistent with most of the other media).

Figure 1 shows the waveform used. The sweep actually ends up back down at 100 kHz by the end of the period, as can be seen on the right side of the waveform. The decreasing amplitude at the higher frequencies is intended to reduce RF noise to keep the FCC happy.

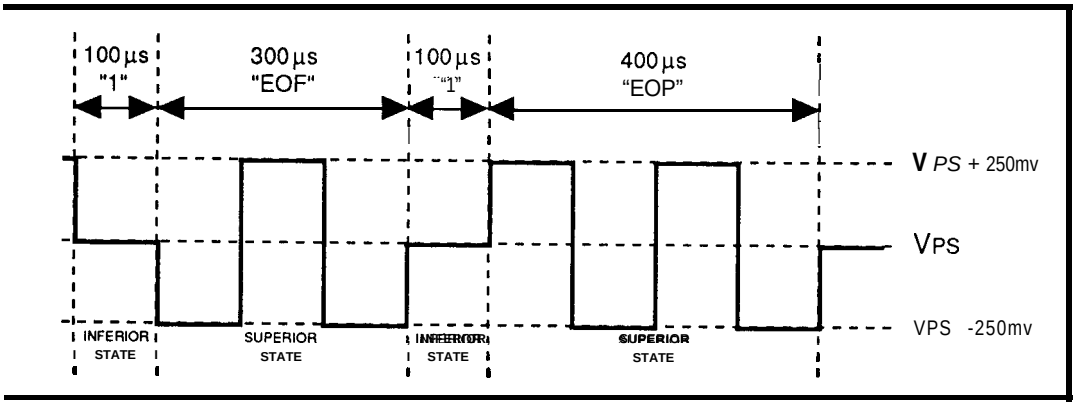
The new spcc complicates things a bit by defining not just one but two superior states: SUPERIOR01 and SUPERlOR02. The two are identical but are 180" out of phase. In addition, there is the normal INFERIOR state which is nothing but silence. (Note that this terminology is based on a spcc which was virtually days away from final rcleasc, so slightly different wordingmaybeuscdin the final spec.) During the preamble period, SUPERIOR01 and INFERIOR are used for encoding so that collisions can be detected (by listening during INFERIOR). Once the channel has been seized and data starts flowing, encoding is performed by alternating between SUPERIOR01 and SUPERIOR82 as in Figure 2.

Current plans are for Motorola to produce the first chips to support this scheme, with first silicon showing up about when you read this. Tests with working prototypes show the new method far outperforms the older method in terms of data integrity and speed with an array of typical power line noise induced on the line. Expect to see some very impressive power-line-based products showing up on the market within the next year or so.

INFRARED

The first new spec to be introduced after theinitial roll-outshowed up around August '90 and was for IR. Many observersexpect the hand-held IR remote control to be the main user interface to the house since it is already familiar to most, is easy to use, is very portable, is inexpensive, and can be used from virtually anywhere in the house with receivers placed at strategic locations. In addition, TVs around the house can be used as display devices to offer the user feedback.

The signaling method for IR is very straightforward. An IR carrier in the range of 850-1000 nm is modulated with a 100-kHz subcarrier (most hand-held remotcs these days use 40



kHz). The presence of this subcarrier represents the superior state while its absence represents the inferior state. The transmitted signal alternates between the superior and inferior states, with the length of those states representing the symbol being transmitted. Like the new power line, a UST of 100 µs is used for an effective data rate of 10000 "one-bits" per second (which probably explains why 40 kHz wasn't used). Figure 3 shows what the signal looks like.

There really isn't much more to it. The spec is the thinnest so far released simply because it's the easiest to describe. There are of course other details related to duty cycle and transmitter power, but I won't get into that here.

TWISTED PAIR

The next spcc to be released came about two months later and is for twisted pair. This one is going to be important whendealingwith devices such as telephones, intercoms, and even thermostats and motion sensors. These are all items which have traditionally used a single twisted pair to communicate through the house, and they'll likely stick to that medium.

There are two key sections of the spec that I'll concentrate on: one that defines what the signalson the wires look like, and the other that defines the wire topology and connectors.

All the CEBus media technically have a control channel and a data channel. On both power line and IR, only the control channel is defined in the first release of the specs. A data channel for each may come at some time in the future. With TP, though, both control and data channels are defined.

The TP control channel uses a 100-µs UST for a data rate of 10000 "one-

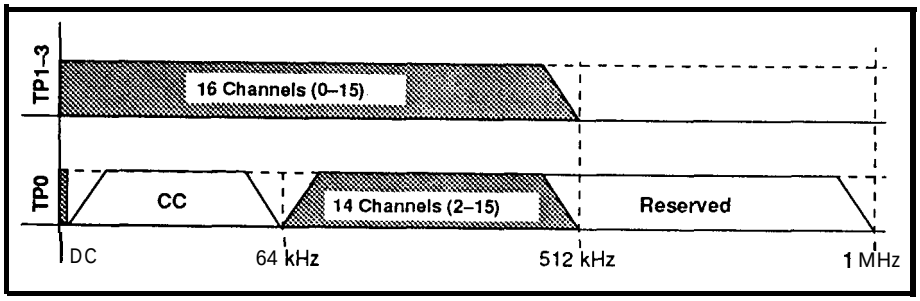


Figure 5a—*TP0 consists of DC power, a control channel, and 14 data channels. The other three pairs consist of 16 data channels each.*

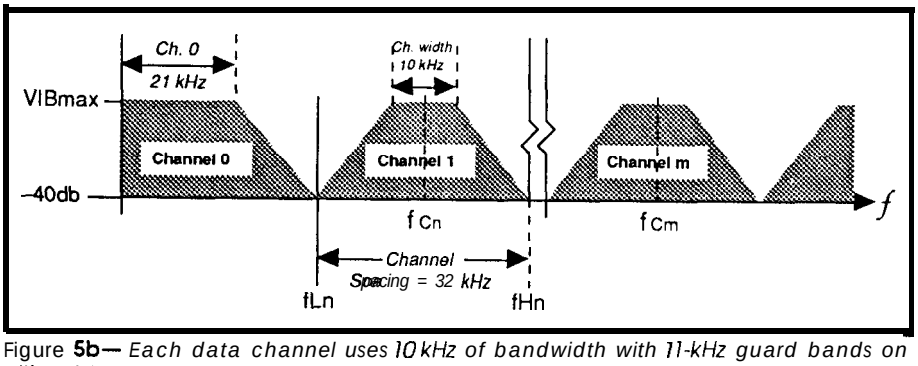


Figure 5b—*Each data channel uses 10 kHz of bandwidth with 11-kHz guard bands on either side. Adjacent channels may be combined when greater bandwidths are needed.*

bits" per second. Rather than use discrete signal levels to denote the superior and inferior states, transitions between levels are used to denote each state. A transition once per UST denotes a superior state while the lack of a transition denotes an inferior state. As with the other media, signaling alternates between superior and inferior states with the duration of each state representing the symbol being sent. The example in Figure 4 shows it much better than I could ever hope to explain it.

The transitions occur around the average DC level on the wire pair, which is actually the power supply for devices that wish to draw power from the bus. The signal level varies between the power supply voltage, the voltage plus 250 mV, and the voltage minus 250 mV. The power supply voltage can be between 16 and 18 volts depending upon the load.

The data channels are frequency multiplexed onto the same twisted pair. The control channel is defined to occupy a region in the frequency spectrum from DC to 64 kHz. Fourteen data channels are then defined starting at 64 kHz and progressing up to 512 kHz in 32-kHz blocks (10 kHz of usable bandwidth with 11-kHz guard bands on either side). When there is more than one wire pair installed (more on that in a moment), TP0 con-

tains the power supply, control channel, and fourteen data channels. The other pairs contain sixteen data channels each, starting at DC and going up to 512 kHz. Figure 5 shows how the channels are set up.

You say 10 kHz of bandwidth isn't enough? The spec allows for channel concatenation for such situations. When two adjacent channels are combined, you end up with the 10-kHz normally associated with each, plus the guard bands that would normally separate them, for a total of 42 kHz. Concatenating four adjacent channels would result in 106 kHz of bandwidth.

On the wiring side of things, the specification offers a number of options; which one you use depends on your setup and type of equipment. A full-blown CEBus-only system consists of four twisted pairs running to each room from a central location, resulting in a control channel and 62 10-kHz data channels. The connector, shown in Figure 6, is an &position RJ-type jack.

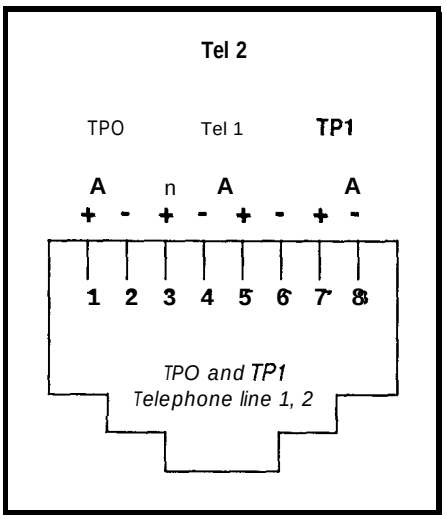


Figure *6-A single wall jack can be used by both CEBus-compatible devices and old-style* telephones.

For those instances where you still have conventional telephones, but want to add CEBus twisted pair, the spec calls for two pairs to be used for CEBus and two pairs for use by conventional telephones. The same 8-position jack is used, with the CEBus connections on the outer pins and the telephone connections on the inner pins where they would be in any conventional installation (see Figure 7). CEBus devices with 8-pin plugs can be attached and will use TPO and TP1. Regular telephones with 6-pin plugs will also plug into the same jack, but will only use the inner two (or four) wires, leaving the four outer CEBus wires untouched.

In the last configuration, conventional 6-position telephone jacks may still be used for conventional telephones, but since CEBus devices will have 8-pin plugs, they can't be plugged into the 6-position jack (which would be pointless anyway).

There is no requirement as to how many pairs must be run in an installation. A single pair may be all you have already installed and may be sufficient to handle the load. Any number of pairs up to four may be used.

COAX AND RF

There isn't a lot to report here. The last schedule I saw calls for both proposed standards to be ready at about

the time you read this, but since most of the others have come in behind schedule (IR and TP were exceptions, though), don't expect to see these two for at least a little while.

It really doesn't matter if these two take a bit longer, though. Coax will likely be used mostly in new construction or remodeling, so isn't as important in the retrofit market. I see RF as being important in the long run, especially-in retrofits, but not-right away. Power line and IR will be the big two, with twisted pair coming in third. With these three in the bag, manufacturers will finally be able to start making products with built-in CEBus.

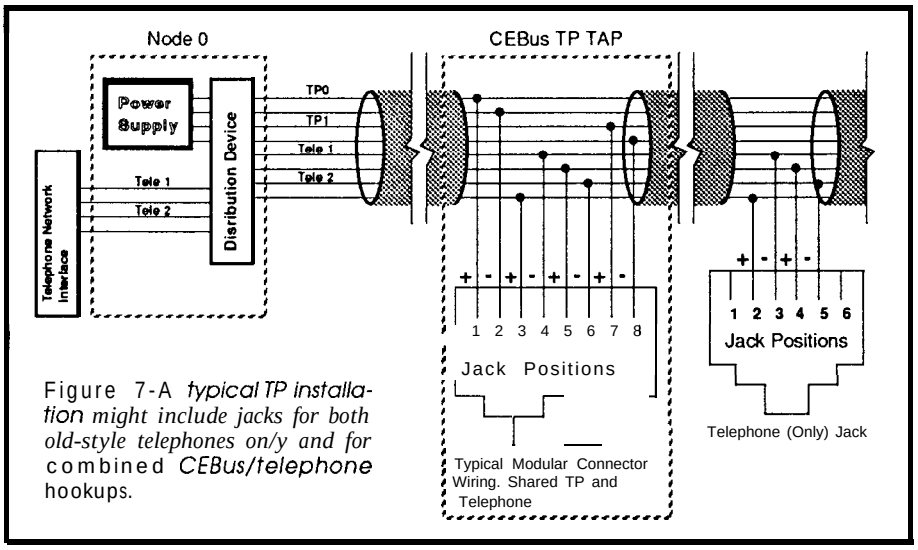


Figure 7-A *typical TP installation might include jacks for both old-style telephones on/y and for* combined *CEBus/telephone* hookups.

BRIGHT IDEAS

So where are we with CEBus? We're a lot farther along than we were a year ago, but not as far as I'd hoped. It's still going to take a few years before you see CEBus built into products,and thecapability will likely start at the high end and work its way down in the product line. Just look at MTS stereo in TVs and antilock brakes in automobiles. I've been hearing about some nifty products on the drawing boards (none that I can talk about, though), so I'm looking forward to the next few years with a great deal of anticipation.

As for a development that is here today, though, Indianapolis Power and Light Company and PSI Energy have built what they call their "Bright Home." Cited as the nation's first CEBus demonstration home, it was officially opened on March 20, 1991, and will be available for six months thereafter for "tours by the media, consumers, companies, governmental agencies, and representatives from academia."

Companies supporting the project with fully functional prototype products include Johnson Controls, Panasonic, Somfy Systems, Sony, Square-D, Thomson/RCA, and Universal Electronics. The Bright Home was introduced in conjunction with the Ninth Annual International Energy Efficient Building Association (EEBA) Conference and Exposition and is claimed to be so efficient that it can be heated and cooled for $19 a month.

Maybea trip to Indianapoliscould be arranged soon.. .

Contact

Electronic Industries Association
2001 Pennsylvania Ave.
Washington, DC 20006
(202) 457-4975

Thanks go to Tom Mock and George Hanover of EIA for their continued support of **my efforts** *to keep* OUT *readers informed.*

IRS

416 Very Useful
417 Moderately Useful
418 Not Useful

SPECIAL SECTION

Echelon's Local Operating Network

Ken Davidson

Here's one for you die-hard Circuit Cellar fans who say you've been reading 'Ciarcia's Circuit Cellar" and CIRCUIT CELLAR INK for years: who is Echelon? If you said, "They make a CP/M replacement operating system that was used on the SB180 single-board computer Steve presented in the September '85 issue of BYTE," you would have been correct about four years ago. Echelon-the ZCPR3 people-closed up shop a few years back and passed the operating system on to others to take care of. Not long after they did, though, a new Echelon started making the news, but this new company had nothing to do with CP/M computers,

ISO Layer	LONTALK Protocol Services	Benefits
7. Application	Standard Network Variable Types	Ensures Compatibility and Interoperability
6. Presentation	Network Variables and Foreign Frame Transmission	Facilitates Use of LONTALK for Internetwork Gateways
5. Session	Request-Response Service	Guarantees that Desired Action has Occured
4. Transport	Acknowledged and Unacknowledged Unicast and Multicast	Reliabilty and Efficiency
	Authentication	Network Security
	Common Ordering Duplicate Detection	Elimination of Errors Caused by Noise and Lost Messages
3. Network	Addressing Learning Routers	Multimedia Networks, Easy Expansion and Reconfiguration
2. Data Link	Framing 16-bit CRC	Data Integrity
1.5 Media Access	Predictive CSMA with Optional Collision Detection and Optional Priority	Efficient Use of the Medium. Consistent Response Time Under Variable Network Loads, Immediate Network Access when Required
1. Physical	Twisted Pair, Power Line, Radio Frequency, Coaxial Cable, Infrared, Fiber Opbc Multiple Data Rates	Low-cost Installation on Multiple Media

Figure 1— Lon Works is *based on the ISO/OSI seven-layer network mode/.*

The new Echelon was touting a novelconceptinbuildingautomation: the local operating network, or LON (hence theircatchy name—EcheLON). They were promising many of the same ideas and features as the promoters of CEBus: a unified networking system that could be embedded into electronic devices that would let them talk to each other via a variety of communicationsmedia.Regardlessof who made the device, they would all speak the same language.

Echelon's approach was somewhat different from EIA's, though—spend enough R&D money to develop a complete system solution in a very short time, then spend even more money on a slick marketing campaign to sell it to the execs of the major manufacturers. Once you comer the market and everyone is licensing your proprietary scheme, you'll be able to pay back the investors and finally make some money.

I can't say that's a bad approach. I might even be interpreting things incorrectly. Only time will tell if they can get enough big companies to jump

on their bandwagon to make it all fly. Instead, let me try to give you an idea of what all this talk of LONs, neurons, LonTalk, and LonWorks is all about.

THE WORKS

All the elements in an Echelon-based system are known collectively as LonWorks. The major elements include the LonTalk protocol, Neuron chips, LonWorks transceivers, and a LonBuilder developer's workbench. Together, a complete network can be configured and designed into end products.

Like CEBus, LonWorks is based on the ISO/OSI seven-layer network model. Figure 1 illustrates what each of these levels is supposed to do and some Echelon-claimed benefits of each. These upper network layers are known collectively as LonTalk. A quick comparison of each of the layers and their functions will show a great many similarities to CEBus.

One big difference between the two is the use of network variables and Standard Network Variable Types (SNVTs) by LonTalk at the Application Layer. An SNVT definition consists of units, a range, and an increment. Some examples include a variable type of "temperature" consisting of Fahrenheit units, a range of ±3200, and an increment of 0.1 degree; or relative humidity with units of **percent, a range** of 0-100, and **an increment** of 1/256%. The vast majority of applications can be specified using SNVTs, but user are free to define anything they need.

Object-oriented concepts play a big role in the interoperability of the network. Nodes are thought of as objects, with the network variables as the object's inputs and outputs. Links between objects are made between inputs and outputs of the same type.

THE NERVOUS SYSTEM

Details of how the seven layers actually operate are very sketchy since Echelon wants to sell you their chips and development systems. At the core of most LONs is going to be one of Echelon's Neuron chips. The **neuron**

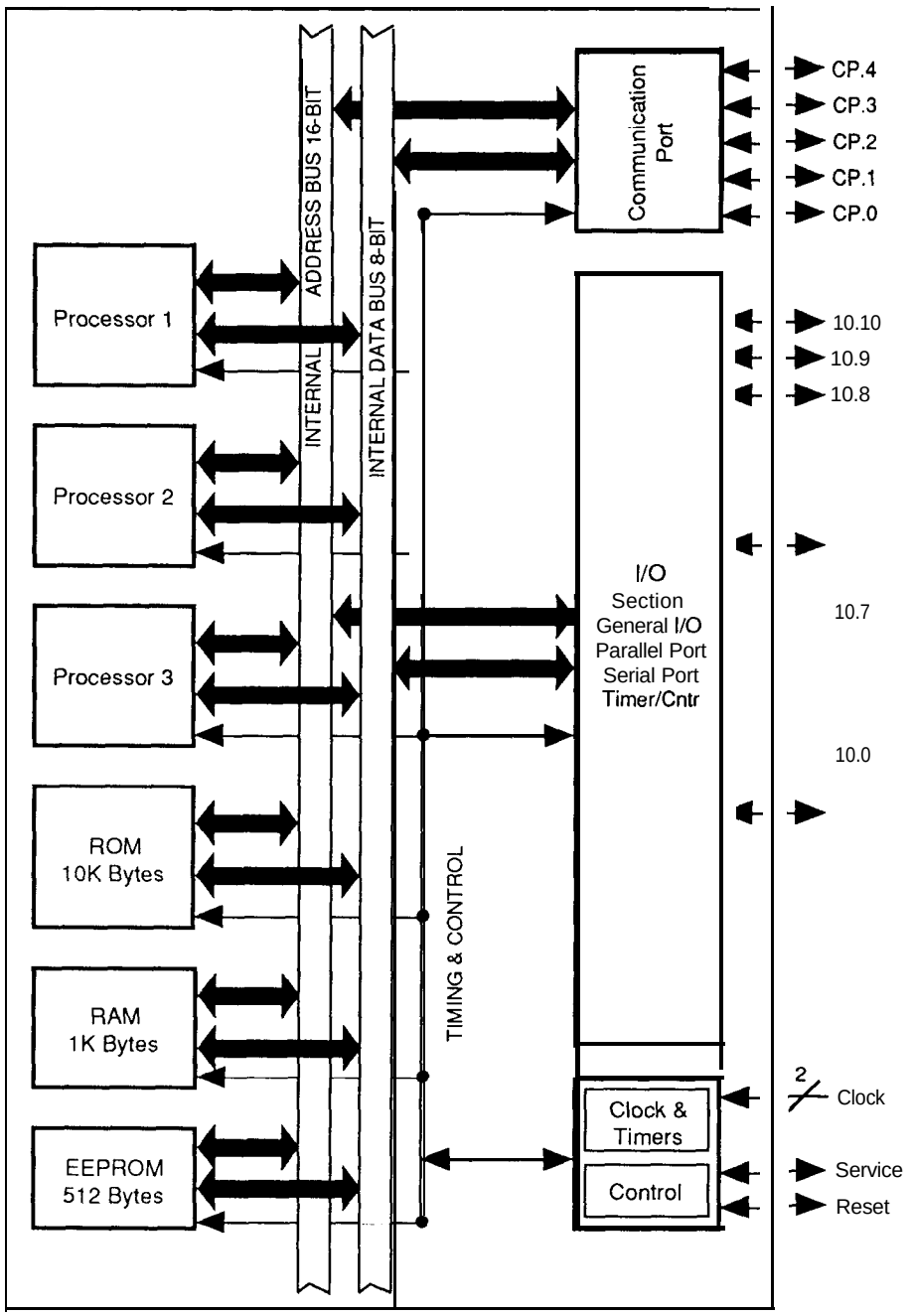


Figure 2—*The* **Neuron 3** *120 implements all but the Physical Layer on a single chip.*

implements all the layers except the Physical Layer, which I'll get to in a bit. There are currently two nearly identical chips available.

The Neuron 3120 is a complete self-contained chip as shown in Figure 2. It contains three microprocessors: two dedicated to LonTalk protocol processing and one dedicated to the node's application program. Also on the chip are 11 I/O lines, a programmable 16-bit counter/timer, 10K of masked ROM, 1K of RAM, 512 bytes of EEPROM, a 5-pin communications port for talking to the Physical Layer, a 48-bit ID unique to the chip, and assorted circuitry for doing wake up, watchdog timing, and so on. Note that the 48-bit chip ID translates to over 2.8 x 10^{14} IDs, so they're not likely to run out of IDs very soon. The chip comes in a 32-pin package.

The Neuron 3150 is identical but adds a second 16-bit counter/timer, another 1K of RAM (for a total of 2K), and requires external ROM. Up to 64K of external memory may be added with 42K going to the node application program. The package size increases to 64 pins.

Both chips are programmed in what is known as Neuron C; standard ANSI C with extensions added to support object-oriented programming, network variables and SNVTs, and some other features to make real-time processing easier.

Both chips are being manufactured by Motorola and Toshiba; the 3150 is supposed to be available in November while the 3120 won't be ready until the first quarter of '92. Motorola's part numbers are MC143120 and MC143150; Toshiba's are the TMPN3120 and TMPN3150.

The LonBuilder Developer's Workbench is available to aid in the development of LON nodes. Just be sure you're really serious before asking any prices, though.

GET PHYSICAL

At the Physical Layer, LonWorks Transceivers are used to communicate with the medium connected to the node. Again, as in CEBus, six media are defined: power line, twisted pair, infrared, radio frequency, coax, and fiber optic. Unfortunately, the physical layer is another area where details are hard to come by. Power line is defined as running at 9600 bps, RF at 5 kbps (30 feet indoors, 150 feet outdoors), and twisted pair at 78 kbps (4500 feet with 64 nodes) or at 1.25 Mbps (1500 feet with 64 nodes). With twisted pair, one pair is used for data while a second pair is used for power (when supplied to the nodes).

And that's about it. I've been able to find nothing on the other media, no chips, and no details. "Transceiver evaluation boards" for the three media mentioned above are supposed to be available, bu t you'll need good luck or great connections designing anything that has to make it into production soon. The modules also range in price from $400 for the twisted pair module up to $1500 for a single power line module.

Finally, the LonBuilder Developer's Workbench is available to aid in the development of LON nodes. There are complete tools available including an integrated development environment, a developer's kit that contains a Neuron C compiler and debugger, a network manager, and a protocol analyzer. Just be sure you're really serious about all this before asking **any** prices, though (the "starter kit" consisting of a development station, two neuron emulators, and neuron C goes for $14,965!).

BOTTOM LINE

Anyone who has seen Echelon's slick advertising or the video of the press conference in which the LON was rolled out or has attended one of their sales seminars would probably be quick to say that the LON will soon be used withevery piece of electronics to enter the home, business office, and factory. The future is here today and the LON is ready to make it happen.

Now step back a minute.

We have a proprietary networking scheme that forces you to use specific chips if you want to stay compatible. "But at least they have chips," you say? At $10 per chip (which is what Echelon is quoting), not to mention the cost of the transceiver and all the glue, you're going to drive the cost of a simple light switch up to many times its current low-tech cost. That may be fine for die-hard home automation fans, but not for the mass market. Industry insiders I've talked with also agree that implementing the LON won't likely be a cheap proposition.

And yes, they have chips (soon), but only for the upper networklayers. Transceivers are only available in the

form of evaluation modules. Like I said before, things aren't even close to being ready to be put into a piece of production equipment.

Will the LON squash the CEBus efforts? I don't think so. CEBus has a good headstart on industry recognition, and the system will likely be cheaper to implement. The CEBus upper network layers are fairly stable, and chips probably aren't far off. The Physical Layers appear to be much more stable than Echelon's, and chips definitely aren't far off. Echelon has a jumpondevelopment tools,but much of what they have is similar to what AISI had for CEBus two years ago (too bad AISI went belly up about a year ago), so there's no telling wha t may be waiting in the wings.

CEBus is also an open standard, so if someone wants to develop their own chips and tools, they have all the information necessary to do so. I would think the guys upstairs signing the checks would be more willing to back an open technology than a proprietary one.

Echelon also seems to be primarily targeting the commercial building market rather than home automation people. In fact, at a recent seminar of theirs that I attended, the lecturer was at times making fun of and downplaying home automation. He stated that he saw the market divided into three areas. One was bubbled-packed devices you could get at Radio Shack that would have limited functionality. The next would be the do-it-yourselfcrs. He described a typical member of this group as part of the "lunatic fringe" who now has a handful of X-10 devices and a computer who "can't get into too much trouble withasmallnetwork."Thethirdgroup consists of professional installers who know what they're doing.

This same lecturer also contradicted himself several times and told some outright lies about CEBus in an effort to downplay other players in the market.

Echelon has a short list of companies already backing their technology, including AT&T, Johnson Controls, Leviton, and Sony. However, all of the companies I just mentioned are also very active on the CEBus committee, so the big boys are keeping all their options open, Indeed, Echelon itself rarely misses a committee meeting.

Only time will tell who comes out ahead in the battle of home automation technologies, but my money is still on CEBus.

Contact

Echelon Corp.
4015 Miranda Ave.
Palo Alto, CA 94304
(4 15) 855-7400

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

4 19 Very Useful
420 Moderately Useful
421 Not Useful

DEPARTMENTS

The Furnace Firmware Project Concludes:

Hard Data For Home Control

The classic recipe for rabbitstew beginswith "Catch a rabbit." The outdoor temperature has been between -10 and +40 degrees Fahrenheit during the last few days, the furnace has chugged along as you'd expect, and my Furnace Firmware was busy capturing and logging data. The rabbit is in hand...

During much of the last year I have presented sections of the Furnace Firmware project, a project that monitors my furnace performance. The list of topics resembles a firmware catalog: liquid-crystal display drivers with ANSI cursor control, keyboard scanning, a piezo beeper driver, analog voltage inputs, fixed-point math, the whole thing written in both C and assembler code. (If you did not learn something new along the way, you ought to be doing some writing of your own!)

In any event, the question is whether the data from the furnace

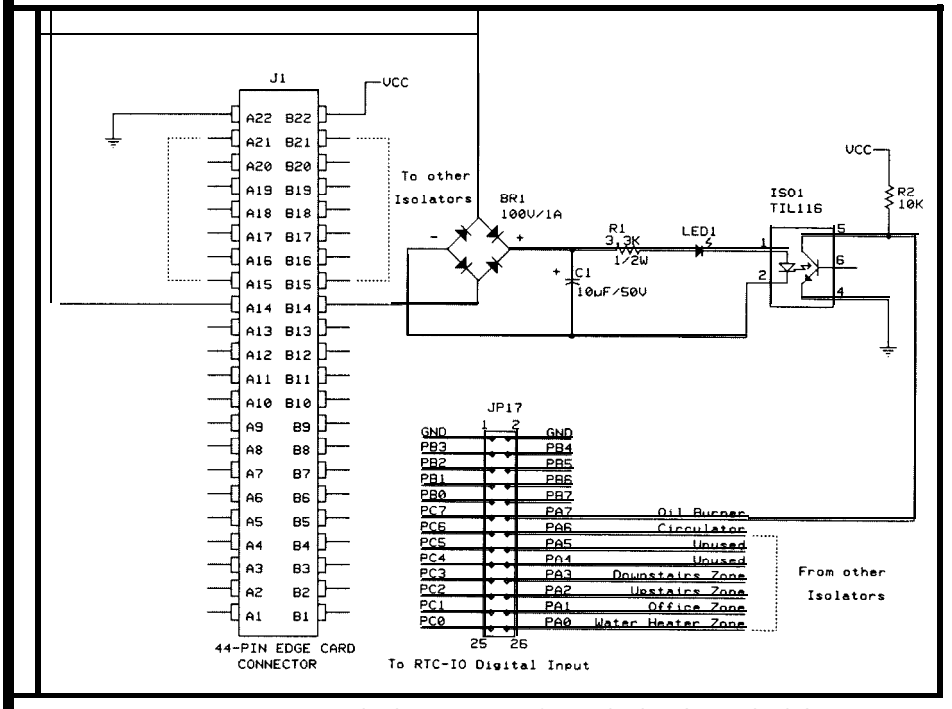


Figure 1 —*Optoisolated AC-to-logic converter for a single channel. Eight converter circuits are combined on a* **single** *board for this project.*

```
/*-------------------------------------------------------------*/
/* Format and display a menu tree                              */
/* Waits for response from keyboard                            */
void  MnuDoMenu(MENULEVEL *pMenuBase) {
unsigned char Key:
MENULEVEL *pThisMenu;
MENUITEM  *pThisItem;
BYTE ItemID;
   LCDSelCursor(LCDCURSOFF);      /* LCD cursor off for menu */
   pThisMenu = pMenuBase;             /* start at menu root */
   while (NULL != pThisMenu) {
      ANSIEraseScrn();                /* clear the display */
      cputs(pThisMenu->pTitle);       /* display the menu  */
      ItemID = 1;
      pThisItem = pThisMenu->Items;
      while  ((NULL!=pThisItem->pName) 66 (ItemID<=MAXITEMS)) {
         ANSISetCursor(ItemID+1,1);
         ConShowUInt(1,ItemID);
         putch(':');
         cputs(pThisItem->pName);
         pThisItem++;
         ItemID++;
      }
      Key = getch();
      switch (Key) {
      case ESC :
         pThisMenu = pThisMenu->pParent; /* back up a level */
         break:
      default :
         if  ((Key  > '0') && (Key < (ItemID+'0'))) {
            pThisMenu=MnuDoItem(&(pThisMenu->Items[Key-'1']));
         }
         else putch(BELL);            /* bad selection       */
      }

   putch('\n');                      /* force CR on serial    */
   ANSIEraseScrn();          /* flush used stuff from screen */
}

/*-------------------------------------------------------------*/
/* Execute a menu selection                                    */
MENULEVEL *MnuDoItem(MENUITEM *pItem) {
   if  (NULL  != pItem->pFn){
      (*(pItem->pFn))(pItem->Param);
   }
   return pItem->pNextMenu;
}
```

Listing 1 b—*The main program calls this routine with a pointer* to **the menu** *structure defined above. The code displays each menu and submenu, then executes the action routine for the selected item. Control returns to the caller after the action routine is finished* or *if the Escape key is* **pressed.**

wave bridge converts the AC signal to pulsating DC, which is smoothed by **the filter** capacitor. Current flows **from** the capacitor through a current-limiting resistor to the optoisolator's LED and a visible LED. Although the firmware displays the active zone, I've always liked having eyeball verification for input signals. ..and the LED draws no logic power from the RTC system.

The zone valves use 24 VAC, but the circulator pump and burner motor run on 120 VAC. Rather than design a line-voltage isolator, I took the easy way out by connecting ordinary **12-VAC** doorbell transformers to the **120-VAC** motors. The **3.3k** resistors shown in the schematic allow enough current at either 12 or 24 VAC to **activate** the optoisolators. No **120-VAC** wires are exposed, as the transformers mount directly on the existing electrical boxes.

Despite the fact that the AC inputs were optically isolated and the analog voltages were powered from the RTC supply, there wereoccasional glitches that scrambled the ADC; each glitch jammed the ADC inputs at 3FF until I turned the power off. It turned out that the zone valve switches driving the circulator pump relay generated enough of an inductive kick to glitch the ADC inputs. A **100-nF** cap across each switch eliminated the glitches, but it took several days to convince myself that the 24-VAC switches were causing the problem instead of the **120-VAC** circulator!

The optoisolators produce a clean digital signal that is read through the **RTC-IO board's** 8255. because the bits change relatively slowly, the program can simply poll them several times each second and get entirely adequate resolution.

A MENU OF CHOICES

Although **DOS** prompt devotees swear by command-line interfaces, the fact of the matter is that menus are better for programs you don't use every day. Rather than remembering the choices and command syntax for everything you want to do, you just pick items from a self-explanatory menu. For a device I hope will run for a year at a crack with minimal intervention, menus were certainly the way to go.

Fortunately, menus are **easy** if you are already using C and feel comfortable with pointer notation. While this can be done in assembler, it is much, much easier in a higher-level language.. **.honest!**

Each menu item is defined as a four-member structure as shown in Listing la. The first member is the item name which will appear on the LCD. The second member is a pointer to a function which will be executed when this item is selected; the third member is passed to the function as a parameter. The final member points to the menu that should be displayed when this item is selected. If either pointer **isNULL** (zero) the corresponding action will not take place.

Menu level structures group the items, with up to three items per level, into a menu tree. Each level includes a title which will occupy the top LCD line, an array of three item structures as defined above, and a pointer to the previous menu level so you can back upthroughthemenus.Thethree-item limit arose because the LCD had only four lines.

Most menu systems use pointers to items in each level rather than the items themselves. I chose the latter strategy because the menus are short enough that the additional level of indirection didn't save much space or add much generality. If you have an

FIRMWARE FURNACE

Ed Nisley

makes any sense. In this column I will describe some AC power line hardware, explain the new menu and data logging code, then present the results so far. There were a few interesting traps along the way, and paying attention here may save you a few hours (or days!) on your own projects.

POWERFUL CONNECTIONS

To review: my furnace has four zone valves, each controlled by a thermostat. The zone valve opens when the corresponding thermostat calls for heat. A switch on each valve closes when the valve is fully open, which turns on the circulator pump to move hot water through the pipes to the heating zones. A thermostat within the boiler (the water never actually boils, but that's what the thing is called) controls the oil burner, which runs only when the circulator is active and the boiler water temperature is below the thermostat's set point.

The Furnace Firmware code runs on a three-board RTC system. An RTC31 board holds the 8031 CPU, 8K of RAM, 32K of EPROM, and the serial port. A slightly modified RTC-LCD board reads a matrix keypad and displays data on a 4 x 20 LCD panel. Finally, an RTC-IO board has a serial nonvolatile RAM chip, digital I/O, and analog inputs for temperatures at seven locations: the return end of each zone near the zone valve, at the boiler outlet, near the point where the zones branch off the main pipe, and outdoor air temperature.

Measuring the state of the thermostat switches, circulator, and burner posed a problem. Although I could use an RTC-OPT0 for AC-to-logic conversion, I decided to build optoisolators from my parts box because1 didn't want to add a fourth board to the stack. Figure 1 shows the converter circuitry for a single input and Photo 1 shows the perf board with eight identical converters. The AC inputs arrive through the edge connector and the outputs are cabled to the RTC-IO's 8255 input port through the ribbon cable header.

Each channel has a bridge rectifier and optoisolator IC, housed in a pair of six-pin DIPs. A single 14-pin socket hold the ICs, leaving the socket's middle two pins unused. The full-

```
#define MAXITEMS 3                          /* most items in a menu*/
typedef void (MENUFN) (unsigned Param); /* menu item function */
struct  menuitem{                           /* item within a menu */
   char *pName;                             /* item name          */
   MENUFN *pFn;              /* pointer to menu item function */
   unsigned Param;                        /* fn param or menu item */
   struct menulevel *pNextMenu;             /* pointer to next menu*/
};
typedef struct menuitem MENUITEM;
struct menulevel {                          /* complete menu level */
   char *pTitle;                            /* menu name          */
   MENUITEM Items[MAXITEMS];                /* items for this menu */
   struct menulevel *pParent;   /* ptr to parent of this item */
};
typedef struct menulevel MENULEVEL;

MENULEVEL  MenuSetup = {
    "Set up the System",
    (("Enter date/time",TmrSetClock,0,NULL}
    ,{"Calibrate sensors",SenSetCal,0,NULL}
    ,{NULL,NULL,0,NULL}
    },NULL
};
MENULEVEL  MenuDisplay = {
    "Select  Displays",
    {{"Sensor",SenSelZone,0,NULL}
    ,{"Control Trace",SenEnableTrc,0,NULL}
    ,{NULL,NULL,0,NULL}
    },NULL
};
MENULEVEL  MenuLogging = {
    "Logging Menu",
    {{"Enable/disable",LogEnableLog,0,NULL}
    ,{"Set threshold",LogSetThresh,0,NULL}
    ,{NULL,NULL,0,NULL}
    },NULL
};
MENULEVEL  MenuMain = {
    "Main Menu"
    {{"Set up the system",NULL,0,&MenuSetup}
    , {"Select displays", NULL,0,&MenuDisplay}
    , {"Control logging", NULL,0,&MenuLogging}
    },NULL
};
```

listing 1 a -*A pair* **of structures** *define each item and specify the items available in each menu display. Because the LCD has four lines of 20 characters each. the menus consist of a single-line tit/e and up to three items. Because the C* ***input*** *routines drive the serial port in parallel with the keypadand LCD, the menus will work equally well from a serial terminal with ANSI support.*

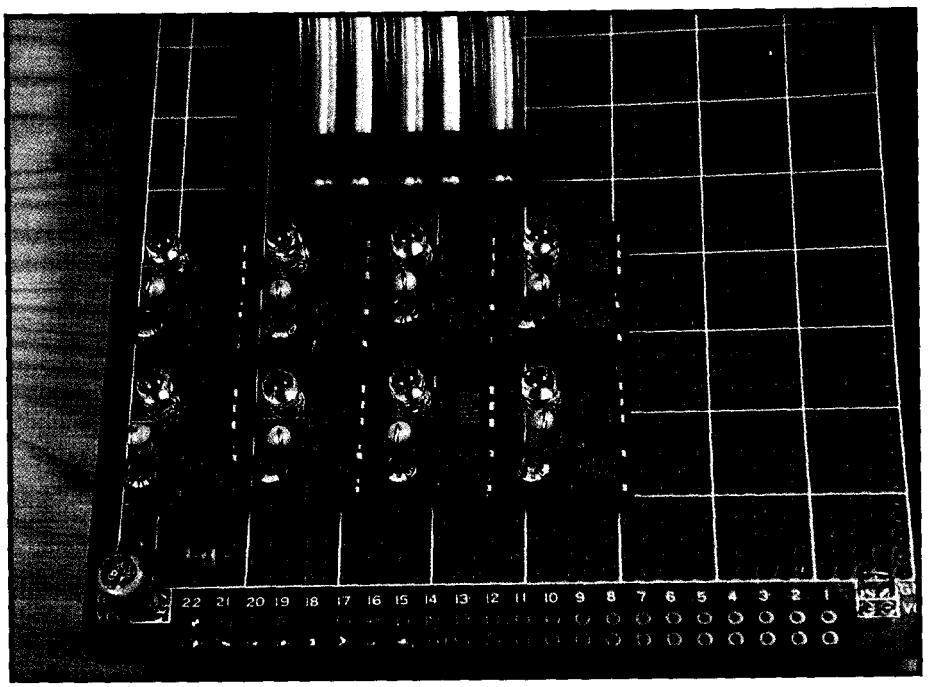


Photo 1 -Eight AC-to-logic converters are built on a single perf board for the Furnace project.

eight-line LCD you might want to put the items in a standard linked list terminated with a NULL pointer.

The menu handler shown in Listing lb displays the menu level title and the text from each of the three items, then waits for a keystroke. ASCII digits 1 through 3 select the corresponding menu item, while the Escape character backs up to the previous menu. Any other characters beep the piezo speaker and are discarded. Because the keypad and serial port are connected "in parallel" the keystrokes can come from either source.

Each item has both function pointer and a menu pointer so that selecting an item can execute the function and proceed to another menu. While the current code doesn't use this feature, it can come in handy for some applications. The `MnuDo Item` function shows how this works.

Photos 2 and 3 show the normal display and the "Main Menu" in action.

DATA RECORDING

The Furnace Firmware project is intended to be a free-standing unit that processes raw data into totals for later examination. But it always helps to review the raw data before deciding how to process it; often you will see unexpected events that your initial code just wouldn't handle correctly.

An RTC31 system has very little space for data, so I decided to send a line of ASCII text to the serial port whenever an "interesting" event took place; a standard PC communications program could then capture the data to disk for later analysis. Although a binary format would be somewhat faster, readable text required no tricky programming on either end of the RS-232 wire.

For starters, I defined "interesting" as a change in any bit input or a temperature change over 2 degrees. Figure 2 shows the results of a few minutes of monitoring; a 24-hour trace during the heating season pretty much fills up a 360K diskette.

The A/D converter glitch I described earlier appeared quickly. It was obvious that the glitch occurred when the circulator went off, but it took some sleuthing and experimenting to find the culprit. A more clever data reduction scheme would have suppressed the evidence by quietly-throwing out the "obviouslybad" data points.

Poring over page after page of numbers, however, is not a good way to spot trends, so (after sanity checking the results) I loaded the data into an Excel spreadsheet to get some graphs. One look at Figure 3 and I

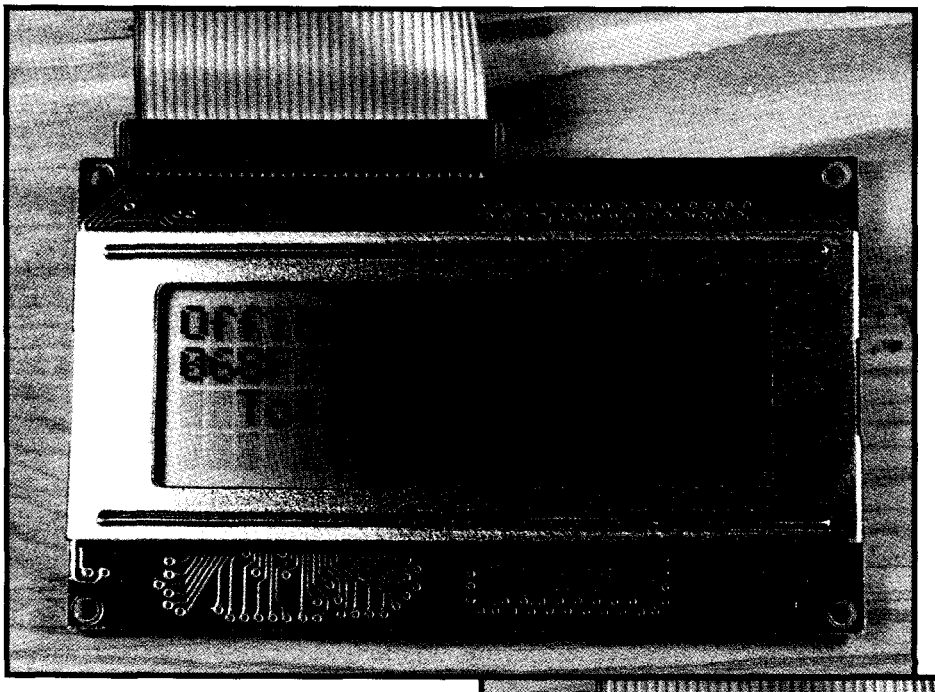


Photo 2—The Furnace Firmware Project's normal display.

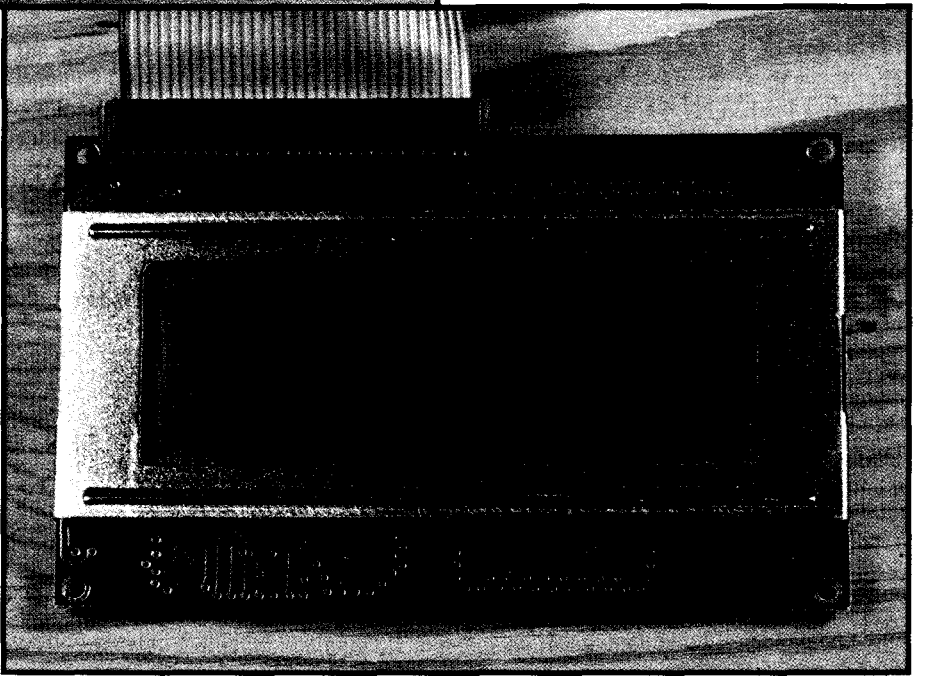


Photo 3—*The menu system for the project was designed with the limitations of the* LCD (e.g., *only four* lines *ofdisplay) in mind.*

think you'll agree a graphic is worth a thousand characters.

The six lower traces in Figure 3 show the six input bits; the Water Heater zone was inactive during this whole recording. The Downstairs thermostat closed at about 2:13 AM and the Circulator and Burner turned on simultaneously one minute later when the zone valve closed.

The top six traces record zone, boiler, and zone branch temperatures. Although the Water Heater zone never went on, the temperature shows the effect of the other three zones passing hot water within inches of that LM35 sensor, heating the water just behind the zone valve. Notice that when a zone valve opens, the temperature dropsdramaticallyasthecooler"room temperature" water flows past the sensor.

One of the assumptions I made when starting this project was that I could measure the flow rate of each zone by noting the elapsed time between corresponding temperatures at two different locations. The boiler and zone branch temperatures (top two traces) illustrate how this works in practice, as those two sensors are separated by 25 feet of l-inch diameter copper pipe.

When both the downstairs and upstairs zonesareactive, the temperature peak occurring when the burner shut off at 2:31 took five seconds to pass from the boiler to the branch sensor. There is almost exactly one gallon of water in that length of pipe, so the corresponding flow rates is about 12 GPM (gallons per minute).

The flow length of each zone can be derived in a similar manner; the 2:31 zone branch peak occurred 72 seconds later in the upstairs zone and 107 seconds later downstairs. I measured 104 and 121 feet of 3/4" pipe, respectively, so the flow rates are about 2.0 and 1.6 GPM, respectively.

Obviously, 2.0 plus 1.6 does not equal 12, even using New Math! The reason is instructive and gives a good example of why you should not take "obvious" data at face value.

GETTING REAL DATA

Recall that the program sent an output record whenever the temperature changed by more than two degrees. This means that the temperature could rise by 1.9 degrees after producing a record, then fall 3.99 degrees without producing another record. The two points on either side of the 2:31 AM peak are nearly three minutes apart; it should be evident that the result of subtracting two such times won't have good resolution.

This is known as the "small differences of large values" error. Put differently, you do not weigh a grain of rice by weighing a truckload of rice, throwing out one grain, and reweighing the truck.

After eyeballing another pile of records, I modified the code to record each active zone along with the boiler outlet and zone branch temperatures at uniform timeintervals. Even though I omitted the zone name to reduce the torrent of data, a trace still fills a diskette in a few hours.

Figure 4 shows the results from the downstairs zone. Figure 5 is the same data with the branch and zone curves offset to align them with the boiler outlet. Ignoring the "startup" transients, the match is quite good with the downstairs zone offset by 80 seconds and the zone branch by 20 seconds.

The downstairs zone holds about 3.7 gallons of water, so the flow rate is about 2.7 gallons per minute. The zone branch section holds one gallon, so the flow rate there works out to 3.0 GPM. Given the inaccuracies in measuring the zone's overall length, agreement within 10% is reasonable.

The temperature drop from the zone branch to the zone return is 6.1 degreesafter it reaches the peak. Combining that with the flow rate, the heat delivery into the zone is given by the formula I described in Issue 15:

$$\frac{BTU}{sec} = \left(\frac{ft^3}{sec}\right)\left(\frac{lbm}{ft^3}\right)\left(\frac{BTU}{lbm \times °F}\right)\left(T_{in} - T_{out}\right)$$

A flow rate of 2.7 GPM is 0.006 cubic feet/second, water's density is 62.4, and the specific heat is a nice, round 1.0; the result is 2.3 BTU/sec-

```
B08    04:24:14  01/23/91  ON Downstairs
T06    04:24:27  01/23/91  141.3    0281 h   Zone branch
Z02    04:24:48  01/23/91  136.1    02ABh    Valve=OFF Office
Z01    04:25:12  01/23/91  095.7    027Dh    Valve=OFF Water Heater
B40    04:25:13  01/23/91  ON Circulator
Z04    04:25:15  01/23/91  113.2    0291 h   Valve=ON Downstairs
Z04    04:25:18  01/23/91  110.6    026Eh    Valve=ON Downstairs
T05    04:25:18  01/23/91  147.3    02B8h    Boiler outlet
Z04    04:25:22  01/23/91  106.0    026Bh    Valve=ON Downstairs

<< many records omitted! >>

Z04    04:26:39  01/23/91  126.2    02AOh    Valve=ON Downstairs
204    04:26:43  01123791  120.9    02A3h    Valve=ON Downstairs
B80    04:26:43 01/23/91 ON Burner
Z04    04:26:48  01/23/91  131.5    02A6h    Valve=ON Downstairs
Z04    04:26:56  01/23/91  134.1    02A9h    Valve=ON Downstairs

<< many records omitted >>

Z03    05:02:23  01/23/91  113.1    0291h    Valve=OFF Upstairs
T05    05:02:38  01/23/91  158.0    02C4h    Boiler outlet
201    05:02:38  01/23/91  096.2    0280h    Valve=OFF Water Heater
B08    05:02:41  01/23/91  OFF Downstairs
T06    05:02:50  01/23/91  157.1    02C3h    Zone branch
Z04    05:02:53  01/23/91  150.1    02BBh    Valve=OFF Downstairs
Z02    05:03:28  01/23/91  154.5    02COh    Valve=ON Office
Z04    05:03:48  01/23/91  147.5    02B8h    Valve=OFF Downstairs
B02    05:04:09  01/23/91  OFF Office
Z04    05:04:43  01/23/91  144.0    02B5h    Valve=OFF Downstairs
T05    05:04:44 01/23/91  155.4    02C1h    Boiler outlet
TO6    05:04:44  0112391   154.5    02C0h    Zone branch
B40    05:05:09  01/23/91  OFF Circulator
```

Figure 2—*The* **first column** ***identifies the line as a Bit, Temperature. or*** Zone ***record. Ihe timestamp is captured when the event occurs because there maybe a delay*** **before** ***the record is formatted foroutputon theserialport. The raw hex value for each Zone and Temperature record proved useful while debugging the ADC input conversion routines.***

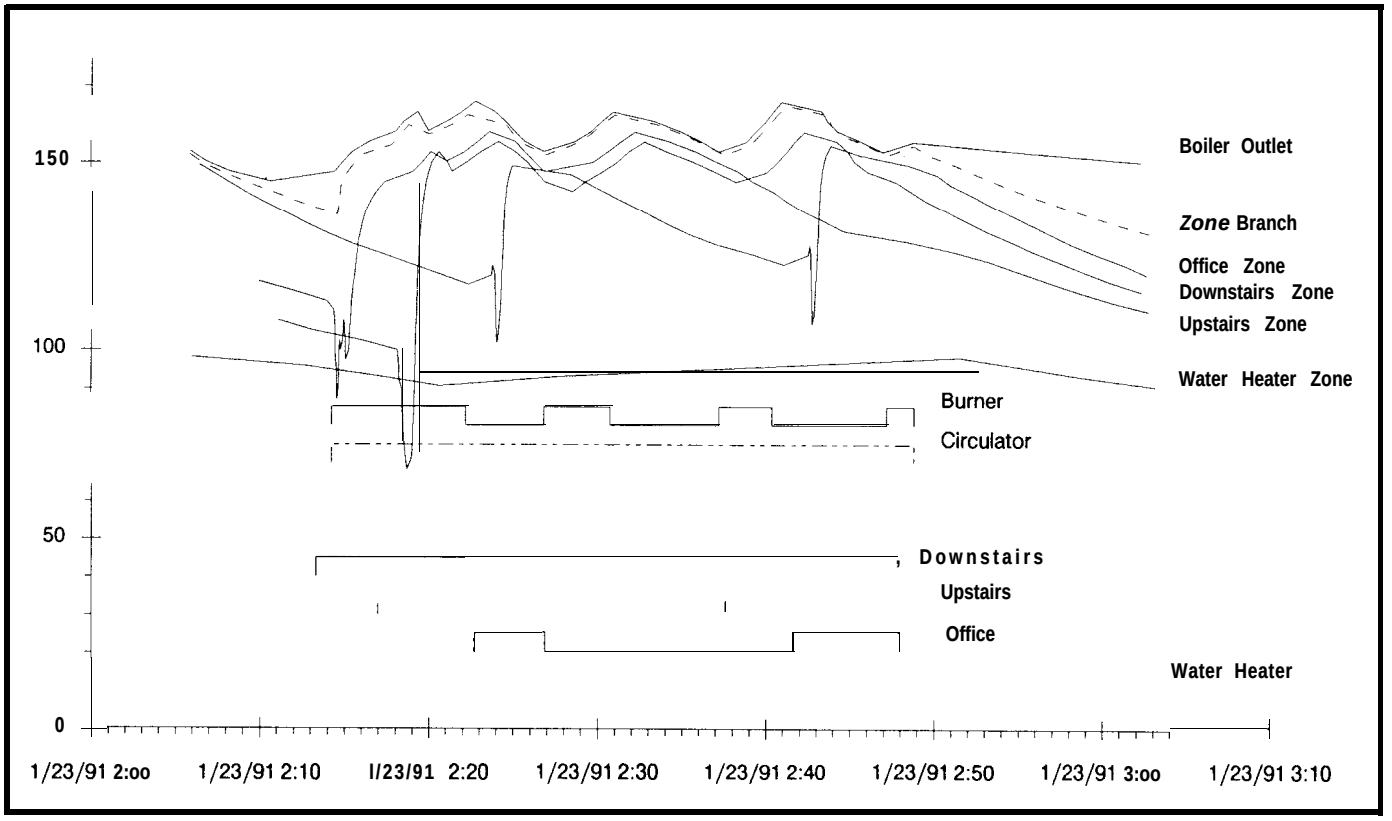


Figure 3—*Loading the raw data from the logger into a spreadsheet shows obvious Trends In temperature.*

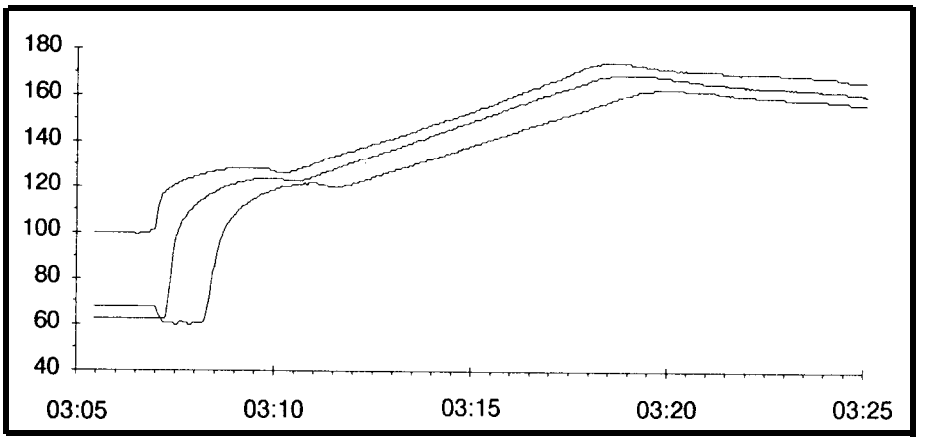


Figure 4—*The plotted temperature data from the downstairs zone. Excel (PS/2 version) was used to generate the graphs for this article.*

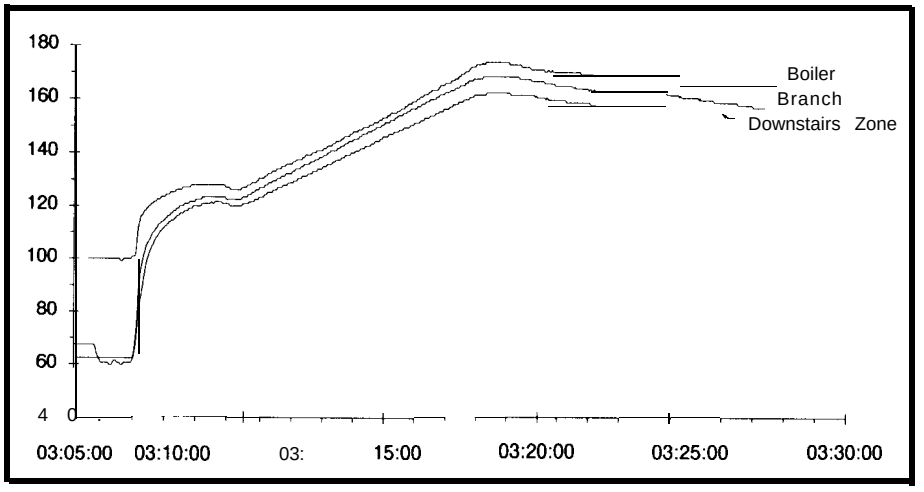


Figure 5—*The same data as in Figure 3 has been replotted with the branch and zone curves offset to align them with the boiler outlet.*

ond or, in more common terms, 8,000 BTU/hourdelivered to thedownstairs zone.

The sanity check for this is to measure the total amount of oil used and see how closely it matches the delivered BTUs. Rather than install an oil flow meter (please!), I'll do it the same way you compute gas mileage: measure the BTUs between fill ups. More likely than not some adjustments will be needed.. .

There is one puzzlement in Figures 4 and 5: the temperature drop from the boiler outlet to the zone branchis5.3 degrees. Because themass flow rate is the same through the entire zone pipe, it seems that nearly as much heat is going into the basement as the upstairs. Frankly, I find this

hard to believe, as 50 feet of radiator should be far more efficient at losing heat than 25 feet of pipe, so I'm going to try some insulation experiments and see what the truth is.

Actually, the heat in the basement isn't going to waste because I spend a lot of time down there playing with my toys. Were it not for the heat from that pipe (and the furnace, of course) I'd probably have to install a radiator. Imagine what sort of project that could turn into!

RELEASE NOTES

The Furnace Firmware project is essentiallycompletewiththiscolumn, so it's time for the final wrap-up..

After modifying the analog input code to use the 10-bit SDA0810 analog-to-digital converter, I changed the fixed-pointmathroutinesto work with three-byte values. The new format allows a dynamic range of ±32K with a fraction resolution of 1/256 (about 0.4%), which should be enough for most applications. As it turned out (I should have expected this!), the additional dynamic range was essential for scaling the raw input to degrees Fahrenheit. *[Editor's Note: Soffware for this article is availablefrom the Circuit Cellar BBS and on Software On Disk #21. See page 106 for downloading and ordering information.]*

The sensor scaling and display routinesare in `SENSORS.C51`, thecode to produce the output records is in `LOGGING.C51`, and the menu drivers are in `MENUS.C51`. `ANALOG.A51` changed to handle the IO-bit ADC routines, as did the code in `FIXMATH.A51` for the new fixed-point format.

The real-time clock on the RTC-IO board now produces an INTO interrupt once per second to resynchronize the 5-millisecond timer based on the 8031's Timer 0. Without this synchronization the two timers will drift slowly apart because of differences between the "one-second" rates produced by the 8031's 11.0592-MHz crystal and the MSM6242's 32.768-kHz crystal. The new code is in `TIMERA.A51` and the interrupt vector is in `LOGGER.C51`.

Although I will continue to twiddle the code for my own use, I don't plan to update the public versions any further, simply because the code is becoming so specialized that it doesn't warrant column space here. If you need more information, contact me through the Circuit Cellar BBS and I'll be glad to give you further advice and counsel.

And if you use any of this code in your projects, please drop me a note! Several readers tell me that bits and pieces of Furnace Firmware will soon show up in some fascinating applications.., what do you have cooking?

Ed Nisley is a Registered Professional Engineer and a member of the Circuit Cellar INK engineering staff. He specializes in finding innovative solutions to demanding and unusual technical problems.

IRS

422 Very Useful
423 Moderately Useful
424 Not Useful

FROM THE BENCH

Jeff Bachiochi

IR Communications

An Essential Link in the Chain of Control

The ability to communicate has been with us since prehistoric man. In some ways we haven't progressed much. A cave man's grunts could be easily mistaken as lust, fear, or even "I'm going out for some pterodactyl eggs honey, I'll be right home." When my wife says, 'I need some <insert item here>, so I'm going to the store. I'll be right back," the true meaning is not what it seems. On the surface, what looks like a simple act of replenishment, is actually cleverly worded secret code. Having broken the code, I now know this means, 'I don't really need any <insert item here>, but it's a good excuse to go shopping. Don't expect me home any time soon!"

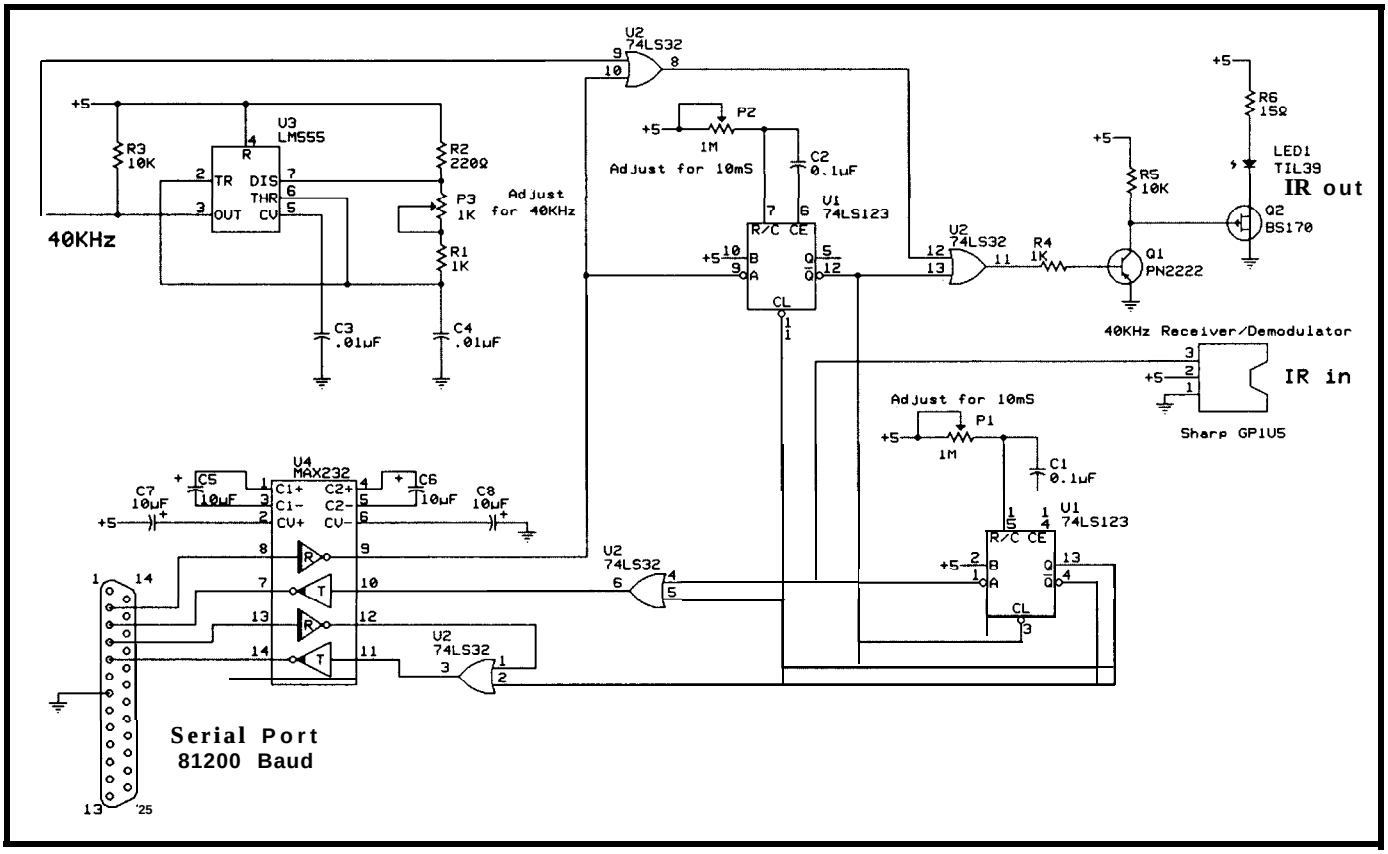


Figure **1 -Two-way** *serial communications between computers using IR is possible using readily available components.*

Talking with kids is not much different. For instance, if I ask, "Is your room clean?" "Yes!" comes a quick reply. It actually means, "I just cleaned it last month, however, now you'd need a bulldozer just to get to the other side." Am I one who should complain? Probably not. I've said more than once, "Sorry, I'm not hungry tonight." When I should have said, "I gorged myself on pizza today for lunch, 'cause I knew you were making meat loaf tonight."

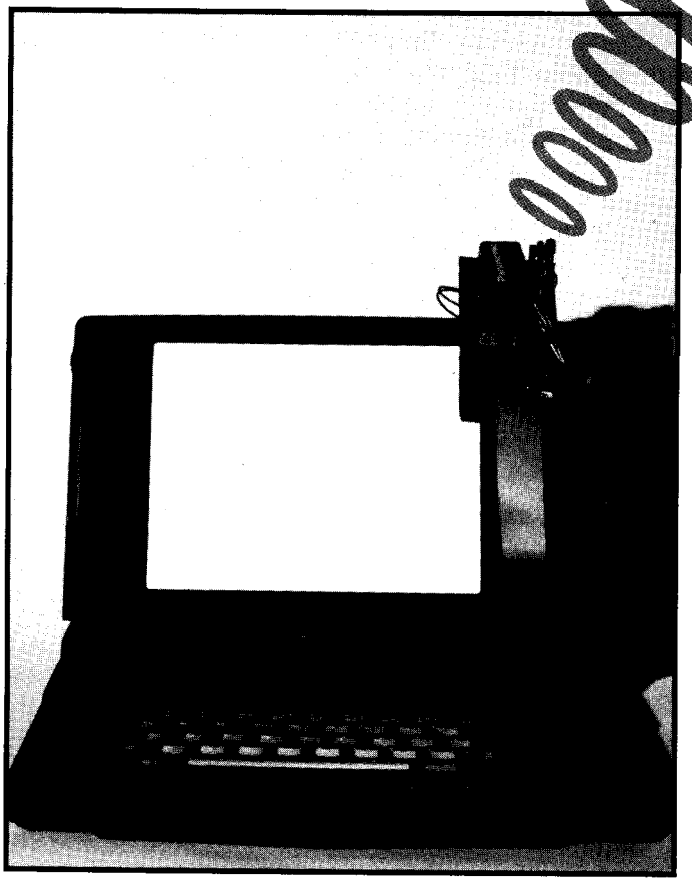
What would happen if our household appliances functioned on interpretation of our commands, a fuzzy logic, instead of merely black or white. We might wake up to only cold water from the shower because it heard us mumble something about how hard it was waking up in the morning, or have the television replace CNN with Mister Rogers just because we remarked on the amount of violence on TV.

HOME AUTOMATION, NOT ARTIFICIAL INTELLIGENCE

I am intrigued with the idea of machines being trained versus programmed, but at this stage I'm satisfied with remote sensing and control of my castle by command. I'll leave the AI alone for now.

It's been about a year since I first wrote about my home control system plans. At that time I was just starting an addition to my home. Now, nearly a year later, the carpenters haven't left yet! At least there is some good news: Since the builders were here throughout most of 1990, I can claim them as dependents on this year's federal income tax.

I originally planned on having some kind of input/output device in each room. These would all be networked (along with sensors and controls) to the master PC in the basement. Placing these I/O ports in the right spot has become a problem. The entryways to each room seem like the most logical place, but even these locations are not handy. It's like mounting your TV remote control on the wall and having to get up to change channels. Dumb! Right?



What I really want is a portable I/O device which can communicate over an IR link. The I/O part has not been fully defined, since it may depend on the success of a workable IR link. For now, I can use a laptop as the I/O device. The IR link is subject of this article.

IR TRANSMISSIONS

Most hand-held IR remotes modulate a 40-kHz carrier to transmit function codes to a receiving device (e.g., television or stereo). *[Author's Note: see Steve's articles on IR reprinted in volume 6 of the "Ciarcia's Circuit Cellar" books.]* The two elements of an IR communications link are the IR transmitter and IR receiver. Sharp makes a 40-kHz IR receiver/demodulator (Radio Shack #276-137) which can simplify much of the IR input circuitry. On the transmit side, a 40-kHz oscillator can be made with an LM555 timer.

What about transmissions? What method should be used? Since the most widely used communication hardware is the asynchronous serial port, it is my logical choice. The biggest stumbling block is how to handle bidirectional communication over the same transmission medium. This is less of a problem due to the nature of I/O networks. The master PC system, which does the actual sensing and control, is actually a slave to the I/O network. Listening intently through its IR ears in every room, the PC slave awaits a command. Meanwhile the portable IR master (the hand-held I/O device) is activated, asking for main menu information. The PC slave sends a menu screen back and awaits further instructions. Using menu screens you acquire data on the house's systems as well as control lights, heat, appliances, or let the cat out!

The drawbacks of the IR system include line-of-sight communications between transmitter/receiver pairs, and interference from other IR transmitting devices. The first problem can be minimized by proper placement of the IR transmitter/receiver on the wall to cover the widest area. The second is a matter of tossing out illegal data.

To prevent self-induced collision or reception, each transmitter/receiver is controlled by a flip-flop arrangement of one-shots. Refer to Figure 1 for the RS-232-to-IR communication adapter, keeping in mind that it takes two of these guys for a minimum link. Both one-shots are adjusted for a minimum time of one data word length. The *Q outputs of each disable the other one-shot from firing, thus preventing transmission during reception, and reception during transmission. If *RTS/*CTS is necessary, *CTS is held inactive (high) during reception preventing the UART from even sending out a character.

The final form of the hand-held I/O device will not need RS-232 conversion. But for now, the converters make it easy to test drive the link using any portable. Who

knows, a sufficiently small laptop might make the design of a special I/Odevice unnecessary. Until I've spent **a** good deal of time on the menu software, I won't know how small of a display I can use or how few keys I can get away with.

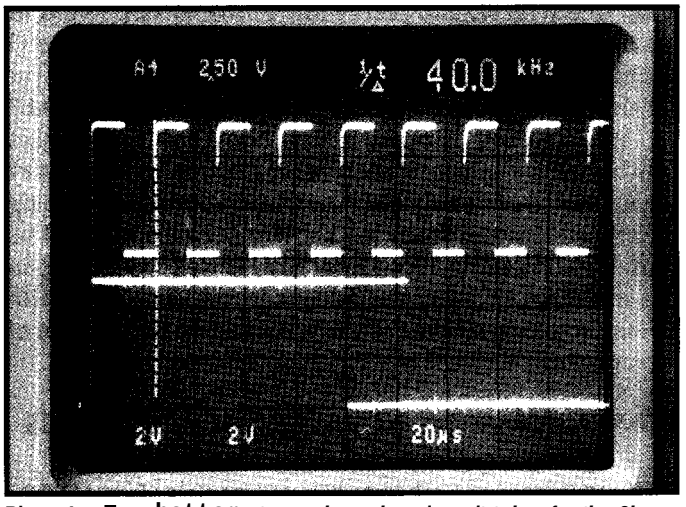


Photo 1 —*The* **bottom** ***trace shows how long it takes for the Sharp IR receiver to lock onto the incoming signal shown on top.***

DATA RATES

The top trace in Photo 1 is the 40-kHz carrier gated on and off with a 1-ms square wave. The scope is synced to the falling edge of the gate (start of a burst). The lower trace indicates the length of time the Sharp IR receiver/demodulator needs to lock on to the presence of the 40-kHz carrier. This is equal to about 6 cycles or 150 µs. To be safe, I figure the actual bit width of a "1" in a serial bit stream should be not less than three-fourths of the bit time for any particular baud rate. Take 2400 baud for example. Each bit time is430 µs. If the receiver takes 150µs to recognize a "1," then the actual received "1" bit time would be (415 – 150) or 265 µs. This time is about 64%, less than the three fourths I'm using as a minimum bit time, so it won't be quite adequate. The next slower rate, 1200 baud, will work. In this case, 860 – 150 = 710 µs or about 82% of the bit time.

To simplify the software, I make the assumption that devices will not speak at the same time. In fact, I tie the

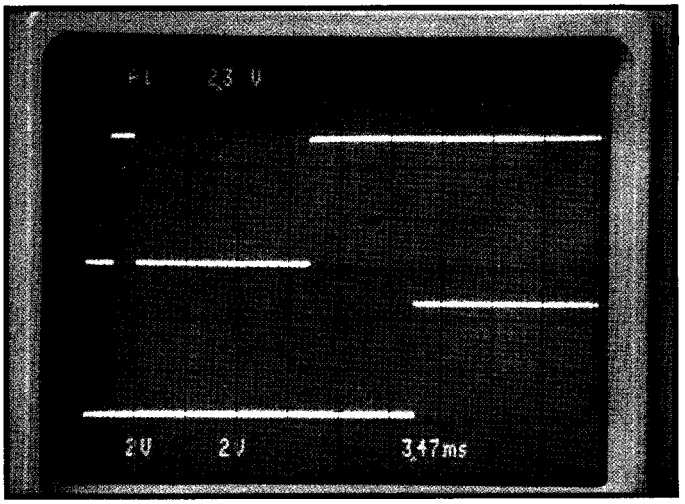


Photo 2—*The* ***one-shot output (bottom trace) shows how the timeout is*** keyed to ***the last 'one' bit sent on the top trace***

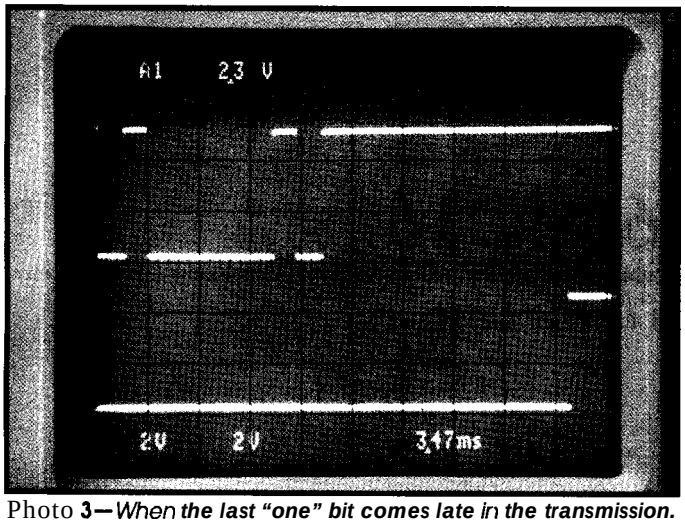


Photo 3—*When the last "one" bit comes late in the transmission. the timeout extends well beyond the end of the character.*

transmit and receive hardware such that one is held disabled while the other is active. One-shots are used to maintain control independent of the bit patterns. To do this, the one-shots are adjusted for a minimum timeout of one character time. They are triggered (and retriggered) on each falling "1" bit edge. The "null" character has only a start bit (the remaining eight bits are all "0"s.) It will timeout immediately after the character time, while other characters will retrigger the one-shot, at each "0" to "1" transition, extending the time out longer than one character time. Photos 2 and 3 show how the one-shot timeouts are affected by different characters. With this method, the confusion of transmitted characters echoing back to the receiver is avoided. However, a response which begins in less than one character time could be missed or garbled if the receiver's one-shot timeout has not occurred, releasing the disable to the transmitter.

CEBUS COMPATIBILITY

IR communication will be part of the CEBus standard. Unfortunately, it will make boat anchors out of all present 40-kHz IR remote devices. The new standard will use 100 kHz. (Don't miss Ken Davidson's update on the CEBus standard in this issue for more information.) When this technology hits, and all the right pieces are available, I'm sure my home will evolve right along with it. For now, I'll be satisfied with the present technology. Sure, there will be better and faster ways of doing things. That's part of the excitement. **The** excitement many overlook is that of making good use out of what we have available today.

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on the Circuit Cellar INK engineering staff. His background includes product design and manufacturing.

IRS

425 Very Useful
426 Moderately Useful
427 Not Useful

SILICON UPDATE

Tom Cantrell

The MC68HC16

Stretching 8 Bits To The Limit

I started in Silicon Valley in the late '70s as a microprocessor marketer at Intel. At the time, the micro wars, versus chief competitor Motorola, were just heating up with simultaneous battles for eight-bit sockets (8080/85 vs. 6800/02/09) and 16-bit share of mind (8086/88 vs. 68k).

Having learned to pitch everything from the 8080A to the '286, I was trained that complexity, for want of a better word, is good. Isn't the need for an XTHL instruction self-apparent? (Actually, it was the last 8080 instruction defined before the hardware designer exclaimed "no more.") Of course, 64K segments make sense. PCs really need four-level protection-but don't worry, the MMU descriptor scheme isn't that complicated...

Since then, thanks to techno-retro-backlash and/orfewerbraincells,I'vecome to appreciate the merits of simplicity. Today though, it seems like it's time to pull out the old habits because ever-increasing LSI density seems destined to lead to more complexity-after all, the only simple way to use all those transistors is in memory chips. In the micro world, even the lowest end controller needs a data book an inch thick. High-end 32-bit micros are as powerful as yesterday's mainframes but, like those dinosaurs, practically need an MIS department to get a "Hello World" program working.

Anyway, since transistors keep getting cheaper, let's put a positive spin on the inevitable. Perhaps, as some argue for the tax code, complexity isn't so bad. You just have to put in extra effort to see the forest through the trees.

With that in mind, let's take a look at the state-of-the-art MC68HC16, a chip that carries its simple 8-bit roots to new, indeed arguably terminal, limits. Don't worry, I'll try to keep it simple...

HUMBLE BEGINNINGS

Many will remember the venerable MC6800 introduced in the mid-'70s. In particular, the programmer's model is incredibly simple, consisting of two 8-bit accumulators, 16-bit index register, stack pointer and PC, and 6 bits of condition code. The only way it could be simpler is to have one accumulator instead of two (which is just the tack taken by even a simpler micro, the 6502). I actually knew a guy who programmed 6800s in hex—after all, the CPU offers only 72 different instructions (fewer than today's somewhat misleadingly named Reduced Instruction Set Computers). Makes you feel kind of nostalgic for those good old days.

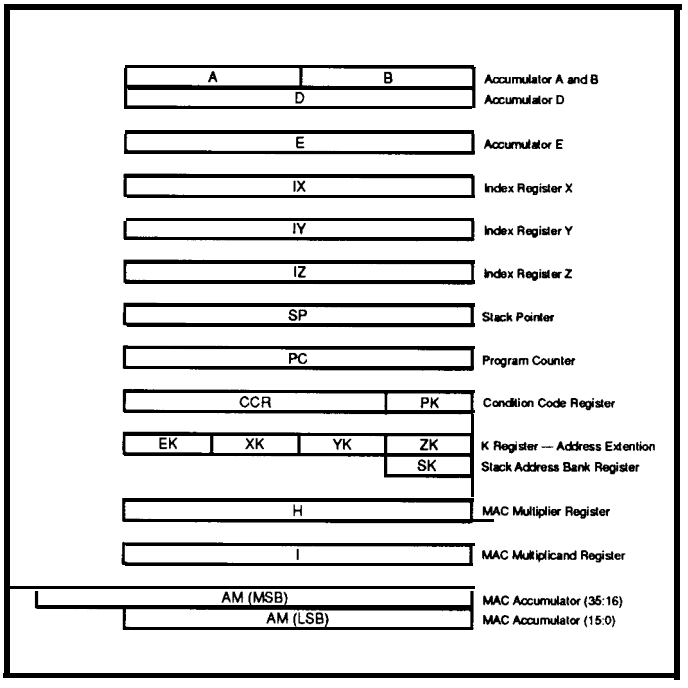


Figure 1 --The *MC68HC16's* register set resembles *that of the MC6800, but is greatly* enhanced.

Well, snap out of it! Times have changed as made clear by Figures 1 and 2. The MC68HC16 is a marvel of the '90s with more pins, registers, instructions, and I/O.

If you look closely, the MC6800 roots can still be discerned. Actually, some of the evolution from the MC6800 was seen in earlier parts. The MC6809 added a second index register (IY) and both that chip and the single-chip MC6801 adopted the scheme of using accumulators A and B together as a 16-bit accumulator D. Along with these, the MC68HC16 adds another 16-bit accumulator (E) and index register (IZ).

Beyond these simple additions, I've got a lot of explaining to do. As you can see, the HC16 is organized as a series of modules (in fact, they call it a "modular microcontroller") communicating via an IMB (InterModule Bus). Just keep in mind that the entire 6800 CPU can't match the functionality of even a small portion of the HC16 block

labeled "CPU16," not to mention all the HC16's I/O modules.

CPU16

The CPU16—despite its 8-bit heritage—is a completely new design built around a pipelined 16-bit ALU. Nevertheless, the new CPU is upward assembly language compatible with earlier 8-bit parts up to and including the popular MC68HC11. However, interrupt-driven routines will need to be rewritten since the interrupt masking **and** context switching portions of the code are inherently affected by the HC16 architectural upgrades.

A key difference from 68*xx* predecessors is the provision for greater than 64K of address space. Four-bit extensions (the K fields) are provided for the PC, SP, index registers (IX, IY, IZ), and E accumulator extending the HC16 reach to 1 MB (actually 2 MB-1 MB of code and 1 MB of data). Like the 8086, HD64180, and other stretched 64K machines, managing the large address space requires a degree of softwaremachinations. The HC16 does have a "pseudolinear" feature in which a calculated 20-bit effective address can cross 64K boundaries during instruction execution, though the K field is not automatically updated.

Perhaps the most notable feature is the addition of a "Multiply & Accumulate" (MAC) instruction which multiplies the 16-bit H and I registers and adds (accumulates1 the result to the 35-bit AM register. This primitive function is at the core of many DSP (Digital Signal Processing) algorithms. Also, index registers IX and IY are updated by adding a specified offset in preparation for loading H and I and executing the next MAC. This nonsequential accessing is called modulo-addressing and is helpful to speed the execution of filtering algorithms. The bottom line is that the simple MAC primitives allow the HC16 to serve some applications that might otherwise require both a CPU and separate DSP. So far, IBM (RS/6000) and now Motorola have both implemented MAC to great advantage; look for MIPS-starved competitors to follow suit.

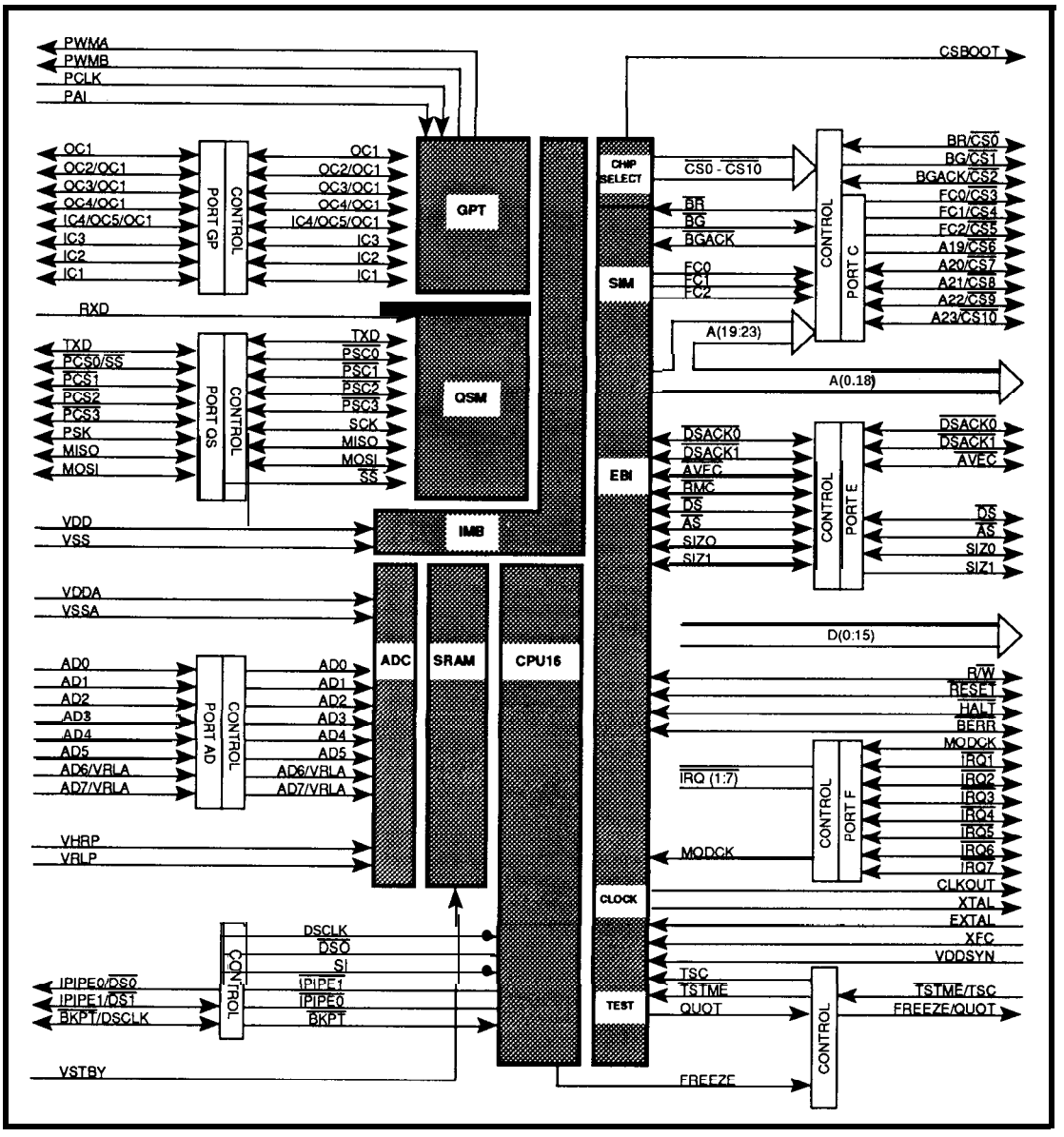


Figure 2—*The MC68HC* **16 is a** *marvel of the '90s,* **though** *its* **MC6800 roots are still obvious.**

Even the usually mundane clock generator merits a close look in the HC16 (Figure 3). Here, Motorola has a real innovation in the use of a phase-locked loop to derive the CPU clock (16.78 MHz) from a 32.768-kHz watch crystal. Most notably, thanks to programmable clock dividers and static design, the HC16 clock rate can be changed at will—an important feature to minimize power consumption.

The HC16 follows the trend of building in hardware test and debugging/emulator-assist logic. The former includes a dedicated scan-type serial port to allow test gear to select each module, shift in a stimulus, and read out the module's response. The latter include a hardware breakpoint/freeze scheme, outputs which indicate the CPU pipeline status, and a "show cycle" option which allows internal transfers to be monitored externally.

Along these same lines, the HC16 includes a LP STOP (low-power stop) command too.

Managing all this new stuff calls for a raft of new instructions (256 vs. 72) and addressing modes (18 vs. 6) compared to the original 6800. Thus, programming in assembler takes more thought than before-maybe it's finally time to switch to C.

As of today, the HC16 is not a single-chip solution like the earlier '01 and HC11. No on-chip program storage (ROM/EPROM) is provided, though 1K of static RAM (SRAM) is included. The latter can significantly boost performance since access takes only two clocks and the SRAM is organized as 512 x 16, ideal for the stack and frequently accessed variables. Just make sure you align the stack and/or data on word boundaries since, though the HC16 supports "nonaligned" accesses (a must for upward compatibility), they are twice as slow. Additionally, code can be executed from the SRAM-a nice touch allowing downloaded routines or even a poor man's cache scheme. Finally, the SRAM array features a separate power supply input for battery back-up (VSTBY) independent of the CPU and the HC16 automatically switches between the main and backup supplies as required.

SIMPLE BUS

Since the HC16, and thenatureof itslikelyapplication, calls for external memory, major resources are devoted to the connection of external chips via the System Integration Module (SIM). Besides the previously mentioned test and clock functions, the SIM offers protection, chip-select, and external bus interface logic.

Protection features include a watchdog timer, external bus timeout (signaled by the BERR* input), spurious interrupt (i.e., no acknowledge returned during interrupt arbitration), and so on. In addition, the HC16 architecture has hooks for User/Supervisor-type protection, though it is currently not implemented (the CPU only runs in Supervisor mode).

If a lot of this sounds familiar, it's because the external bus interface of the HC16 owes a lot more to the 68k CPU family than earlier 8-bit chips. 68k-like features include the interrupt scheme (eight inputs with acknowledge and "autovector" modes), external bus arbitration (BR, BG, BGACK), function codes (User/Supervisor, Code/Data), and asynchronous dynamically sized transfers (SIZ0/1, DSACK0/1*). The latter allows a mix of 8- and l&bit devices to be connected.

What is new is a chip-select controller featuring a total of up to 12 chip selects, which seems like **more** than enough for most applications. Most of the pins can be assigned alternate functions as shown in Figure 4 (note that though A20–A23 are routed on the IMB, currently they just follow A19). Each chip select area's base address and size is programmable, as well as the number of wait states (when using the available synchronous bus [non-DSACKI mode). CSBOOT* is one pin that doesn't have an alternate function; it must beconnected to the boot ROM(s) to be selected

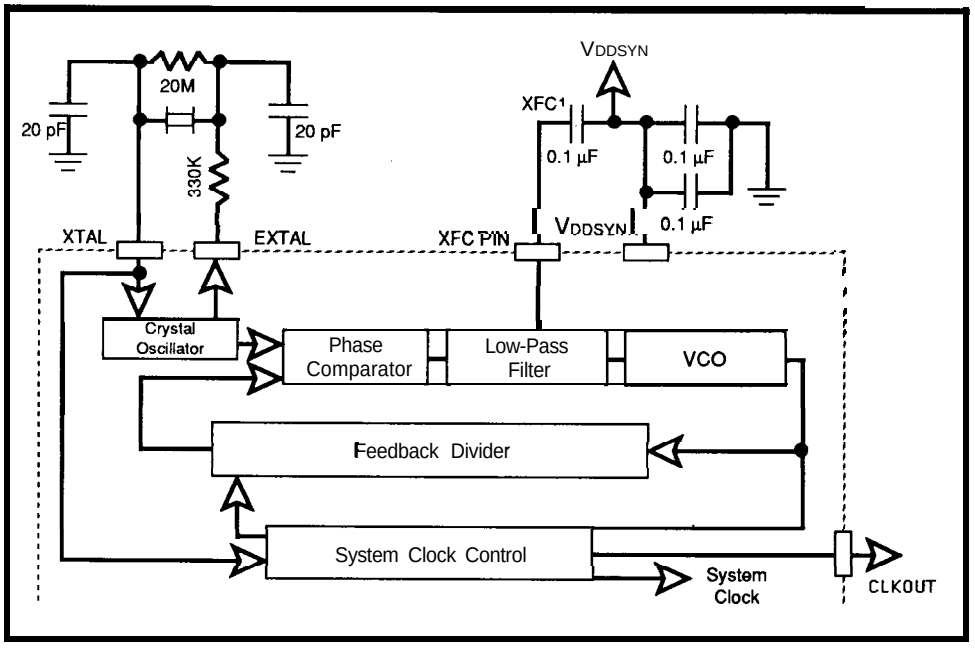


Figure 3—*The MC68HC16's phase-locked loop clock speed con be changed at will.*

at reset. I say "ROM(s)" because, thanks to the dynamic bus size scheme, the HC16 could limp by with a single 8-bit ROM. Needless to say, performance concerns would suggest that a 16-bit memory setup is called for, which makes the minimum HC16 system three chips instead of two (at least until 16-bit EPROMs are plentiful and cheap). The extra memory chip is simply the computing equivalent of "no free lunch."

The only thing that seems to be missing is DRAM refresh support. I imagine Motorola, perhaps rightly, assumes most HC16 designs won't use DRAM (especially as SRAMs get denser and cheaper). If DRAM is called for (e.g., a 512K setup might be well served with four 256K x 4 DRAMs) you'll have to use external DRAM refresh logic ($) or a bandwidth consuming interrupt and software refresh scheme.

BACK TO REALITY

We're about halfway done, which is pretty **good** considering the HC16 'Technical Summary" is over 100 pages long. What remainsare thetraditional I/Ofunctionstimer/counter, serial I/O, and A/D-though in the HC16, even these old standards are given quite a facelift.

Let's start with the counter/timer module known as the General Purpose Timer (GPT-Figure 5), since it is largely unchanged from the module used on the HC11. Note that the SIM includes some basic timing capabilities (watchdog and periodic interrupt) in addition to the GPT.

The major functionality of the GPT consists of eight channels with 16-bit resolution in a compare/capture unit composed of three input capture channels, four output compare channels, and one channel that can be programmed as either. In addition, the HC16 (and the HC11) has a conventional 8-bit counter known as the Pulse Accumulator.

The block diagram may look complicated, but the concepts of input capture and output compare are pretty easy to understand. Everything starts with a clock source which is software selectable as either an external pin (PCLK) or a division of the system clock.

Pin	Chip Select	Discrete Outputs
CSBOOT	$\overline{\text{CSBOOT}}$	—
$\overline{\text{BR}}$	$\overline{\text{CS0}}$	—
$\overline{\text{BG}}$	$\overline{\text{CS1}}$	—
$\overline{\text{BGBACK}}$	$\overline{\text{CS2}}$	—
FC0	$\overline{\text{CS3}}$	DO0
FC1	$\overline{\text{CS4}}$	DO1
FC2	$\overline{\text{CS5}}$	DO2
A19	$\overline{\text{CS6}}$	DO3
A20	$\overline{\text{CS7}}$	DO4
A21	$\overline{\text{CS8}}$	DO5
A22	$\overline{\text{CS9}}$	DO6
A23	$\overline{\text{CS10}}$	ECLK

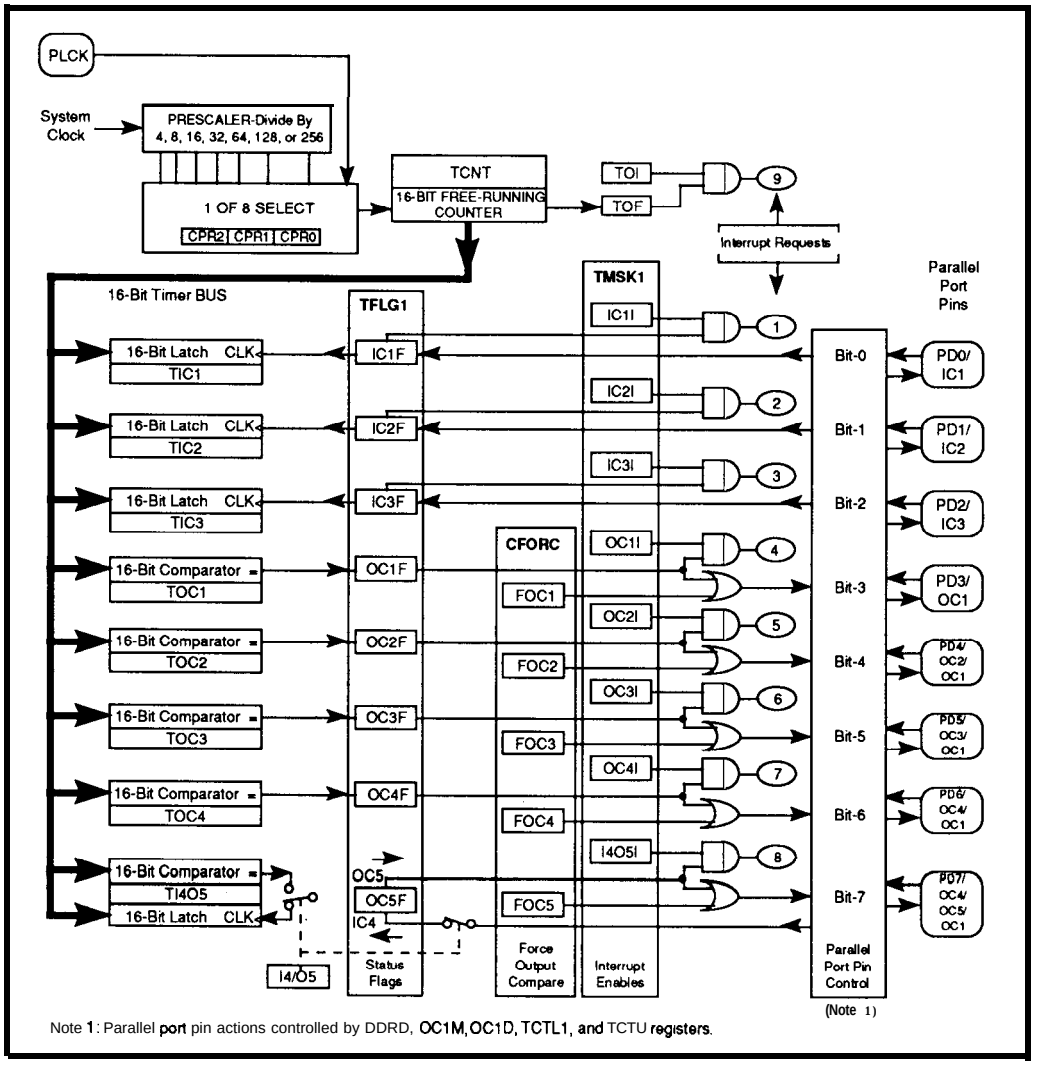
Figure 4— The *chip select controller provides for up to* **12** *chip selects, as well as a* **number** *of alternate functions.*

Input capture is used to measure the time between transitions on an input. Conceptually, it is quite similar to the technique you would use with a regular timer. In that scheme, the input to be measured connects to an interrupt line and the interrupt handler reads the timer. The problem that often emerges is the basic tradeoff between interrupt priority and latency. If the priority is too low, accuracy is lost since the delay between the interrupt and timer may become long and/or nondeterministic. On the other hand, setting the priority too high will likely step on some other real-time-critical task. Obviously, multichannel requirements amplify the problem to the point something gives.

What the input capture hardware actually does is trivial. It automatically reads the timer value into a register when the external signal transitions. But this small change makes a big difference to the CPU, which can access the captured timer value at leisure.

Output compare is kind of the reverse. Rather than measuring an input, it generates a timed output. Again, a timer/interrupt scheme could be used to wait the appropriate time and then perform an output, but the problems of priority and latency remain. For the HC16, each output compare channel has a register which is compared with the timer every cycle. Then, when there is a match, an automatic output-"1," "0," toggle—is performed. Thus, the CPU can schedule an output event, and go do something else.

What's new on the HC16 is a two-channel pulse-width modulation unit. The PWM can generate two separate waveforms of programmable frequency and duty cycle with no CPU overhead.



Note 1: Parallel port pin actions controlled by DDRD, OC1M, OC1D, TCTL1, and TCTU registers.

SMART A/D

A major advantage of the HC11 over a lot of popular 8-bit chips is the integration of an 8-channel multiplexed successive approximation A/D converter. Like the HC11 version, the new model features "scan" modes that automatically perform multichannel reads rather than requiring the CPU to initiate and monitor each conversion.

Figure 5— The *General-Purpose Timer (GPT) is basically unchanged from that in the MC68HC11,* with *eight 16-bit channels. four output compare channels, undone channel that can work as either.*

On the HC16, the ADC offers 10 bits of resolution (compared to the HC11's 8 bits) a nice fit on the new CPUs 16-bit architecture. Indeed, the ADC offers three complete sets of result registers storing unsigned left- and right-justified versions of the data and even a signed version. The latter is interesting in that the ADC itself is unipolar (only positive voltage). The trick is that the ADC logic automatically sets "0" to the midpoint of the low and high reference voltages-this can cut software that might be needed to scale the input in the same way. Speaking of reference voltages, the HC16 goes beyond the HC11 by offering two separate programmable reference sources. This can ease the design of external sensor signal conditioning circuits.

SUZI-Q

No, they didn't call it the "Serial Universal NRZ Interface-Queued." Nevertheless, the Queued Serial Module (QSM), though dryly named, offers its own degree of innovation.

The QSM actually includes two serial ports. The first is the ubiquitous UART (Universal AsynchronousReceiver/Transmitter) for serial communications. The UART offers all the usual features-programmable baud rate, data format, parity, etc.-including the modem wake-up feature. These allow a party-line arrangement of micros, typically using RS-485, to share a common set of wires. To prevent each node from being saturated monitoring all the bus traffic, the UART can be set to wake up and check the data only under certain conditions. One is when a ninth

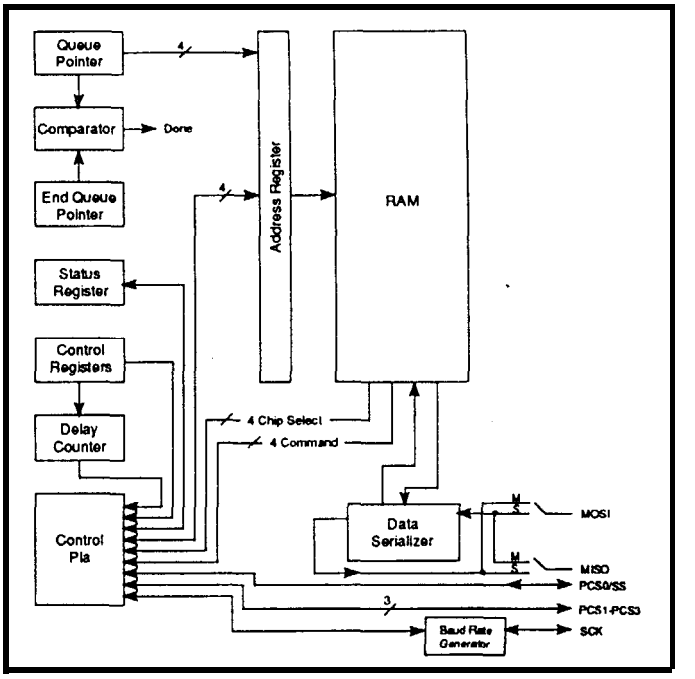


Figure 6—*The Queued Serial Peripheral Interface (QSPI) is optimized to communicate with other QSPI-equipped devices, much like the Philips PC bus.*

data bit is set. That bit can be used to signify that the other 8 bits contain an "address" which should be checked by all. Another approach offered detects an idle line condition. This could be useful in implementing a collision avoidance-type network protocol.

The unique aspect of the QSM is the second serial port: the Queued Serial Peripheral Interface (QSPI-Figure 6). The HC11 also has an SPI port, but without the Q feature. SPI is a specialized serial protocol optimized for connecting various similarly WI-equipped peripheral chips using a minimum of traces and glue logic. The concept (LAN-in-a-box?) is similar to the I²C (InterIntegrated Circuit) bus offered by Philips/Signetics and discussed in my previous article on their upgraded 8052-compatible family of controllers.

The Q feature, much in the manner of the A/D converter scan logic, offloads the CPU by buffering sequences of commands, transmit data, and receive data. The 80-byte QSPI RAM is organized as 16 words each of transmit/receive data along with 16 bytes of commands. With this feature, the QSPI can autonomously handle multiperipheral transfer sequences that would normally call for lots of CPU hand holding.

THE END OF THE LINE

Arguably, the HC16 is about as far as an 8-bit architecture can be taken. Indeed, it's not even 8 bits anymore. Motorola concedes that the HC16 is the final step a customer can take before biting the big bullet and moving to their 32-bit controller offerings: the 683*xx* line. In particular, the latter offer the final solution to the 64K problem. Presumably the "4GB problem" won't be a show stopper anytime soon for embedded applications (though the workstation types are starting to clamor for a 64-bit address space!).

However, the move to 32 bits means even more chips, power, and money and shouldn't be taken lightly. It's bad form to brute force a design with processor overkill. Clever designers will be able to squeeze excellent price/performance out of "high-end low-end" micros like the HC16.

Contact

Motorola Semiconductor Products
5005 E. McDowell Rd.
P.O. Box 20912
Phoenix, AZ 85008
(602) 244-6900

Tom Cantrell holds a B.S. in economics and an M.B.A. From UCLA. He owns and operates Microfuture, Inc., and has been in Silicon Valley for ten years working on chip, board, and system design and marketing.

IRS

428 Very Useful
429 Moderately Useful
430 Not Useful

PRACTICAL ALGORITHMS

Charles P. Boegli

Filtering Sampled Signals

Software DSP

A recent article [1] described a method of passing sampled signals through low- or high-pass filters by loading them into an array and manipulating them to obtain an array representing the filtered signal. This modus *operandi* is common in dealing with time series.

In practice, loading a time series into an array before operating on it is undesirable, because it may contain far too many values to be assigned to an array. For example, a five-minute interval of signal sampled 2000 times/sec contains 600,000 numbers, each of which, in six-digit single-precision floating-point format, occupies four bytes, for a total of 2.4 megabytes. This exceeds the memory capacity of many personal computers, and the array is vastly larger than BASIC can handle. Using integer representation requiring only two bytes per number eases the problem. Since the raw output of most A/D converter boards is already in integer form, this method is advantageous for real-time filtering; scaling to calibrated output **maybe postponed** until filtering is finished. The storage of a large array of digitized signal values often, however, quickly meets memory limitations.

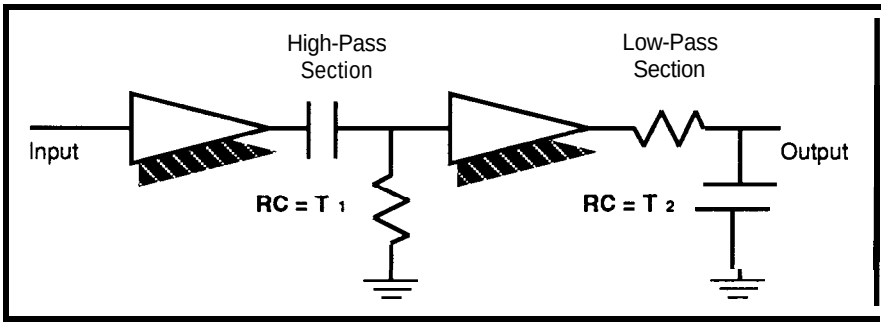


Figure 1 -The *high-pass/low-pass combination shown in the equations can be duplicated in hardware, using a circuit such as this one.*

High- or low-passing such signals is nevertheless possible without using an array any larger than two; further, the operation is so rapid that the record of the digitized signal may be read, and the filtered output written, simultaneously. The sole requirement is for disk storage sufficient to accommodate the entire input and output sample trains. This article shows how it's done.

BAND-PASSING SAMPLED SIGNALS

Since high- and low-pass filters are special cases of band-pass filters, we'll consider a band-pass filter first. This treatment assumes a filter made up of a high-pass filter having a time constant T_1 in series with a low-pass filter with a time constant of T_2, where both filters are electrically independent. Figure 1 shows an analog circuit for this type of filter. A filter of this type has an attenuation of 6 dB/octave above the frequency represented by T_2 and below that defined by T_1, the response being 1.0 between these points. Filters with attenuations that are integral multiples of 6 dB/octave can be simulated by passing the signal through the simple filter more than once.

The transfer function of the high-pass filter is

$$\frac{T_1 s}{T_1 s + 1}$$

where T_1 is the RC constant of the filter shown in Figure 1; similarly, for the low-pass filter it is

$$\frac{1}{T_2 s + 1}$$

so the transfer function for the band-pass filter is

$$F(s) = \frac{T_1 s}{T_1 s + 1} \times \frac{1}{T_2 s + 1} \qquad (1)$$

The response of this filter to a unit-step input is this expression multiplied by $1/s$, or

$$F(s) = \frac{T_1}{T_1 s + 1} \times \frac{1}{T_2 s + 1} \qquad (2)$$

Separation of this expression in the usual manner leads to

$$F(s) = \frac{T_1}{T_1 - T_2} \times \left(\frac{1}{s + \frac{1}{T_1}} - \frac{1}{s + \frac{1}{T_2}} \right) \qquad (3)$$

The corresponding time function is

$$f(t) = \frac{T_1}{T_1 - T_2} \times \left[\exp\left(\frac{-t}{T_1}\right) - \exp\left(\frac{-t}{T_2}\right) \right] \qquad (4a)$$

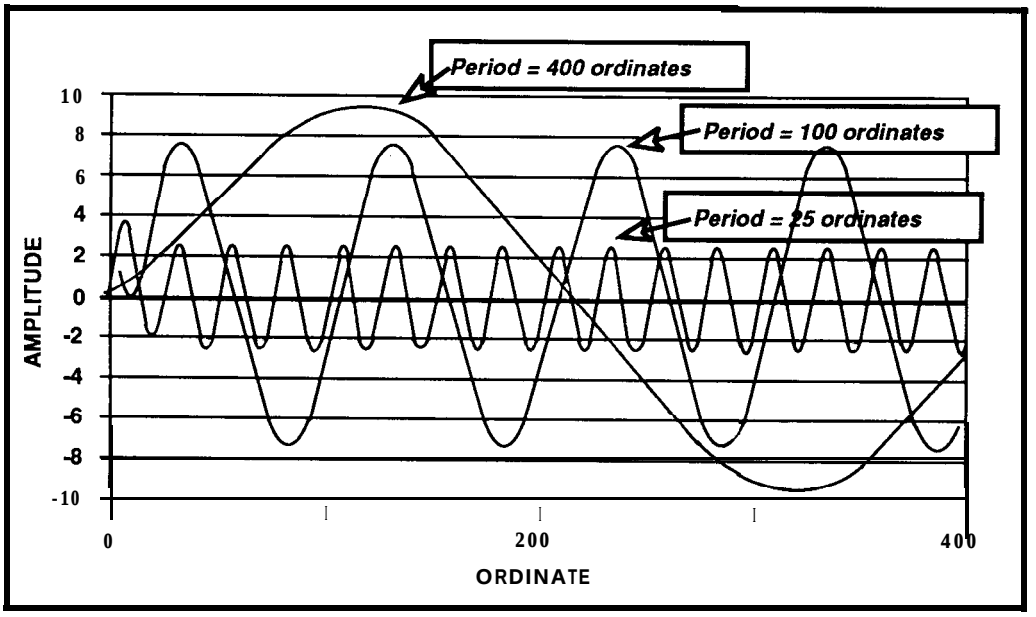


Figure 2—*Low-pass filter response to a sine-wave input. In this example, the corner is set to 100 ordinates/cycle.*

which is, precisely,

$$f(t) = \frac{T_1}{T_1 - T_2} \times \left\{\left[1 - \exp\left(\frac{-t}{T_1}\right)\right] - \left[1 - \exp\left(\frac{-t}{T_2}\right)\right]\right\} \qquad (4b)$$

where the expression exp(x) denotes e raised to the xth power.

Since the time responseof a simplelow-pass filter with a time constant T is $1 - \exp(t/T)$, equation 4b shows that the time response of a band-pass filter (to a unit-step input) is the difference in time responses of two low-pass filters having time constants T_1 and T_2. The problem is thus reduced to that of simulating the response of a low-pass filter to a sampled signal [2].

The response of a low-pass filter at time t to a step input of magnitude A being

$$f(t) = A \times \left[1 - \exp\left(\frac{-t}{T}\right)\right]$$

the response at time t + h is

$$f(t + h) = A \times \left\{1 - \exp\left[\frac{-(t + h)}{T}\right]\right\} \qquad (5a)$$

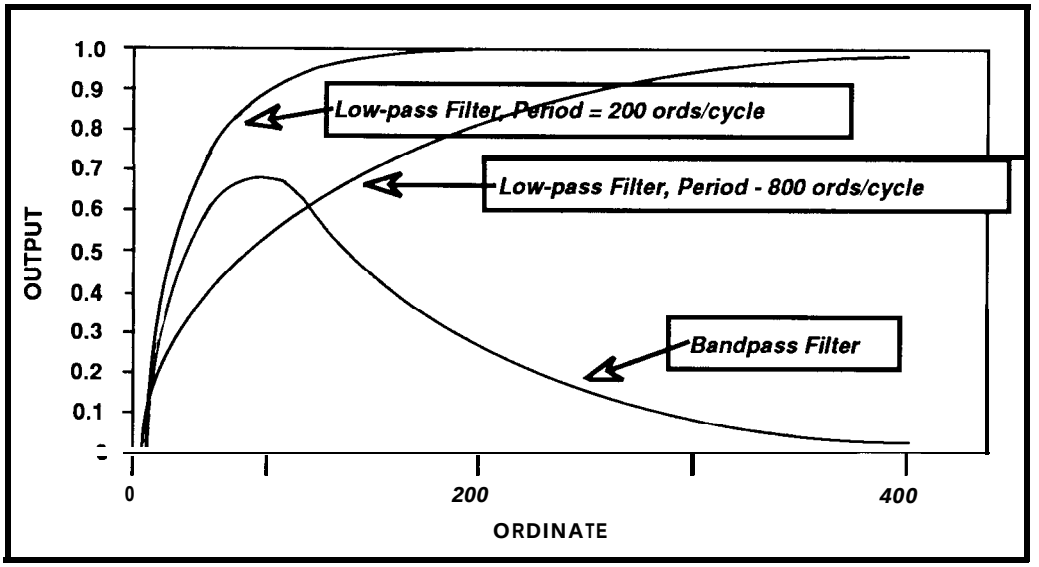
which is easily expanded to

$$f(t + h) = A \times \exp\left(\frac{-h}{T}\right) \times \left[1 - \exp\left(\frac{-t}{T}\right)\right] + \left[1 - \exp\left(\frac{-h}{T}\right)\right] \qquad (5b)$$

Digitized data can thus be low-pass filtered by taking exp(–h/T) times the preceding filtered ordinate and adding to it [1 – exp(–h/T)] times the next input value, where h = 1 if the time constants are in terms of ordinates per radian in these expressions. The load in computer memory at any time is only two numbers for each filter.

Figure 2 shows the effects of this operation on a sine wave of amplitude 10. The corner period being 100 ordinates/cycle, the 400-ordinate/cycle wave should be scarcely attenuated, while the 100-ordinate/cycle wave should be down 3 dB (7.07) and the 25-ordinate/cycle wave down 12 db (2.5). The filtered record usually shows a starting transient which can easily be seen in the figure, and may disturb operations like finding the maximum value in the entire record. It maybe eliminated by applying a Hanning window to the input data, which can easily be done while the work is in progress.

Figure J-Response *of a band-pass filter to a unit step input.*



Writing a program to filter signals is not difficult, but theoperationoftheequations is demonstrated much more easily with a spreadsheet program, all of which have built-ingraphingcapabilities. Figure 3 quickly demonstrates that two applications of equation 5b followed by subtracting the output of one low-passfilterfromtheother, does indeed simulate the response of a band-pass filter. *[Editor's Note:* Software for *this article is available from the Circuit Cellar BBS or on Software On Disk #21. See page 106 for downloading and ordering information.]*

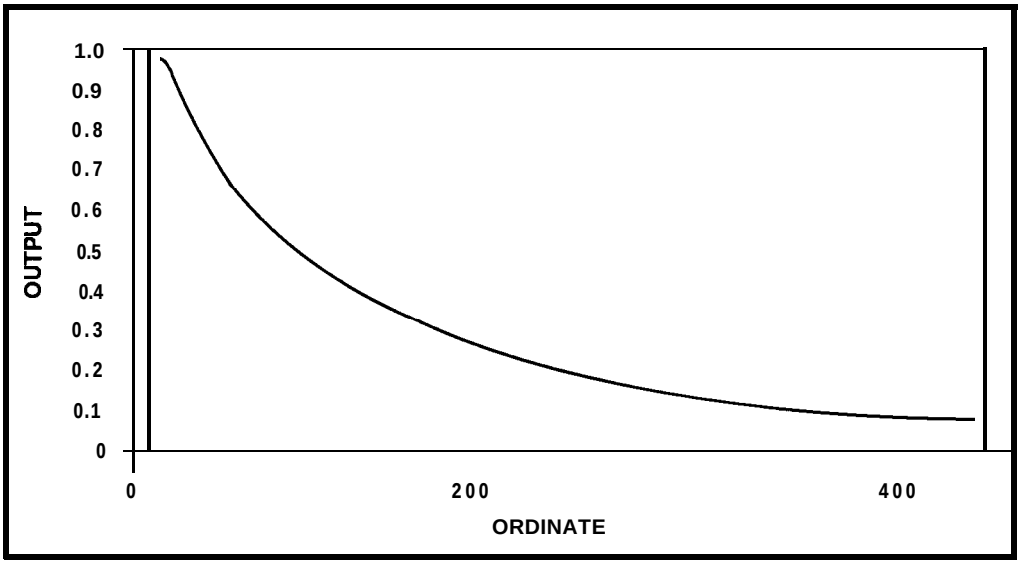


Figure 4-Response *of a* ***high-pass filter to a unit-step input where the time constant is set to 1000 ordinates/cycle.***

HIGH-PASS FILTERING

To realize a high-pass filter, the time constant T_2 of the low-pass part of the band-pass filter is moved to zero (that is, the comer frequency is moved to infinity). Expression (4b) degenerates to

$$f(t) = \exp\frac{-t}{(T_1)} \qquad (6a)$$

which is, of course, that for the timeresponseof a high-pass filter to a unit step input. This is identical to

$$f(t) = 1 - \left[1 - \exp\left(\frac{-t}{T_1}\right)\right] \qquad (6b)$$

showing that the response of a differentiator is related to that of an integrator of the same time constant. Figure 4, made with the same spreadsheet used before, shows the time response of a typical high-pass filter to a unit step

input. (Because the low-pass time constant could not be moved to zero without causing errors, the response does not quite attain a value of 1.0 at the beginning.)

LOW-PASS FILTERING

Similarly, the response of a low-pass filter can be determined by setting T, in expression 4b to infinity, which moves the comer frequency of the differentiator to zero. Then, expression 4b degenerates to

$$f(t) = 1 - \exp\left(\frac{t}{T_2}\right) \qquad (7)$$

which is the response of an integrator to a unit step function input.

SIMULATION BY DIFFERENTIATORS

As a matter of passing interest, equation 4a can also be viewed as a difference in time responses of two differentiators. This treatment is akin to using the so-called z-transform [3]. My personal experience with such band-pass filters is that they work as well as those made with integrators but have a tendency to be oscillatory with sharp transients. In this respect, simulation with integrators is preferable.

REFERENCES AND COMMENTS

(1) McConnell, Dean: *Digital Signal Processing*. Circuit Cellar INK, The Computer Applications Journal, February/March 1990. pp. 30–41.

(2) If the time constants T, and T_2 are identical, equation 4b does not apply. Equation 3 is an improper separation in this case because of the repeated factor in the denominator of equation 2. The expression corresponding to equation 4a is then

$$f(t) = \frac{t}{T} \times \exp\left(\frac{-t}{T}\right) \qquad (4c)$$

For too-small differences between T, and T_2 the accuracy of calculation using two low-pass filters is unacceptable. The program will not allow entry of such values.

(3) See, for instance, Truxai, John C., Control System Synthesis. New York, McGraw-Hill Book Company. Inc.. 1955

Charles P. Boegli is president of Randen Corporation in *Blanchester, Ohio. Randen is a small consulting/engineering company that specializes in interfacing computers to test and monitoring equipment, and in analog circuit design.*

IRS

431 Very Useful
432 Moderately Useful
433 Not Useful

CONNEC-TIME

Conducted by
Ken Davidson

Excerpts *from the Circuit* Cellar **BBS**

The Circuit Cellar BBS
300/1200/2400 bps
24 hours/7 days a week
(203) 871-1988
Four Incoming Lines
Vernon, Connecticut

In this installment **of** *ConnecTime, we're going to cover one of the newer hard disk drive interfaces that's recently been gaining popularity, a method* **of** *write-protecting hard disk drives, and a little prototyping trick. We'll start off talking about debouncing switch inputs (something most embedded applications designers encounter sooner or later).*

Msg#:37170
From: WALTER CRUDUP To: ALL USERS

Does anyone have short piece of code that would be good to use for switch debouncing of a input to an 8051 chip? The input will come from an 8255 chip port. How long is the noise on a typical switch closure or opening?

Msg#:37180
From: TOM MOORE To: WALTER CRUDUP

I suppose the amount of delay depends somewhat on the type of switch used. Well, I'll stick my neck out and say wait as long as you can afford to. :-)

Seriously, 1/20 of a second was the value I used once. It depends also on how fast and what type of input you have and how fast you need to react to it. Maybe some sort of hysteresis would help, like a Schmitt trigger (you might want to use CMOS for added help), but that's only if you can add hardware and NEED it.

Msg#:37183
From: AL DORMAN To: TOM MOORE

For hardware debounce try the MC14490; it works pretty well. For a software debounce, you test the input and ask, "Is it on? <delay> Is it still on? <delay> Has it been on for long enough?" Then either call the input true or wait until it turns off. You may want to test if it has been on too long, such as the "coin in" on a slot machine (coin jam). The amount of delays for the on test will be determined by how bouncy the switch is and how long till you consider it to be true. Also, during the delay, keep watching to see if the switch goes off. At this point you would bail out and consider the entire test as false.

Msg#:37217
From: MATTHEW TAYLOR To: TOM MOORE

Interesting subject. Just got a keypad hooked up to our furnace at work and it needed debouncing. Fifty milliseconds made the keypad seem sluggish, so I tried 10 ms and it has a great feel to it. BTW, I am using a good keypad, which does make a difference.

Msg#:37221
From: WALTER CRUDUP To: TOM MOORE

Thanks Tom and all who replied. I did not want to waste a lot of time checking the switches, so I decided to use Timer 0 to generate a interrupt every 20 to 50 ms. I check for a closure and then on the next interrupt, I check to see if the state was the same as the last time. If so then switch was made. I think this might work, and I can keep doing other things instead of sitting in a loop.

But, now I have a problem with the timer. I'm using mode 1 (16-bit) and am setting THO and TLO with values. I do reloads of the values to THO and TLO upon entering the interrupt routine, but the interval never changes, no matter what values I load. Based on the time of each interrupt (I turn on a port and check with a scope), the timer always uses FFFF (65535). Suggestions anyone?

Msg#:37252
From: ED NISLEY To: WALTER CRUDUP

I bet you forgot the "#" in front of the values you're jamming into the timer registers. That turns them into direct addresses, which will probably have zeros or some other constant data.

Gotta tell you I'm talking from experience on this one; won't tell you how long it took to figure it out.. .

Msg#:37228
From: AL DORMAN To: WALTER CRUDUP

Do you MOV IE,#OOH when entering the subroutine? Otherwise, your timer interrupt is still running and you may never get out of it. For example, TLO and THO may be timing out again while you are doing stuff in the interrupt routine and looping back to the

interrupt vector. The last thing you do on getting out of the timer interrupt is MOV IE,#82H. You must also PUSH and POP the ACC and some other stuff for proper operation (I don't have a listing handy).

Msg#:37253
From: ED NISLEY To: AL DORMAN

Better you should PUSH IE, CLR EA, POP IE in the subroutine. Never assume that you can arbitrarily blow away the contents of a whole register and reload it with a constant. The next version of the code will use another interrupt that will fail mysteriously!

We're all familiar with the workhorse of hard disk interfaces, MFM. Larger and faster drives are finding MFM limiting, though, so numerous new interfaces have starting surfacing, including SCSI, ESDI, and IDE. We recently had a caller asking about the specifics of IDE, and he received some interesting responses.

Msg#:36424
From: ALAN SANDERS To: ALL USERS

What are the details of the new IDE drive interfaces? I understand it's basically a direct connect to the XT/AT bus. The controller is embedded in the drive. I do know it uses a 40-pin IDC connector, so I suspect the only signals are:

8 or 16 data lines
3 or 4 least-significant address lines
direction control
enable (possibly with every other pin being ground)

Does anyone know the actual pin-out of this connector?

Msg#:36438
From: VICTOR PORGUEN To: ALAN SANDERS

I would also like to know. I have been told that IDE drives cannot be low-level formatted, being essentially hard sectored. This means one can't change interleave on them. Anyone know more about this?

Msg#:36476
From: NICHOLAS MARINO To: ALAN SANDERS

Conner Peripherals provides a schematic for an IDE interface card to their IDE drives in the drive's technical guide. I use large (200M+) IDE drives a lot since they are fast and, for some reason, mostly available in 3.5" size, even for the large capacity drives. I've had to upgrade my BIOS to AMI's latest (4/90) in order to get my Conner drives working properly. Without the upgraded BIOS, the drives hang for a few seconds every couple of thousand accesses. I don't know why.

I've just purchased a Seagate 211M drive. It seems like a good unit, but I don't like the fact that you have to install a dedicated

driver and XBIOS extension to get full capacity. I've also used large-sized ESDI drives. I wasn't impressed. Mine were big, power hungry, noisy, and needed expensive and hard-toobtain controllers. I'm sticking with IDE for now.

Msg#:36501
From: JAMES D STEWART To: VICTOR PORGUEN

You are correct: you cannot hard format the drive. Rest assured that it comes formatted with the fastest interleave that the drive can support. Probably it doesn't even have standard size or format sectors. Since the host can't get to the physical drive, it doesn't matter. Here is a quick description of the bus:

Reset, Data O-15, *IORD, *IOWR, *IRQ14, ALE, *CS1, *CS2, IO16, GND, *PDIAG, *SLV/ACT

When the drive is connected to the PC/AT bus with a trivial interface, the ROM BIOS in the AT will run the drive. If you need more information, call Conner at (408) 433-3340 and ask for a product manual. Their 44-meg IDE drive is the CP3044.

Msg#:36546
From: VICTOR PORGUEN To: JAMES D STEWART

Thanks, I will call Conner. The subject has interest for me from an unusual viewpoint: infection with computer viruses. Some of them try to play unusual tricks with various portions of the disk. What you say implies that it will be far more difficult for them to do so on the new IDE drives. It also means, however, that if they do manage to get to it, it will be also much more difficult to remove their effects.

Msg#:36903
From: MICHAEL KOEPKE To: ALAN SANDERS

I work with laptop designs that use the Conner IDE drives. The two signals *PDIAG and *SLV/ACT are defined as follows:

*PDIAG - Passed diagnostic. Output by the drive if it is a slave. Input to the master if it is a master. This low-true signal indicates that the slave has passed its internal diagnostic command.

*SLV/ACT-Signal from thedriveisusedeithertodriveanactive LED whenever the disk is being accessed or as an indication of a second drive present.

The Conner IDE drives let you run two drives off an IDE interface board. By changing several jumpers on the drives, you can run in a master/slave environment.

Also, if you are planning to use an IDE drive for an embedded system, you might consider the Prairie Tek drives. They support the same signals as the Conner but with one additional feature: You can use the drive in either XT (8-bit) or AT (16-bit) mode. A signal (AT/*XT) controls which mode the drive operates in. Most Conner drives can only operate in 16-bit (AT) mode. Model numbers for Prairie Tek are Prairie 120 (20 MB) and Prairie 240 (40 MB).

Speaking of hard drives, have you ever wanted to protect your hard disk from accidental or malicious changes by pulling it out of your system, putting a sticker over a little notch, and sticking it back in? If so, you might find the following conversation interesting.

Msg#:36933
From: TOM MOORE To: ALL USERS

I've been toying with the idea of adding a small circuit to the hard drive controller I now have (an RLL unit). It would go something like this: take write gate before it leaves for the hard drive and using a logic gate switch either let it pass through or disable it by forcing it high. The gate(s) would be wired to a toggle switch mounted on the case of the computer. Does this sound workable?

The only gotcha I know of might beduring power up/down if my add-on circuit would accidentally glitch the 'WG line causing, of course, an unwanted write. Maybe I could nab the "WG signal BEFORE it gets to any anti-power-glitch circuitry on the controller board? I realize not everyone might be able to use this type of write protect (e.g., SCSI drives), but what about those of us that have MFM or RLL systems? Just a passing thought.. .any ideas?

Msg#:37015
From: VICTOR PORGUEN To: TOM MOORE

The hardware write-protect you describe is perfectly feasible. I've beenemploying it for over a year now to work with computer viruses, as a means of ensuring total security against unwanted writes and yet leaving the disk fully readable.

As you say, all you need to do is wire a SPST switch across wire number 6 of the controller-to-disk ribbon cable. That's the Write Gate line. Some really interesting effects are seen: DOS has NO IDEA WHATSOEVER that it can no longer write to the disk. It thinks it has written and, if you keep a modest BUFFERS= statement in CONFIG.SYS, the name/size/date of the new file are entered there, as if the file had been written to disk. If you then reboot, the information is gone!

If, in addition to doing the above, you also use a largish (1.5MB or so) disk cache, you end up with a system on which you can work with fairly large files, create new ones, delete them, and so on, without ever writing a single byte to the disk.

I have never experienced any problems due to glitches on bootup or any other. The only caution I would have is the following: the ribbon cable consists of two kinds of wires: signal-carrying and ground. The even ones are signal, the odd ones are ground. They form 125/150-ohm transmission lines. If you extend wire number 6 to the switch by any appreciable distance, you should take pains to also extend the accompanying shield/ground wires, so as to keep the impedance more or less even and prevent rounding the shoulder of the square wave signals. I think their rise time is fairly short.

Finally, how is one supposed to build a prototype of a circuit when the chips being used are intended to be surface mounted? Sockets are sometimes available, but often only with solder tails. The following messages talkabout a neat trick for making a home-brew prototyping socket.

Msg#:36479
From: EDDIE WHITE To: ALL USERS

Does anyone know of a wire-wrap adapter for 52- and 68-pin PLCCs? JDR is selling a 68-pin PLCC prototype adapter, but nothing for 52 pins. By wire-wrap I mean something with standard 0.100" vector board spacing and has either wire-wrap legs or could be fitted into a wire-wrap socket by some fashion.

Msg#:36505
From: JAMES D STEWART To: EDDIE WHITE

Buy a PLCC through-hole solder-tail socket and plug it into a PGA wire-wrap socket.

Msg#:36548
From: PAUL SHUBEL To: JAMES D STEWART

Thanks for the great suggestion on using PLCCs. This really makes my day! One question though: The solder-tail socket I have seems to have thin leads. Will they fit snug enough in the PGA socket to be reliable?

Msg#:36549
From: JAMES D STEWART To: PAUL SHUBEL

I've used them several times and they fit tight. I've had no problems.

The Circuit Cellar BBS runs on a 10-MHz Micromint OEM-286 IBM PC/AT-compatible computer using the multiline version of The Bread Board System (TBBS 2.1M) and currently has four modems connected. We invite you to call and exchange ideas with other Circuit Cellar readers. It is available 24 hours a day and can be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, and either 300, 1200, or 2400 bps.

IRS

434 Very Useful
435 Moderately Useful
436 Not Useful

STEVE'S OWN INK

Steve Ciarcia

Reach Out...

Maybe I've given some of you the wrong impression. It's my own fault, of course: Over the last three years I've written about fixing a BMW with a 2 x 4, how not to automate a sunroom, and then, in the previous issue, why I don't carry a laptop computer. Some folks could read all of this and decide that I don't likecomputers or, worse yet, that I'm somehow not participating in the modern age. Let me assure you that nothing could be farther from the truth. The fact of the matter is that my life is full of intelligent machines, but I've learned to pick them so that they do necessary jobs. For me, one of the most necessary jobs is simply keeping in touch.

If you've been reading my articles for very long, you know that I've put a lot of time and effort into my house. My toys are there, the Circuit Cellar is there, and I like spending time there. In fact, it's usually pretty tough convincing me to go away. When someone does pry me away from the house, it makes me feel good if I can know what's going on there. Two years ago, we made ROVER. In some ways, ROVER is the ultimate way to stay in touch since I actually get to see what's happening at the Circuit Cellar. There are limitations to the system, but it's nice to be able to see who's driving up the road when I'm not there. As much as I like ROVER, though, there are times when it's just not practical.

For instance, if I'm scheduled for a day of meetings away from the office, carrying ROVER will put a definite crimp in my style-not to mention my arm! Since I haven't wired my car with 110 VAC (yet...), driving down the highway is no good for ROVER. For times such as these, the voice synthesizer I described in the special section lets me stay in touch. It's not as good as seeing the action, but at least I can take my house's word that everything is as it should be. There are computers and controllers attached at every point of the transaction, but they take a back seat to the communications involved. As communications technologyevolves, I plan to go even farther in my "keeping in touch" projects, but there is, in the middle of all this, a paradox.

The paradox doesn't really involve the technology, it involves me. I have a feeling that the same paradox will eventually hit a lot of people, even if it takes longer for them to get to the same point. The paradox is this:

The more communications evolve to let me stay in touch when I'm away from home, the more they reduce the need for me to leave home in the first place.

I've already said that I like being at home, and as the technology for bringing more services into my house increases in capability, the pressure to travel decreases. I like the idea of videoconferencing from the Circuit Cellar. It would be nice to see the range of information available through telephone and cable systems increased. With enough bandwidth in and out of the house, I could even go to the next step with ROVER, simply sending live video, rather than digitized frames, to wherever I happened to be. This last idea is where portable computers and I finally come together, too.

I still don't want to type letters using a pizza box that sits on my lap. What I do want is a fully functional multidata terminal that isn't a burden to carry. If there were a small (say, small notebook size) terminal that let me connect to telephone or cable, display the information that came to me, and respond, then I'd carry it everywhere. Admittedly, there aren't a lot of services taking advantage of those capabilities today, but I want them now and I'll need them before long.

Looked at in one way, the history of computers has been about communications. Yes, computers process data, but primarily so that information can be transmitted between groups of people. Yes, computers control, but much of that control has to do with transmitting commands to a device and results back to the user. My needs may be out of the ordinary, but they fall well within the basic march of computing. Now, I just need someone to make the products I need. Or maybe I can find that small surplus case that was lying around here.. .