CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

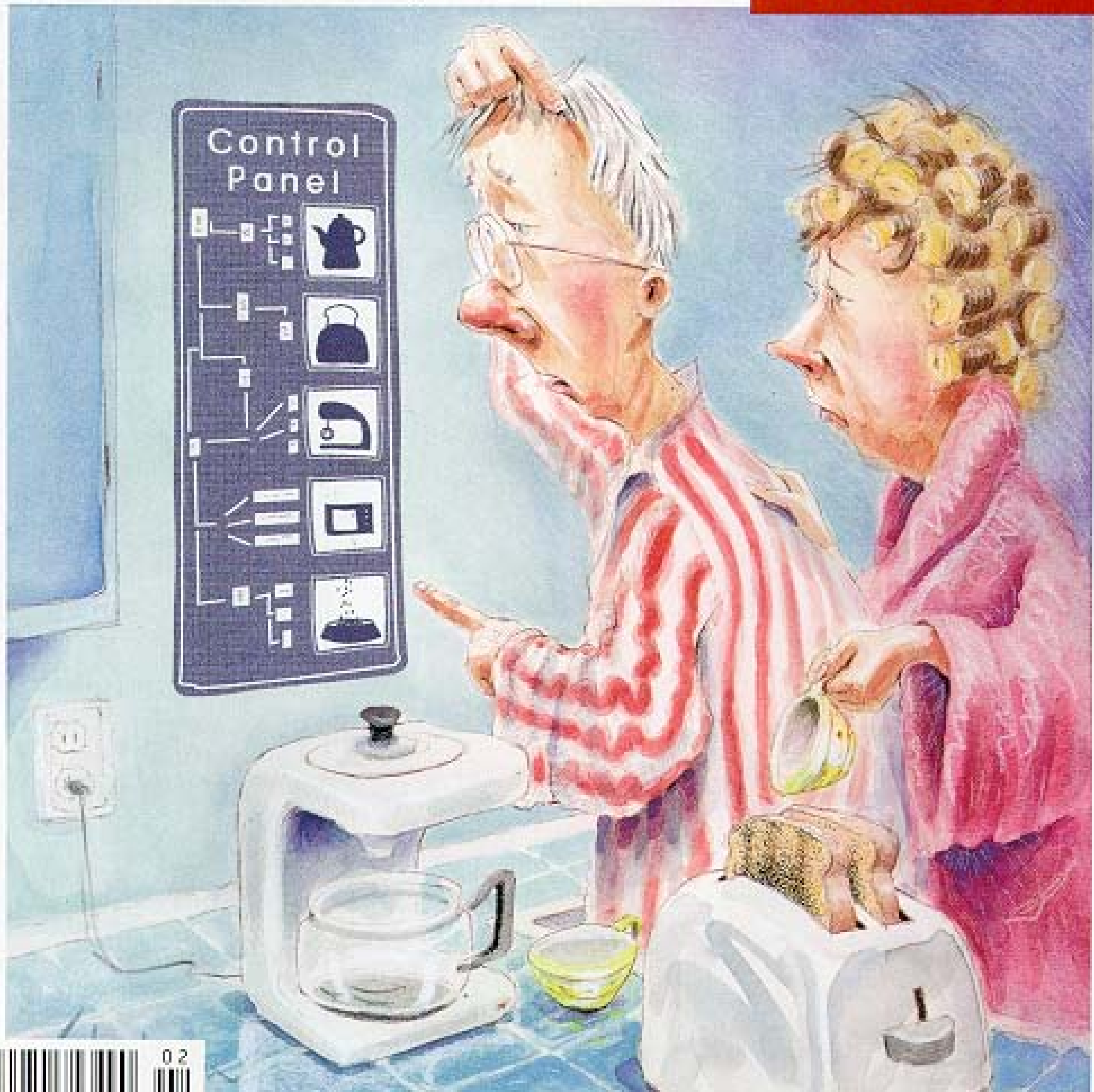
HOME & BUILDING AUTOMATION

SMART HOUSE Overview

Temperature Sensor Interfacing

CEBus Revisited

February 1993 — Issue #31



$3.95 U.S.
$4.95 Canada

EDITOR'S INK

Increasing the Bandwidth

Welcome to the first monthly issue of the Computer Applications Journal. We have two new monthly columns starting in this issue. In the first, called "Embedded Techniques," John Dybowski will be addressing hardware, software, and firmware issues related specifically to designing embedded controllers and data collection devices. John had taken over the "Practical Algorithms" column, but we found the title to be too restrictive for the material he wanted to cover. We look forward to John's insights from his many years of embedded design.

The second column, called "Patent Talk," is something completely new for us. Each month, Dr. Russ Reiss will be selecting patents covering inventions that he feels would benefit our readers. When you're working on a new product, it's often useful to know what's already been patented either so you can avoid the same design and, hence, avoid lawsuits, or so you can incorporate all or part of the design in yours and make the appropriate licensing arrangements. Russ discloses his intentions in more detail in this month's introduction.

Along with upgrading the volume of information you'll be getting from the printed page, we've also made several upgrades to the BBS to increase the amount of information you can get for your dime. For several years, we've had a USRobotics Courier HST 9600-bps modem on one of our four incoming phone lines, while the other three were stuck at 2400 bps. We have now added three V.32bis 14.4-kbps modems to the main lines that support everything from 300-bps Bell 103 right up to the latest standards (V.32, V.32bis, V.42, V.42bis, MNP5, etc.). What a treat it is to be able to connect at speeds identical to a direct-wire connection. It's now possible to download a 400K file in under five minutes.

To go along with the faster modems, we now have a QWK interface. Called QSO (QWK Support Option), you may select which messages you want to read (e.g., all new messages posted since your last call), download them in compressed form, and read them at your leisure using a special off-line reader. You may compose replies while off-line and upload them to the BBS, again in compressed form. We have several QWK-compatible off-line readers available on the BBS, and there are many others available in the BBS community at large.

Using the combination of the 14.4-kbps modems and QSO, you should now be able to call the BBS, download a week's worth of messages, and hang up in a matter of minutes, where it would take most people 45 minutes or more to read the same messages on-line at 2400 bps. After composing your replies, you should be able to call, upload them, and hang up in under two minutes. Using an automated script with your comm program, you can probably shorten these times even more.

We continue to strive to bring more to you, our reader. Let us know how we're doing.

Ken

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

ASSOCIATE EDITOR
Lisa Nadile

TECHNICAL EDITOR
Robert Rojas

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
John Dybowski & Russ Reiss

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

GRAPHIC ARTIST
Joseph Quinlan

STAFF RESEARCHERS:
Northeast
John Dybowski
Midwest
Jon Elson & Tim McDonough
West Coast
Frank Kuechmann

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Susan McGill

CIRCULATION COORDINATOR
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly 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 (12 issues) subscription rate U S A. and possessions $21.95, Canada/Mexico $31.95, all other countries $49 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 and subscription related questions to The Computer Applications Journal Subscriptions. P 0. Box 3050, Southeastern, PA 19398 or call (215) 630.1914. POSTMASTER. Please send address changes to The Computer Applications Journal, Circulation Dept., P.O. Box 3050, Southeastern, PA 19396.

Cover Illustration by Bob Schuchman

Circuit Cellar BBS—24 Hrs. 300112001240019600114 4k bps, 8 bits, no parity, 1 stop bit, (203) 871-1988; 2400/9600 bps Courier HST. (203) 871-0549

INSIDE ISSUE 31

READER'S INK

MICROVOLT DISAGREEMENT

As an analog design engineer, I look forward to receiving each issue of the *CAJ* to further my knowledge in digital and microprocessor design and applications. However, as a reader of the *CAJ*, I must take exception to the article "Microvolt Measurements," Dec. '92fan. '93, issue #30, by Russ Lindgren.

I have serious reservations that the design proposed by Mr. Lindgren would measure anything close to 18-20 bits. How did Mr. Lindgren test his design? Mr. Lindgren says he presents some design tricks to getting 20 bits of precision from the AD7703. I would consider his analog design and circuit construction techniques anything but unique.

The analog interface proposed by Mr. Lindgren simply does not address the pertinent details of a precision A/D system. If you wish to reliably measure 18-20 bits, you must follow certain fundamental rules. Please note my comments below:

1. The author suggests for accuracy you should solder or wire-wrap everything. Absolutely not! The OPA121 KP precision op-amp is an excellent op-amp, but you cannot place this part on a prototype board with a wire-wrap socket (as shown by the author) and expect anywhere near specified performance. This component must be placed on a printed circuit board with a ring guard around its inputs and a substantial ground plane should be used. And furthermore, the pod enclosure should be constructed of metal and not plastic.

2. The author shows two 4–6-inch single-conductor wires supplying the analog input signal to the ADC. Absolutely not! These wires act as antennae. Contrary to what the author says regarding special cabling (none required), this area is critical for low-noise applications. The input leads should be as short as possible and shielded.

3. The computer switching power supply is definitely not recommended for precision A/D conversion regardless of how convenient it may seem. Noise pulses on the digital ground will cause significant errors in the A/D conversion process if the analog and digital grounds are as one with the computer ground as shown by the author. You must take advantage of the separate analog and digital grounds as provided by the AD7703. Low-noise analog supplies or batteries are recommended for the analog portion of the circuit with decoupling capacitors on each supply line of the AD7703.

4. The six-pole Gaussian filter does not filter all frequencies above 10 Hz as stated by the author. In fact, the digital filter *does not* provide any rejection at integer multiples of the sampling frequency $n * f_{CLKIN}/256$ [where n = 1, 2, 3, .). A low-pass active filter preceding the analog input would remove this noise as well as help reduce broadband noise.

5. The author also states, "that in his experience with the AD7703 the only signal conditioning required is to limit the input frequencies to less than the sampling rate f_S of the filter." What about the Nyquist criteria? The input frequency should never exceed $f_S/2$.

6. The digital interface to the AD 7703 should be minimized, as recommended by the manufacturer, with the CMOS logic to reduce crosstalk between analog and digital portions of the circuit.

I hope that you will pass this information along to your readers.

Ronald S. Wolff
Sharpes, Fla.

Russ Lindgren replies:

I appreciate your suggestions resulting from your experience as an analog design engineer. These suggestions clearly address the analog techniques critical for repeatable, high-resolution measurements when using standard devices such as op-amps, comparators, and especially A/D converters. However, because the AD7703 is a unique device designed especially for converting the types of signals found in process control, a simple circuit, such as the one presented in my article, can yield excellent results for relative measurements.

The designers of the AD7703 took the basic sigma-delta converter design and, by adding extra filtering on-chip, developed an A/D converter that needs no additional analog components to accurately digitize low-frequency signals. Plus it's quite inexpensive. To provide additional noise rejection, I added an averaging queue in the PC software and created a practically all-digital design that needs no tweaking.

When converting an analog signal into a digital stream of values, wouldn't it be useful to move most of the noise high above the bandwidth of the signal, where it can be easily filtered out with a simple low-pass filter! That's exactly the approach taken by the AD7703 sigma-delta converter. The internal low-pass filter of the AD7703 sits between the input sample-and-hold amplifier and the one-bit comparator, providing rejection of out-of-band signals and prefiltering the signal to within the Nyquist limits. Since this sample-and-hold requires a fixed settling time, it can't accurately track signals above 16 kHz, which is also the maximum sampling rate of the ADC.

To further describe this operation, I'll quote from the AD7703 data sheet (page 7): "The AD7703 samples the *input signal at 16 kHz, which spreads the quantization noise from 0 to 8 kHz. Since the specified analog input bandwidth of the AD7703 is only 0 to 10 Hz, the noise energy in this bandwidth would be only 1/800 of the total quantization noise, assuming that the noise energy was spread evenly throughout the spectrum. It is reduced still further by analog filtering in the modulator loop, which shapes the quantization* noise spectrum to move most of the noise energy to *frequencies above 10 Hz. The SNR performance in the 0- to 10-Hz range is conditioned to the 20-bit level in this fashion." I hope this illustrates the simplicity and power of the sigma-delta technique.*

Finally, I do wish to mention that the design presented in the "Microvolt Measurements" article offers excellent noise rejection, but is not guaranteed to yield absolute measurement precision to 20 bits. For that type of accuracy it's best to rely on high-accuracy, calibrated DMMs. The design will, however, provide repeatable results when intelligently applied to real-world measurements.

X-10 Overseas

We have received many (many) calls and letters from our overseas readers asking about X-10 modules (and specifically the *TW523*) for *220-V* systems. We've finally tracked down some sources for these parts. We are told that virtually all modules available in the U.S. have been adapted for use with *220* V. In addition, plug options are available for numerous countries. Contact:

Europe	Pacific
Celtel Limited	X-10 Ltd. (HKG)
P.O. Box 135	Room 1103-4, Hilder Centre
Basingstoke, Hampshire	2 Sung Ping Street
RG25 2HZ	Hunghon, Kowloon
England	Hong Kong
Phone: 0256-64324	Phone: (852) 334-8848
Fax: 0256-8 18064	Fax: (852) 764-2437

We Want to Hear from You

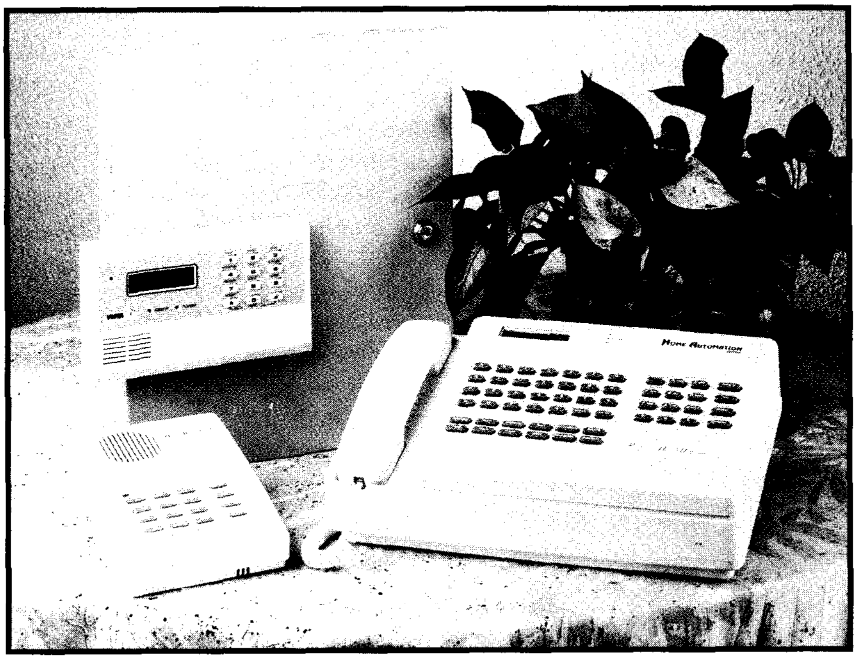
We encourage our readers to write letters of praise, condemnation, or suggestion to the editors of the Computer Applications Journal. Send them to:

The Computer Applications Journal
letters to the Editor
4 Park Street
Vernon, CT 06066

NEW PRODUCT NEWS

Edited by Harv Weiner

HOME AUTOMATION SYSTEM



Ademco Device Manufacturing Co. announces a home automation system that fully integrates with a professional security system. **The Home Automation System** links with Ademco Device Manufacturing's VISTA system to provide professional home security with telephone, lighting, appliance, and temperature control. The Personal *Home Director* and remote control devices known as *Room Directors* operate the system. A synthesized voice with 2,000 to 3,000 phrases guides the user through the system setup and operation.

Ademco's Home Automation System combines a speakerphone and an answering machine that can store a dialing directory of up to 100 names. The telephone functions also include message forwarding, call screening, toll-saver, and time-date stamping. It can also operate as a household intercom that can selectively call up to *16* stations.

The system controls lighting and appliances remotely and at preprogrammed times. The energy management feature improves on a single thermostat by averaging the temperature readings in various rooms.

Professional home security features protect the home from intrusion, burglary, and fire through hard-wire or supervised wireless technology. The Personal Home Director, Room Director, or both control security functions.

The system is installed by a professional security or alarm system dealer, with prices from $4,500 to $6,000.

Ademco Device Manufacturing Co., Home Automation Division
165 Eileen Way . Syosset, NY 11791 . (516) 921-6704

#500

REAL-TIME CLOCK CHIP FEATURES NONVOLATILE RAM

Dallas Semiconductor has introduced a new series of timekeepers with real-time clock/calendar and nonvolatile RAM functions accessible in a byte-wide format. The **DS164x** Timekeeping **RAM** series features memory densities in the range of 2-5 12 KB. Easily substitute the device packages, which adhere to JEDEC byte-wide pinouts, in ROM, EPROM, and EEPROM sockets providing R/W nonvolatility with the addition of a real-time clock.

The devices incorporate the real-time clock registers in the uppermost 8 bytes of the memory map. The RTC registers provide year, month, date, day, hours, minutes, and seconds data in 24-hour BCD format.

Devices in the family are from the DS1642 (2K x 8) to the DS 1647 (512K x 8). The DS1642 sells for $13 in production quantities.

Dallas Semiconductor
4401 S. **Beltwood** Pkwy.
Dallas, TX 75244-3292
(214) 450-0448

#501

NEW PRODUCT NEWS

IN-CIRCUIT EMULATOR FOR MOTOROLA 68300 SERIES PROCESSORS

Huntsville Microsystems announces the release of its new "LITE" series of **real-time emulators** for the Motorola 68300 family. The units include HMI's powerful SourceGate debugger.

SourceGate provides full emulator control integrated with source-level debugging and support for all major C and Ada compilers. SourceGate uses a high-speed serial port (115.2 kbps) to support communications, and a parallel port for code downloading.

The emulators are configured with 256 KB of RAM and support mapping of all chip select signals with dynamic bus sizing for 8- or 16-bit operations. They also include 2 KB of dual-ported RAM for real-time monitoring of critical memory variables.

In addition to multiple software breakpoints, the unit supports up to four hardware events for triggering breakpoints or traces. The 4K by 63-bit trace buffer provides for direct viewing of source code and can read full address bus, data bus, chip select, and control signal information. Special features allow viewing of the trace during emulation as well as proper tracing of processor show cycles, dequeuing of pipelined instructions and support for all Port-E configurations.

The LITE emulator is packaged in a direct-plug pod enclosure (4.8" x 4.5" x 1 .5") with external connections for communication lines and power supply. Connection to target systems with unsocketed CPUs is provided using a special tristate adapter.

The initial release of the LITE emulator supports Motorola's 68330133 1/332/333 CPUs with versions for the 68340 and 'HC 16 family planned.

The LITE emulator sells for $4,995, which includes the SourceGate debugger.

Huntsville Microsystems, Inc.
P.O. Box 12415
Huntsville, AL 35815
(205) 881-6005
Fax: (205) 882-6701

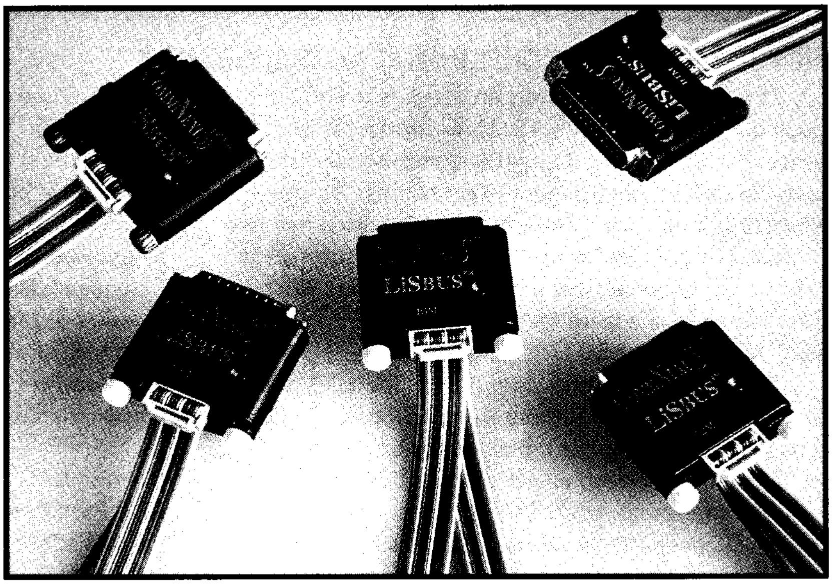
#502

OPEN SYSTEM FOR INTERCONNECTING PERIPHERALS

Gigatec S.A. of Switzerland has introduced an innovative technology for connecting numerous peripherals to one controlling computer. The **LISbus Asynch I/O System** is a cost-effective method of communicating with as many as 100 peripherals at distances up to 4,500 feet without expensive network hardware and software.

LISbus establishes a hard-wire connection through analog switching elements with any single slave peripheral chosen at any time. The connection provides eight transmission lines and one ground line under RS-232. The LISbus Link Control Software allows use of current software to communicate with the peripheral.

Gigatec based LISbus on a patented technology that uses the impedance of the bus cable to replace binary addresses. The impedance of a conductor is a fundamental physical principle, and, contrary to binary addressing, can't fail.



A LISbus Starter Pack, consisting of a Bus Interface Processor, a Bus Terminator Module, three I/O Modules, the LISbus Link Control Software, cables, power supply, and manuals sells for $995. Connect four peripherals to a controlling computer. Additional I/O modules are $130 each.

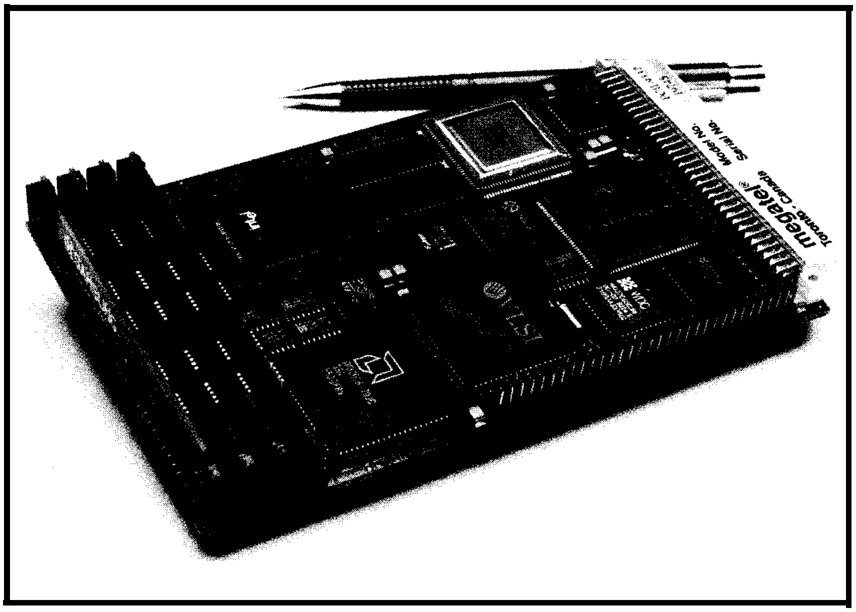
Gigatec (USA), Inc.
871 Islington St. . P.O. Box 4705
Portsmouth, NH 03802-4705
(603) 433-2227
Fax: (603) 433-5552

#503

NEW PRODUCT NEWS

EMBEDDED PC FEATURES ON-BOARD VGA CONTROLLER

Megatel Computer has introduced an i80386SX PC-compatible single-board microcomputer with an on-board VGA video controller. The **PC/II** is the only SBC currently on the market that is a full function PC. The 4″ x 6” board is also smaller than other '386SX-based embedded controllers and is ideal for applications where space is at a premium.



Features available on the PC/II include up to 16 MB of user DRAM, 32K to 256K BIOS flash memory, a SCSI host adapter, floppy disk controller, IBM monochrome and VGA/LCD-compatible video controller, and an AT-compatible BIOS. Standard features include two IBM-compatible RS-232 serial ports, one RS-232/485 multiprotocol serial port (10-MHz version), and one 9-bit general-purpose parallel I/O port with BIOS support as a printer port. Also included are PS/2 mouse support, real-time clock with battery backup, and the IBM PC bus. The PC/II uses CMOS to reduce power consumption to approximately 5 watts.

Megatel Computer provides a complete legal BIOS (in flash EPROM) that will boot standard versions of PC-DOS, MS-DOS, and DR-DOS. They also offer a choice of DR-DOS 3.1 or 5.0 on a 1 meg or 2 meg EPROM. The 5380-compatible SCSI controller supports hard drives, CD-ROMs, optical storage devices, and other functions.

The PC/II is available as a stand-alone product for $995 with 1 MB of memory or as a Developer's Kit for $1095. Besides the PC/II, the kit includes a passive backplane, transition I/O board, cable set, board jacket, and user manual. Quantity pricing is available.

Megatel Computer (1986) Corp.
125 Wendell Ave.
Weston, Ontario
Canada **M9N 3K9**
(416) 2452953
Fax: (416) **245-6505**

#504

ENCYCLOPEDIA FOR CONFIGURING MOTHERBOARDS

Thousands of main computer boards in service today have wide disparities in their configurations and jumper settings. More than 80% of these boards don't have the Original Equipment Manufacturer's name. When repairing such boards, technicians often had two choices: blindly alter jumper settings until they get it right or sell the customer a new motherboard. Micro-House has added a third option with their new ***Encyclopedia of Main Boards.***

The ***Encyclopedia of Main Boards*** includes diagrams of all necessary jumper, connection, and component locations as well as tables containing jumper setting and connection positions. The five-volume set contains additional, related information that technicians need to effect speedy repairs and upgrades.

The Encyclopedia contains the documentation for over 50 chip sets as well as CPU, NPU, and power consumption specifications. It diagrams and documents Main Board bus architectures and explains standard memory configurations and caching. MicroHouse also provides the guidelines for configuring CMOS and extended CMOS memory. It includes an illustrated glossary of Main Board terms and components and a directory of manufacturers' addresses plus phone, fax, and BBS numbers. A separate index volume and indexing software allows for quick searches for information.

MicroHouse
4900 Pearl East Circle, **#101**. Boulder, CO 80301
(303) 443-3389 . Fax: (303) 443-3323

#505

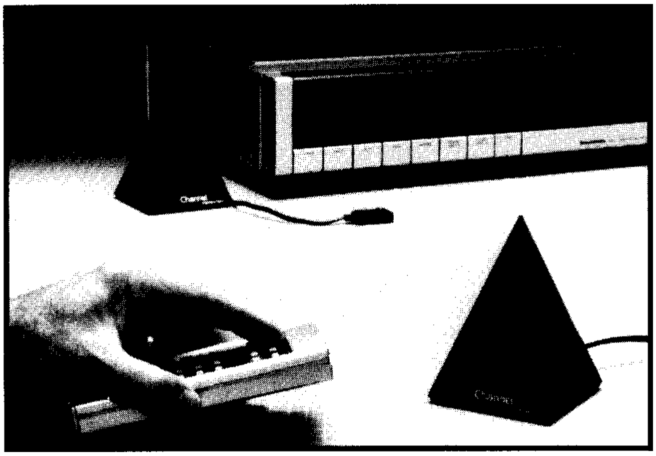
NEW PRODUCT NEWS

INFRARED WIRELESS RELAY SYSTEM

The Infrared Wireless Remote Control Relay System from Channel Plus easily controls a VCR, a laser disc player, or another video source. If you use the Channel Plus modulators to distribute video signals to household TVs, you can watch and control a video source from any room.

The IR Target/ Repeater System 2000 consists of an IR target, IR repeater, and IR LED emitter extension. The IR repeater and IR target will communicate up to 100 feet. Where placement in front of the video sources is not convenient, Channel Plus provides a foot long IR emitter extension. Both the IR repeater and IR target plug into a 120-volt output. The system will work with any IR remote supplied with the video source equipment.

An IR repeater, needed for each location of a video source, can send IR signals to all video sources that can "see" the repeater. Use an unlimited number of IR targets with one or more IR repeaters.

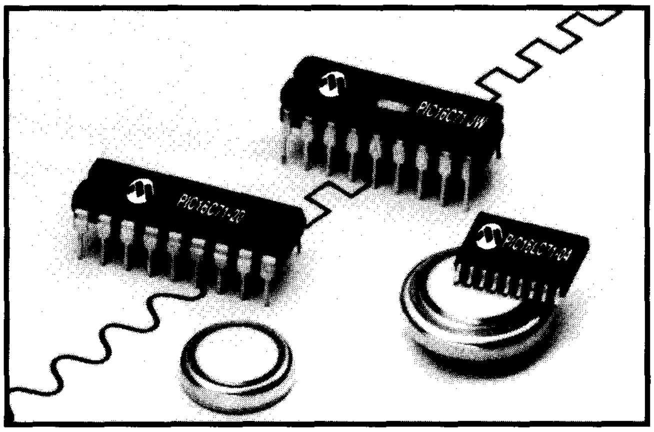


You control video devices from their hand-held remote control units. When you point the remote control at the IR target, it sends its signal through the air to the IR repeater in another room. The IR repeater repeats this signal, which the video source sees just as if the remote was in the same room.

Prices for the IR Target/ Repeater System start at $70. Channel Plus products are manufactured by Multiplex Technology.

Multiplex Technology, Inc.
3200 E. Birch St.
Brea, CA 92621-6258
(714) 996-4100
Fax: (714) 996-4900 **#506**

NEW PRODUCT NEWS



USER-PROGRAMMABLE RISC MICROCONTROLLER FEATURES ADC

Microchip Technology has introduced an advanced, low-cost, 8-bit RISC microcontroller with on-chip A/D conversion and an upgraded CPU core. The PIC16C71 features internal and external interrupt sources, an expanded stack memory, and 14-bit-wide instructions. The chip maintains the advantages of user programmability while adding a four-channel, 8-bit ADC with a conversion rate of 20 µs per channel.

The RISC instruction set consists of 35 instructions, each of which executes in one 200-ns instruction cycle (with the exception of program branches). Capable of operation up to 20 MHz, the PIC16C71 uses a dual-bus Harvard architecture with a 14-bit-wide instruction and an g-bit-wide data path. The high-performance core incorporates 1024 x 14 of program EPROM memory and 36 x 8 general purpose SRAM registers along with four interrupt sources, including wake-up on keystroke.

Other features of the PIC16C71 include 12 special-function registers, expansion from a two-level-deep to an eight-level-deep stack, 8-bit real-time clock/counter with 8-bit programmable prescaler, and a high-current sink/source for direct LED drive. Special features include a watchdog timer, serial programming for easy in-circuit software configuration, a security EPROM fuse, power on reset, and EPROM fuse-selectable oscillator options.

The PIC16C71 sells for $3.25 in quantities of 10,000.

Microchip Technology, Inc.
2355 W. Chandler Blvd. . Chandler, AZ 85224-6199
(602) 963-7373 . Fax: (602) 899-9210

#507

NEW PRODUCT NEWS

PC-COMPATIBLE EMBEDDED CONTROL SYSTEM

MJS Designs has introduced a new tool for embedded control using an IBM PC or compatible. The Total Control 2000 (TC2000) is a unique way for engineers to develop standard microcontroller functions and for experimenters to create the "perfect" electronic home. It incorporates the features of a learning tool, hardware developer, and programmer's kit.

The TC2000 kit includes an ISA-compatible PC card that has standard I/O features, such as 10-bit ADC, 16-bit pulse with modulators, edge detector inputs, asynchronous serial I/O, a 32-bit timer/counter with 1-µs resolution, and general-purpose I/O. The kit also includes a Windows 3.0/3.1 click-on driver for easy hardware debugging and a generic ANSI standard C library source code for controlling over 33 functions.

The Windows driver software simplifies program development. Instead of remembering bits in a register to control serial I/O or high-speed capture inputs, configure I/O by clicking a mouse button. Also, use the driver to debug hardware without software errors getting in the way. Create control programs using standard PC C development tools.

Uses for the TC2000 include digitizing sound using the ADC, sound playback through the Pulse Width Modulator, data transmission from the PC to other serial devices with rates up to 1 Mbps, as well as simple turn on and off control functions.

The TC2000 kit includes the TC2000 ISA-compatible PC card, 4 feet of ribbon cable, documentation, the click-on driver software that runs under Windows, and the ANSI standard C source library routines. The kit sells for $249.99.

MJS Designs
1438 W. Broadway, Ste. B185
Tempe, AZ 85282
(602) 966-8618
Fax: (602) 966-8528

#508

FEATURES

FEATURE ARTICLE

H. Brooke Stauffer

The SMART HOUSE System

A Technical Overview

There are a number of home automation technologies vying for market share. SMART HOUSE was developed as a proprietary system initially aimed at new construction. Find out just what makes SMART HOUSE tick.

SMART HOUSE is a complete *enabling system* that provides the common resources needed for home automation in a multiproduct, multivendor environment-a system controller, a housewide wiring network, communications protocols, standard interfaces [outlet designs] for connecting other products, and basic user controls. These controls include programmable wall switches, a control panel, and a DTMF telephone. Providing complete home automation functionality depends on the addition of other products, such as more complex user controls, home appliances, and application-specific controllers.

These SMART HOUSE-compatible appliances and controllers are equipped with a proprietary communications circuit, the ***branch slave chip,*** which permits them to communicate with the system controller and with each other.

A basic building block of the design is a system of three multiconductor cables installed during original construction in place of conventional house wiring. This cabling system combines power, control, telephone, and coaxial conductors and provides a dedicated six-wire bus throughout the house. This home automation en-

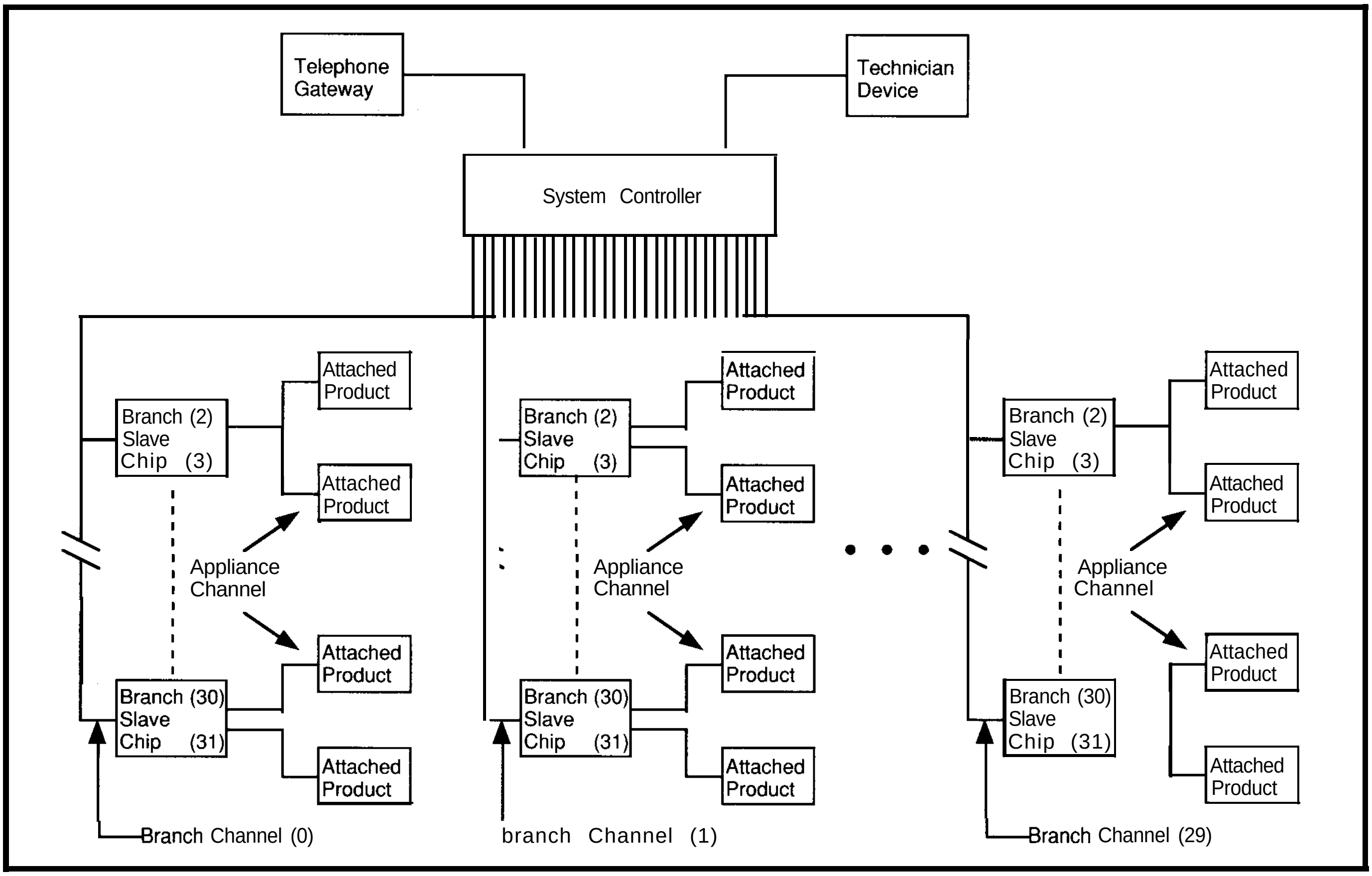


Figure l--The *control/communications subsystem consists of three main components--the system controller, the dual branch slave chip, and the appliance chip-and three communications channels-the branch channel, the appliance channel, and the serial channel.*

abling system is intended primarily for new home construction, though a retrofit design is under development.

I'd like to give you a concise technical description of the SMART HOUSE system, concentrating on control/communications aspects such as system architecture, protocols, messaging, and logic structure.

SYSTEM ARCHITECTURE

The SMART HOUSE control/communications subsystem is responsible for the coordination, operation, and management of house functions. Its two primary functions are controlling energy delivery to outlets and appliances, and providing pathways for appliances and system components to exchange messages and functional requests. It exchanges data and messages between appliances and the system controller or between two or more appliances, using the Core Language and the object-oriented SMART HOUSE Applications Language (SHAL).

The control/communications subsystem consists of three primary components and three communication channels (see Figure 1). The three components are

*the system controller
*the branch slave chip
*the appliance chip

The communications channels are

•the branch channel
*the appliance channel
*the serial channel

COMPONENTS

System Controller

The system controller manages the SMART HOUSE control/communications system. It consists of a number of I/O ports, two microprocessors, a real-time clock, and volatile and nonvolatile memory. The system controller is implemented as a single electronics and interface connector board. Included are the hardware and software necessary for managing communications and control functions on two serial channels and up to 30 branch channels. Each branch channel supports up to 30 appliance channels, so the system controller can manage as many as 900 uniquely addressable nodes. Each node represents an outlet or appliance attached to the control/communications subsystem.

The system controller maintains and permits access to internally stored database information, including

- Node addressing information
- System clock (date and time)
- Appliance identifications
- Switch/sensor and appliance statuses
- Controller status
- Appliance report mappings
- Programmable control logic
- Allocated coax transmit channels
- Telephone gateway configurations
- Remote access privileges for remote service providers (energy and communications utilities)

Branch Slave Chip

The branch slave chip constitutes two addressable nodes, each of which supports an appliance channel (for more information, see the "Communi cations Channels" section). The branch slave chip provides the necessary protocol conversion between the appliance and branch channels. It also sends messages and commands to and from the system controller. Branch slave chips are installed in system outlets and in system devices like the remote control circuit breaker unit. The branch slave chip also contains circuitry to control latching relays.

Appliance Chip

SMART HOUSE-compatible attached products include equipment normally hard-wired (hot water heater, central air conditioning) and appliances normally plugged into wall outlets [consumer electronics, washer or dryer). These appliances may or may not contain a SMART HOUSE appliance chip. The appliance chip enhances the communications abilities of "normal" and "complex" attached products; "conventional" and "simple" attached products do not use the chip.

The appliance chip implements the appliance channel's physical and link layer protocols. In normal mode, the appliance chip allows an attached product to send and receive 8-bit ***appliance reports*** to and from the system controller. In complex mode, the appliance chip provides a standard microprocessor or microcontroller interface for sending and receiving Core Language messages to and from the system controller.

COMMUNICATIONS CHANNELS
Serial Channels

The system controller supports two dedicated EIA-232-D asynchronous channels, each operating at 9600 bps, full duplex. One channel communicates with the telephone gateway, making control of the system by touchtone phones possible. The second serial channel communicates with the SMART HOUSE technician device-a laptop computer equipped with proprietary software that configures and troubleshoots the system.

Branch Channels

The branch channel is a synchronous serial interface between the system controller and an appliance channel representing an addressable node (outlet or appliance). The system controller can support up to 30 branch channels. Each branch channel is capable of supporting up to 30 nodes for a total of 900 addressable nodes. Each branch slave chip supports two independent nodes, so a single branch channel can have up to 15 branch slave chips attached to it. A polling protocol invokes link layer data exchanges between the system controller and the branch slave chips. To ensure system integrity, only devices containing branch slave chips may attach to the branch channel.

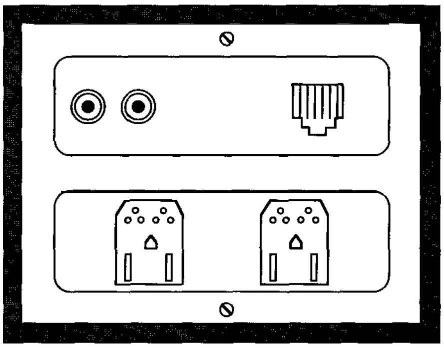


Figure 2-The SMART Convenience Center rep/aces the traditional wall outlet and includes connections for power [conventional and SMART), coax, and telephone.

Physically, six dedicated conductors in two different multiconductor cables distribute the branch channel throughout the house. These cables run from the system controller to outlets for plug-in appliances and directly to hard-wired appliances.

Appliance Channel

The appliance channel communicates between attached products and branch slave chips located at outlets. Appliances, sensors, switches, and application-specific controllers (such as security or energy management) send and receive control information, status information, and Core Language messages over the appliance channel to the branch slave chip. The branch slave chip provides the necessary protocol conversion between the appliance and branch channels and sends messages and commands to and from the system controller.

The appliance channel isolates attached product failures, so a failure at one node will not affect other nodes on the SMART HOUSE control/communications subsystem. Physically, six conductors in appliance cord sets equipped with a special SMART HOUSE attachment plug that connects to system outlets distributes the appliance channel (see Figure 2).

COMMUNICATIONS PROTOCOL

The control/communications subsystem distinguishes between control and communications information. Control information includes appliance status, diagnostic, power control, and other discrete (1 byte or less) information. Communications information is defined as the passing of Core Language messages and associated data.

A transport layer protocol ensures the accurate delivery of message packets. A proprietary application layer protocol handler implements the Core Language and executes commands from attached products and remote service providers [energy and communications utilities). Other handlers execute programmable control logic resident on the system controller. The Core Language includes functions for transporting SHAL from appliance to appliance and between appliances and the system controller.

Addressing

All components containing branch slave chips are manufactured with a nonvolatile 1 5-bit ***long address;*** typical long addresses would be 38353 or 17028. During the system addressing process using the technician device, each branch slave chip issues this 15-bit long address to the system controller with an appended 16th bit (0 for the first node of the branch slave chip and 1 for the second node). The system controller stores these two 16-bit addresses and assigns a unique S-bit ***short address*** to each of the two nodes.

The system controller can uniquely identify any node in the house by prefixing a 5-bit number representing the branch channel [physical six-wire bus) associated with that node to the 5-bit node short address. This 10-bit representation is referred to as the ***physical address*** or

branch/node number. Each of the up to 900 appliance channels in the system has a unique physical address or branch/node number; a typical physical addresses would be 02/13.

Programmable Control

The system controller periodically scans programmed "event" conditions and ascertains if and when to execute the corresponding programmed "action" logic. Programmable control logic can take one of two forms. It may be structured as a dynamic event/action sequence or as high-priority event/action logic that controls only power and control bit states.

Dynamic Event/Action Control

Dynamic event/action logic consists of one or more specified events followed by one or more specified actions. The system controller makes a number of comparisons and detections during its normal operation. Each of these constitutes a possible event entry that can be configured into the dynamic event/action table and scanned periodically by the system controller. Core Language messages, including transmission of applications language messages, constitute actions for the dynamic event/action structure. Some message types are not allowed to be action types for reasons of system integrity.

High-Priority Event/Action Control

The high-priority event/action table is used for programming events that control power delivery or control bit state at a specified node. High-priority event/action logic is created by technicians or homeowners using the technician device or user interfaces like control panels. This table cannot be used to execute Core messages.

ATTACHED PRODUCT OPERATIONAL MODES

The SMART HOUSE system accommodates a wide range of attached product complexity. In order to provide for differing levels of operational complexity and reduce requirements for electronics within these products, the control/communications system supports four product ***operational modes***—conventional, simple, normal, and complex.

Simple, normal, and complex attached products are collectively referred to as ***SMART appliances.*** Their capability to communicate with the system controller distinguishes them from conventional appliances.

Conventional Mode

Conventional mode or nonsmart appliances require only 120 VAC or gas energy. This type of appliance connects to a system outlet with a standard AC power plug or gas appliance connector.

Conventional mode appliances cannot communicate with the system controller. However, the controller can detect when an electrical appliance is attached to a 120 VAC outlet and use status changes to trigger the execution of event/action logic. For example, if a conventional electrical appliance suffers a fault or becomes disconnected from its AC outlet, the system controller can detect this problem and display a message on a control panel.

Simple Mode

Simple mode appliances do communicate with the system controller, but don't require an appliance chip. They can convey up to seven states (on, off, neutral, etc.) to the controller and receive binary control commands. The system controller reserves the eighth state for internal use.

Simple mode appliances communicate status and control information to the branch slave chip in the system outlet to which the appliance is attached. Such communication is done using three status lines and one control line in the appliance channel (appliance cord set). The attached product communicates "upstream" to the system controller over the three status lines with a signal called the *appliance status.* The system controller communicates "downstream" to the product over the control line, which is referred to as the *control bit.*

An example of a simple mode appliance is a programmable SMART wall switch. The system controller monitors changes to the switch's upstream appliance status, and directs downstream power control commands to controlled nodes (AC outlets, light fixtures, hard-wired appliances, or gas outlets). The smart wall switch can use the control bit to light an LED representing the ON or OFF power state of a controlled node.

A temperature sensor is another simple mode implementation (Figure 3). The sensor uses the appliance status lines to indicate one of three conditions to the system-over setpoint, nominal, or under setpoint. Using dynamic event/action logic, the system controller monitors changes in the sensor's status and issues commands to a gas or electric furnace.

Normal Mode

A normal mode appliance uses a SMART HOUSE appliance chip to send and receive 8 bits of data to and from the system controller in addition to the status and control data described above for simple appliances. These data, called an *appliance report,* may also be sent to another normal mode appliance and may be queried or modified by complex mode appliances issuing Core Language messages. This function, called *appliance report mapping,* is useful to product designers because it provides enhanced inter-appliance communication at low cost.

Appliance report mapping is effectively a version of programmable control in which the system controller takes the upstream appliance report issued by one normal mode appliance and periodically transfers it to other appliances. The system controller executes these transfers whenever it receives a change in a designated

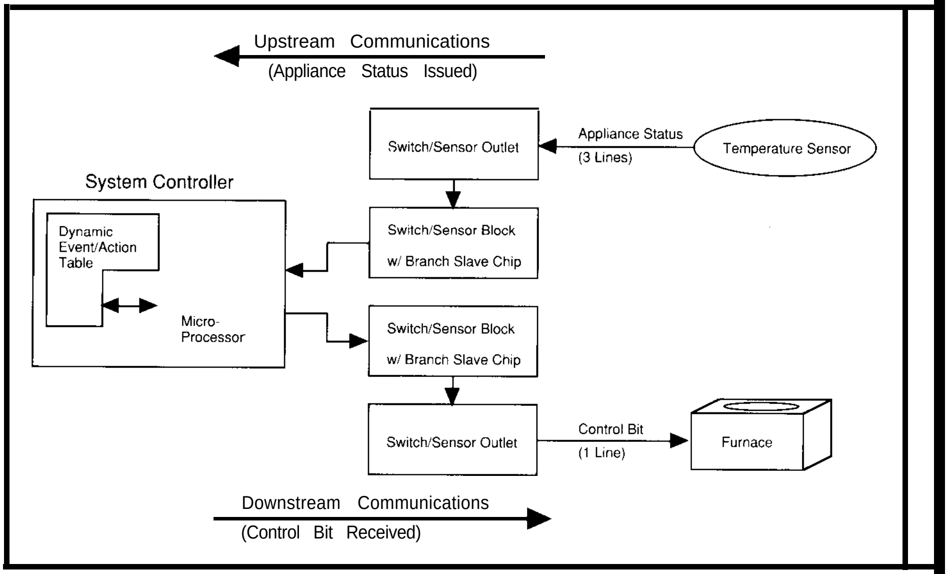


Figure 3—*A simple mode example is a temperature sensor. The sensor can use the appliance status lines to indicate one of three conditions: over setpoint, nominal, and under setpoint.*

upstream appliance report. One upstream appliance report can be mapped to multiple downstream locations.

Product manufacturers determine the encodings for 8-bit appliance reports; for example, eight ON/OFF control inputs to other appliances, two hex digits, a binary number from 0 to 255, or an ASCII character. The system controller does not interpret the contents of an appliance report. It simply routes the data to another normal mode appliance, which is responsible for interpreting the data or performing some action based on the data.

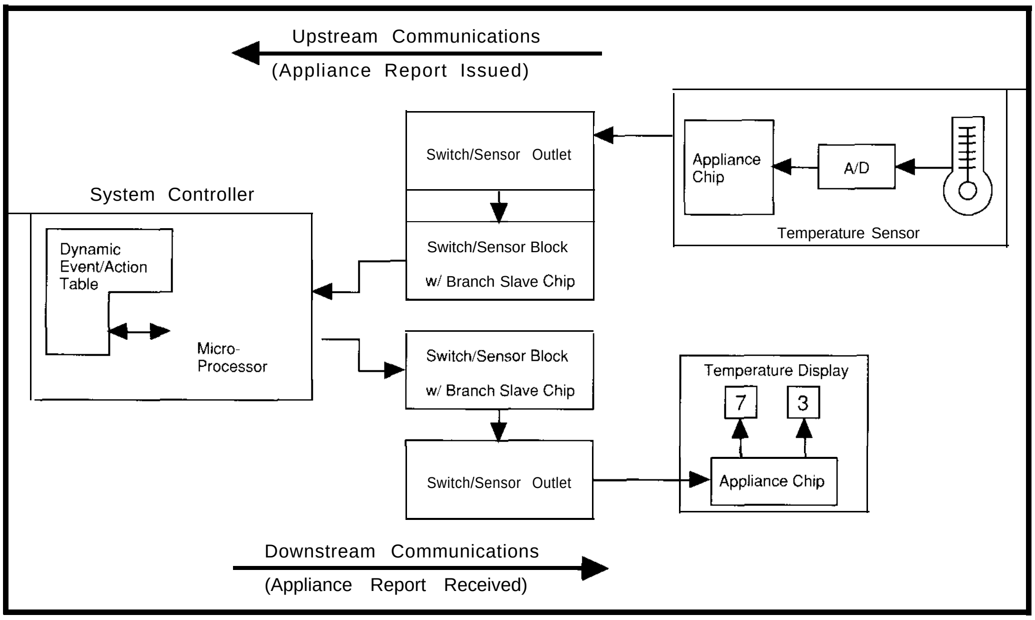


Figure 4-A example of normal mode operation involves an exterior temperature sensor that drives an internal digital display

An example of a normal mode appliance is a programmable eight-button switch panel that controls selections of *house modes,* which are preset system configurations for lighting control, HVAC setback, and so forth. The switch panel uses an upstream appliance report to identify which of the eight possible, mutually exclusive house modes has been selected by the homeowner. Each switch on the panel signals a binary status (open or closed) to one bit of the upstream appliance report. The system controller interprets a status change in a bit as a request to change the house into a different predefined mode.

A second example of normal mode operation involves an exterior temperature sensor that drives an internal digital display; both are made by the same manufacturer (see Figure 4). The system controller takes the upstream appliance report issued by the sensor and transfers it to the display, which decodes the data to show temperature.

Complex Mode

A complex mode appliance uses the SMART HOUSE appliance chip in conjunction with a microprocessor to exchange Core Language messages with the system controller or other complex appliances. The microprocessor used is the one that would be in the product anyway, with an added interface for the appliance chip.

One example of a complex mode appliance is a clothes washer that sends a message for display on a TV set indicating a completed wash cycle. Another is an environmental monitor that displays messages on a touchscreen control panel (see Figure 5). The microprocessor in the environmental monitor uses inputs from three sensors to compile a Core Language message, which the appliance chip sends to the system controller. The controller routes this Core Language message to the touchscreen, where it is interpreted to display three separate outputs to the homeowner.

TRANSPORT LAYER PROTOCOL

The transport layer protocol exchanges control and data packets between an appliance and the system controller. It acknowledges and retransmits messages, provides stop-and-wait flow control, and piggybacks acknowledgments to conserve packets. Data packets exchanged by the transport layer contain between 2 and 62 bytes of application layer data.

CORE LANGUAGE

The SMART HOUSE Core Language is an application layer protocol that exchanges commands, responses, and notifications between an appliance and the system controller. The language includes over 100 message types, each performing a specific function. Using the Core Language, complex mode appliances can request system services such as the following:

- Obtaining the time and date
- Controlling power delivery at outlets
- Communicating with other appliances
- Assigning switches to outlets or hard-wired appliances
- Creating programmable control logic
- Placing and receiving calls via the telephone gateway; for example, a security controller might place an alarm call to a central monitoring station

OBJECT-ORIENTED SMART HOUSE APPLICATIONS LANGUAGE (SHAL)

Using the Core Language, complex mode appliances can communicate

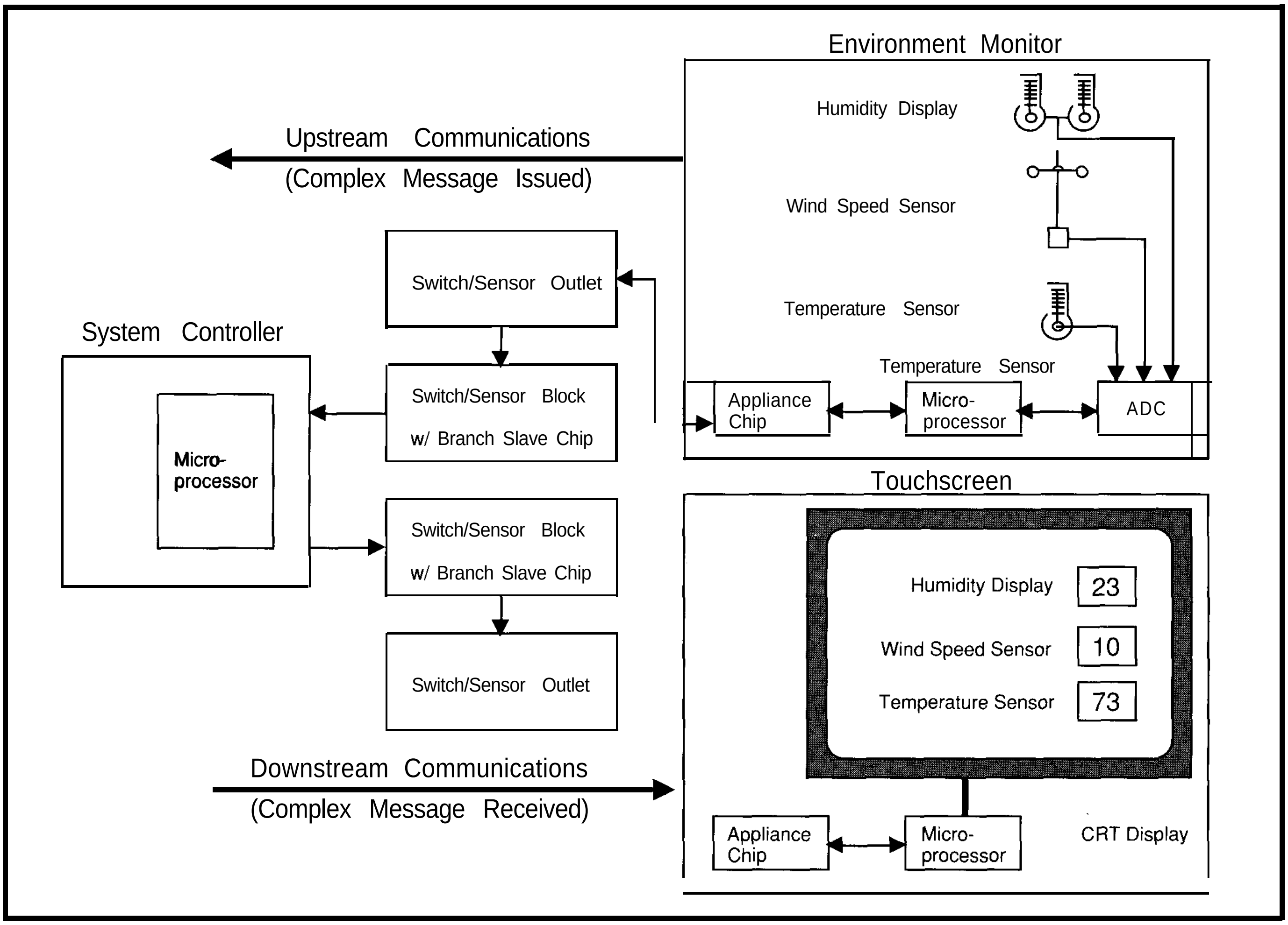


Figure 5—*An example of a complex mode appliance is an environmental monitor that displays messages on a touchscreen control panel.*

information among themselves or to the system controller. SHAL standardizes the format for this information exchange.

When an appliance sends a message to the system controller or another appliance, it first constructs an applications subpacket. This subpacket has a header byte that defines what type of message it is and whether a response or an acknowledgment is required. If a response is required, a response ID is also sent in this header byte; the response message then transmits this ID back to the sender for identification. The second byte in the packet is always an *object class-name code,* followed by the *object identifier code* in the third.

For example, if the message is to be sent to a VCR, then the second and third bytes in the message are the object class-name code for "unit" followed by the object identifier code for "VCR." This information is followed by the actual message to the VCR, which may be up to 58 bytes long (bytes 4 through 62). SMART HOUSE L.P. assigns object class-name and object identifier codes. The manufacturer specifies the messages a particular appliance [such as a VCR) can receive and respond to. These messages can be listed in a data sheet for other product manufacturers' information.

SMART HOUSE USAGE

SMART HOUSE is currently aimed at the new construction market with professional builders and installers the intended buyers. Equipment for retrofits is still being developed at this time. For more information on SMART HOUSE, contact SMART HOUSE L.P., 400 Prince Georges Blvd., Upper Marlboro, MD 20772, (301) 249-6000. ❑

Brooke Stauffer is Technical Director, Regulatory Affairs for the Association of Home Appliance Manufacturers (AHAM). He was previously Manager of Codes and Standards Development for the SMART HOUSE Limited Partnership.

SOURCE

Stauffer, H. Brooke and Mullin, Ray C., SMART HOUSE *Wiring,* Delmar Publishers, 1992. Available for $21.95 (plus shipping) from Delmar Publishers at (800) 998-7498.

Just published in November, this new book is the comprehensive guide to designing and installing the SMART HOUSE system for new homes. Three new cables, special wall outlets, and an integrated distribution system called the service center put all the electrical and electronic services together in one neat package. The SMART HOUSE system is installed during original house construction in the place of conventional wiring.

IRS

401 Very Useful
402 Moderately Useful
403 Not Useful

#116

Switched-On CEBus: A CAL Interpreter

FEATURE ARTICLE

Jeff Fisher

The key to future home automation products will be a standard means of communication. CEBus shows promise of being that standard. Find out what the lanuguage of CEBus looks like and how it is used.

Are you ready to learn a new programming language? Don't worry. This one is unique enough to be interesting, simple enough to learn in an afternoon, and powerful enough to be useful in a variety of situations. It's called the Common Application Language [CAL), and it's part of the top layer of the EIA CEBus home automation standard. Ken Davidson's CEBus article in this issue has an extensive list of references should you need some background on CEBus. I'd like to introduce you to this nifty little language and describe a modest implementation of a CAL interpreter.

CEBus follows the seven-layer IS0 network model. While layers one through six define how to move data around, layer seven, the *application layer,* defines how an application uses the data. Parts of the CEBus application layer involve connectivity and resource allocation, but the nucleus is a true language.

What's a language doing in a home automation standard? Adding flexibility. By adding remote programmability to a device, you can teach it to do things that its designers never envisioned. An admirable goal, but what kind of language do I want my light switch to use? Obviously, it must be very simple to parse and execute, it must be terse, and it must be flexible. CAL is all these and more.

CAL EXPLAINED

First, I have to warn you that I haven't been able to get on the EIA's magic mailing list for CEBus updates, and I got tired of regularly paying $60 just to find out if anything changed. Therefore, my version of the CAL specification is probably out of date. Also, the CAL specification is vague in a few areas much to my frustration. Fortunately, the language is fairly elegant. With this benefit, and calling on some experience with LISP interpreters, I made some educated guesses to fill in the blanks. Finally, I only aim to introduce you to CAL. Read the EIA specification for complete details.

The EIA designed CAL for computer-to-computer communication, so isn't very human readable. A program could easily convert CAL to printable text, but the EIA didn't define a format for this "source" version. The language is very terse, with complete programs as short as 2 bytes. CAL is this concise because it is simply a sequence of 1- to 2-byte tokens, defins defaults for most everything, and is object oriented.

Now, OOZs [Object-Oriented Zealots) shouldn't get too excited, and OOLs (Object-Oriented Loathers) shouldn't dismiss CAL; the EIA uses the term "object oriented" loosely. There are no classes, class hierarchies, or inheritance. Put simply, the current *state* of the interpreter affects what a given opcode does. More accurately, you can put CAL tokens in just about any order, but the meaning of a given token can be drastically affected by the token placed before it. There are two levels of state: context and object. Contexts represent different sections in a device, such as the tuning and amplifier sections in a stereo. Objects represent different parts of a context, such as the volume and balance controls in the amplifier context.

The EIA calls opcodes *methods.* The CAL specification provides a list of generalized methods that acts like a master instruction set. The manufacturer implements only those methods that make sense for each object. For instance, a volume control might have plus and minus methods to raise and lower the volume setting, while true and false methods might turn a bass boost object on and off.

The EIA defines some common contexts, their objects, what methods they should implement, and what they should do. However, this list is not and never will be exhaustive. The truth is manufacturers will pick their own contexts and objects, decide what methods to implement in each object, and choose the actions each method performs. I only hope that devices receiving CEBus messages will come with documentation on the contexts, objects, and methods that they understand. Likewise, devices that send CEBus messages should not hard-code the CAL portion of the message.

The specification includes a complete, two-page Backus-Naur Format (BNF) description of CAL. My dilemma? How was I going to put YACC (a compiler writer's tool that uses BNF input) onto an 8051? After staring, glassy eyed, at the BNF notation for a couple of hours, I started writing code. My first attempt was big, complex, and wrong. The next attempt was smaller, faster, and worked better. By the third rewrite, I had it nailed. The key was converting the BNF notation into a table that a parser could use (Table 1). This table is not equivalent to the BNF; it allows a syntax that the BNF says is illegal, but it will properly parse all legal constructs, which is what I wanted.

PARSING CAL

A CAL parser gets the next byte from the CAL program by looking it up in the left column of the table, and it returns the specified return type and return value. Thus, 22 parses to a type of *value7* and a value of 22. I'll write this statement as *value7*:22. The code F5 FF parses to *value8*:FF.

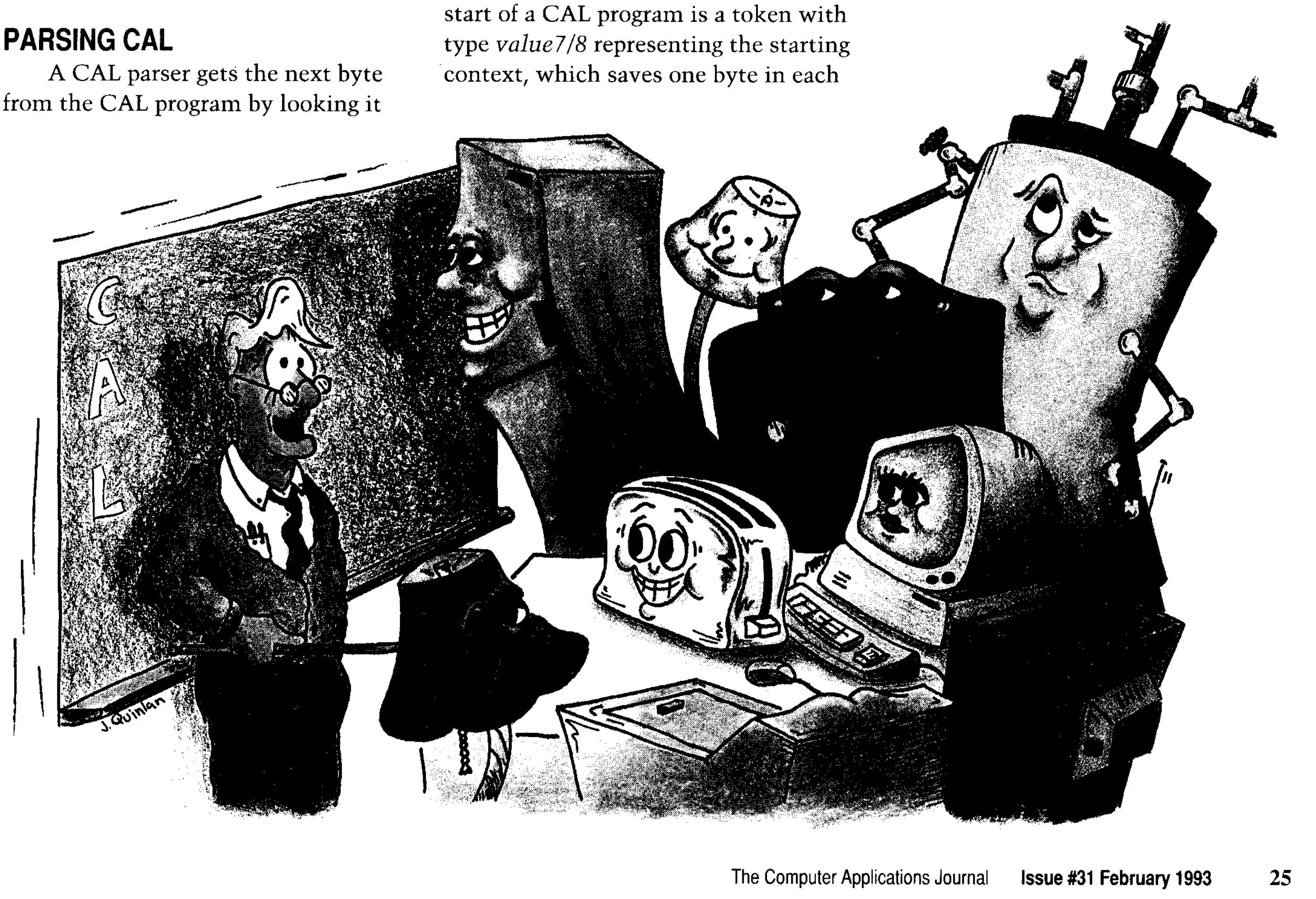
To parse a context (F6) token, the parser calls itself recursively to get the next token. The token returned must be type *value7* or *value8* (*value7/8* for short). The parser returns the type *context* and the value returned by the recursive call. Thus, F6 01 returns *context*:01, and F6 F5 F2 returns *context*:F2.

UDASE (F7) and data (F4) both parse the same way. The parser calls itself recursively to get the next token. The token returned must be type *value7/8*. This value represents the length of the data or UDASE (user-defined message) block that follows. The parser sucks up that many bytes from the CAL program, and returns them along with the type (*UDASE* or *data*). Therefore, F4 05 00 01 02 03 04 parses to *data* <00 01 02 03 04>, and the code F4 F5 80 00 01 02...7D 7E 7F parses to *data* <00 01 02...7D 7E 7F>. That's 128 bytes of data.

There is one special case. The start of a CAL program is a token with type *value7/8* representing the starting context, which saves one byte in each message (an initial F6). For example, 01 DO 80 F6 02 D1 81 parses to *context*:01, *object*:D0, *method*:80, *context*:02, *object*:D1, *method*:81.

The method argument list follows a method ID. Note the division in Table 1. You can't use the types above the line in an argument list. An argument list is zero or more tokens, optionally separated by the delimiter token FA. The method uses the default argument when the CAL code does not specify the argument (and there is only a delimiter with no data value or the argument list ends prematurely). For example, 01 D0 80 22 FA 23 calls *method*:80 with *value7*:22 for argument 1 and *value7*:23 for argument 2. In another example, 01 D0 80 FA 23 calls *method*:80 with the default for argument 1 and *value7*:23 for argument 2.

According to the specification, the current context and object affect the default arguments to a method! Therefore, the routine that gets an argument for a method first looks in the CAL program for the argument. If it doesn't find one, it uses the current context and object to look up the default argument.



INTERPRETING CAL

To execute a CAL program, the interpreter repeatedly calls the parser, acting on each returned type. It makes note of any context or object changes. For method types, it uses the current context, current object, and method ID to decide which method to call. The method calls the parser to get any arguments it needs. When the method returns, the interpreter skips any method arguments by parsing until it encounters a nonargument type (see the top half of Table 1). The interpreter repeats this process until it runs out of code to parse.

Each context has a default object, and each object has a default method. The interpreter calls the default method when it encounters a *value7/8* type. The value becomes the first argument to the default method. The ETA calls it, confusingly, an *alias message.*

Don't give up now, you're almost there. A few examples will help you. The code 01 80 calls *method:80* in the default object of context 01. As an example of an alias message, the code **01** DO 20 calls the default method for *object*:D0 in *context*:01. *value7*:20 is the method's first argument. The interpreter calls the default method because it encounters a *value7* type.

A final description completes the CAL syntax discussion. CAL has provisions for variables. The EIA calls a variable name a *reference string.* (The word "string" is a complete misnomer-ignore it.) A reference string is simply a *value7/8*-type argument. The argument's value (typically an ASCII letter like *A)* refers to a variable associated with the current context and object, which is an *instance variable* in object-oriented parlance. The specification says almost nothing about instance variables, and CAL uses them heavily, so I was pretty much on my own with them. Unfortunately, reference strings do not have a unique type; methods must be hard coded to assume some arguments are reference strings and some are immediate data values. This situation gets a little confusing because CAL requires multiple versions of each method to handle the combinations.

Let me give a real-world example. Table 2 shows a 16-byte CAL program that uses real contexts, objects, and methods as described in the specification to control a hypothetical television set. Follow along in the table as I describe the program. The first byte indicates the initial context: audio context. When the context changes, the interpreter sets the current object to the default object of the new context. The second byte indicates a new object: volume control object. The third byte is the method ID for the *add immediate* method. The add immediate method adds argument 1, a data value, to the reference string indicated by argument 2. Byte 4 is the first argument to the add immediate method. Because the CAL code does not specify argument 2, the interpreter uses the default, which is the current volume level. Thus, the third and fourth bytes increase the current volume level by two "notches."

Byte 5 calls the swap method, which exchanges the contents of the two reference string arguments. Byte 6 is the first argument: a reference string that indicates the mute level variable. The code does not have a second argument, so the interpreter again uses the default: current volume level. Thus, bytes 5 and 6 swap the mute level with the current volume level.

Byte 7 indicates a new object: the feature switch object. Byte 8 calls the *true* method. The true method sets the variable referenced in argument **1.** Byte 9 is the argument, and an *S* represents the surround sound variable. Setting this true enables surround sound.

Byte **10** indicates an upcoming change in context. Byte **11 is** the new context. Byte 12 selects the sharpness control object. Byte 13 calls the *loud* method. The load method has two arguments, both reference strings. It copies the contents of the first variable to the second variable. Byte 14 is argument 1 for the load method, and a *P* represents the preference variable. Because the CAL code does not specify argument 2, the interpreter uses the default: the current sharpness level. Thus, bytes 13 and 14 load the preference sharpness level into the current sharpness level.

Byte 15 is the escape token. It indicates the end of an argument list, so an alias message that follows will not be taken as part of the current method's argument list. Byte **16** is an alias message. In this context and object, an alias message sets the current sharpness level variable to the first argument value. Therefore, byte **16 sets** the current sharpness value to hex 20.

In a mere **16** bytes, you raised the volume level, muted the sound, turned on surround sound, reset the sharpness to the preferred level, then set the sharpness to a specified level.

Byte	Return Type	Return Value
80..CF	method ID	Current byte
DO..EF	object ID	Current byte
FC	escape	
F7	UDASE	Parse again. Must be type value7/8. Use as length of UDASE that follows.
F6	context	Parse again. Must be type value7/8. Return this value.
00..7F	value7	Current byte
F5	value8	Next byte
F4	data	Parse again. Must be type value7/8. Use as length of data that follows.
FO	null	
F1	minimum	
F2	maximum	
F3	default	
F8	bitstring	
FA	delimiter	
FB	endoffile	
FD	begin	
FE	end	

Table 1—*The* key *to* efficiently *converting* the *BNF* CAL *format to something the* parser can *easily handle is to put the information* **in** *table form.*

IMPLEMENTING CAL

Space prohibits my listing or describing all the contexts, objects, and methods defined in the specification, but I will tell you that methods exist for testing, branching, calling, returning, if, and so forth. The EIA appears to have added some of the more interesting methods, such as *call, return,* and *if,* as an afterthought; the standard just gives their name and method ID. The specification also doesn't bother to mention what the tokens *null, minimum, maximum, default, bitstring, endoffile, begin,* and *end* mean. Also, it doesn't mention anything about call stacks, if methods return tokens that can be arguments to another method, or how a CAL program returns anything to the caller. The specification also implies that you can call a CAL program stored in a variable. I guess, "..implementation is left as an exercise for the reader." (As I've said, I'm not working off the latest version of the spec, so they may have added more information by now.)

So I exercised. I made up argument lists for the methods without them. I used begin and end to bracket a "block" of CAL code, allowing a block of anything to become one argument. **(So** the method *if* can call code that is in its arguments.) I have all methods leave a token in the return value. Also, a call stack allows *call, return,* and *exit.* I haven't come up with a use for *null, minimum, maximum, default, endoffile,* or *bitstring* types yet.

Using begin and end to create code blocks was easy. When the parser hits a begin, it calls itself recursively until it hits an end. A flag to the parser tells it to treat begins as block starts or separate tokens, which gives me a new type from the parser called *beginend.*

A single data structure, calPtrs, holds any type of token (refer to Listing 1). The calTokenType, an enum, is held in type. For simple tokens with no value, that's it. val (#definedtolen) holds the value for value tokens, such as *object, method, context, value7,* and *value8.* msg holds a pointer to the start of the data, and len holds the length of the data for the variable-length tokens *UDASE, data,* and *beginend.*

There are four calPtrs structures. calPtr is the main one, pointing to the remaining CAL code. calRet holds return values. calContextList points to the context list (always a *beginend).* calVarList points to the variable list (always a *beginend).* The variables calContext and calObject hold the current context and object ID.

The key to handling object-oriented (context and object dependent) method references, default

Byte No.	CAL Code	Description	Current Context	Current Object
1	30	Start off with the audio context	30	default
2	DB	Select volume control object	30	DB
3	87	Call the add immediate method	30	DB
4	02	Argument one is "2"	30	DB
5	8E	Call the swap method	30	DB
6	50	Argument one is *P* (mute level)	30	DB
7	D5	Change to feature switch object	30	D5
8	81	Call the true method	30	D5
9	53	Argument one is *S* (surround sound)	30	D5
10	F6	Change context...		
11	38	...to video monitor context	38	default
12	DF	Select the sharpness control object	38	DF
13	8A	Call the load method	38	DF
14	50	Argument one is *P* (preference)	38	DF
15	FC	Escape the current method	38	DF
16	20	Call the default method (an alias message)	3 8	DF

Table *2-/n* a *mere 16 bytes, you can raise the volume, mute the sound, turn on the surround sound, resef the sharpness to a preferred level, and set the sharpness to a specified level on a hypothetical television sef.*

method arguments, and variables is in the structure of the context and variable lists. Because I already have a CAL parser, I thought, why not store all this data in CAL format? That way I can search with the parser.

So calContextList points to a hard-coded CAL string. (See Listing 2 for example context and variable lists.) The string contains context tokens, object tokens, method tokens, and argument tokens. It looks like regular CAL code trying to call every method in every object in every context. The first object in each context is the default object for that context. The first method in each object is the default method for that object. The tokens following a method are the default arguments for that method. The only new feature is the first token after a method is a *value7*-type number that represents an index into the method table. The interpreter uses this number to call the method.

When the interpreter is ready to call a method, it parses calContext-List looking for the proper context, then again looking for the proper object, then again looking for the proper method. One more parse and it has the index to call the method through the method table. It leaves a pointer, argPtr, there for later parsing of the default arguments. If the interpreter hits a context while looking for the object or it hits an object or context while looking for the method, the method is invalid.

The variable list contains *context-*, *object-*, and argument-type tokens. A *value7/8*-type token indicates the variable name, and the following token gives the variable's type and value. All variables must be created at compile time and given an initial value. The type or length can't change at runtime.

To dereference a variable, a routine parses calVarList looking for the proper context, then again looking for the proper object, then again looking for the proper name. When it finds it, the routine leaves the pointer pointing to the next token (the variable). If it hits a context while looking for the object or it hits an object or context while looking for the variable, the variable is invalid.

Listing I--The first step *in writing a CAL application is to set up the necessary types, structures, variables, and function prototypes.*

```
/************** TOKEN TYPES ********************/
typedef enum{
    object, method, nul, minimum, maximum, def,
    data, value8, context, udase, bitstring,
    delimiter, endoffile, escape, begin, end,
    value7, error, beginend,
} calTokenType;

/********** STRUCTURE DECLARATIONS *************/
typedef struct {
    calTokenType type:
    /* pointer to current pos in CAL message */
    char        *msg;
    /* length of message in message buffer */
    unsigned int len;
    /* len is value in return type struct */
#define val      len
} calPtrs;

/******** GLOBAL VARIABLE DECLARATIONS *********/
calPtrs          calPtr; /* main set of pointers */
calPtrs          calRet;/* returned value */
calPtrs          calContextList; /* the context list */
calPtrs          calVarList;  /* the variables list */
unsigned         calContext;  /* current context */
unsigned         calObject;   /* current object */

                          /* method table */
void                  (*(*calMethodTable)[])(void);
                          /* method table len */
unsigned     calMethodTableLen;

/******** GLOBAL FUNCTION DECLARATIONS *********/
calTokenType     calParse(int, int);
calTokenType     calEval(void);
calTokenType     calGetArg(int);
calTokenType     calDeref(unsigned short);
```

CORE INTERPRETER

The core interpreter, cal.c, contains only 181 lines of code. This file contains four externally callable routines: calParse parses a token, calEval executes CAL code, calGetArg gets an argument for a method, and calDeref looks up a variable.

The parser, calParse, is a small routine that switches based on the next code byte. The parser uses calPtr.msg as a pointer to the next code byte. As it reads bytes, it increments calPtr.msg and decrements calPtr.len. It puts the parsed token in calRet. The parser calls some internal routines and calls itself recursively.

The parser has two arguments, both flags. If the first argument, argOnly, is true, nonargument-type tokens will cause the parser to return an error. This aspect is useful when parsing arguments; an *error* token indicates the end of the list. If the second argument, dobegin, is true and the parser encounters a begin token, it will locate the matching end token, set calRet.type to *beginend*, set calRet.msg to the byte following the begin, and set calRet.len to the length of the block. The input pointer, calPtr.msg, is left pointing to the byte after the matching end token. If dobegin is false, the parser returns *begin* and *end* as individual tokens.

The interpreter, calEval, first looks at the type of token in calRet. Anything other than a *beginend* type evaluates to itself, so the interpreter returns with calRet untouched. A *beginend* type needs to be interpreted, so it pushes the old environment

Listing 2—A program that uses the CAL interpreter must have a method table array, a context list, and a variable list.

```
/* method table */
static void (*methodtable[])(void) = {
mFalse,    mTrue,  mTest,  mCompare, mAdd,
mAddImm,   mSub,   mSubImm,mLoad,  mLoadImmNum,
mLoadImmStr,mSwap, mAnd,mOr, mXor,
mNot,      mBranch,mBranchCond,mCall,       mReturn,
mExit,     mNop,   mIf,
};
/* context list with default arguments */
static unsigned char contextlist[] = {
CAL_CONTEXT,  1,
    O_PRIMARY,
        M_FALSE,          0, 'A',
        M_TRUE,           1, 'A',
        M_TEST,           2, 'A',
        M_COMPARE,        3, 'A', 'B',
        M_ADD,            4, 'B', 'A',
        M_ADD_IMM,        5, 1,  'A',
        M_SUB,            6, 'B', 'A',
        M_SUB_IMM,        7, 1,  'A',
        M_LOAD,           8, 'B', 'A',
        M_LOAD_IMM_NUM,  9,  0,  'A',
        M_LOAD_IMM_STR,  10,
            CAL_DATA,     1,  0,  'S',
        M_STORE,          8, 'A', 'B',
        M_SWAP,          11,'A', 'B',
        M_AND,           12,'B', 'A',
        M_OR,            13,'B', 'A',
        M_XOR,           14,'B', 'A',
        M_NOT,           15,'A',
        M_BRANCH,        16,'P',
        M_BRANCH_COND,   17,'=','A','B','P',
        M_JUMP,          16,'P',
        M_CALL,          18,'P',
        M_RETURN,        19,
        M_EXIT,          20,
        M_NOP,           21,
        M_IF_THEN_ELSE,  22, 1, 1, 0,

/* the variable list with initial values */
static unsigned char varlist[] = {
CAL_CONTEXT,  1,
    O_PRIMARY,
        'A',CAL_LITERAL,  0 ,
        'B',CAL_LITERAL,  0 ,
        'C',CAL_LITERAL,  0 ,
        'D',CAL_LITERAL,  0 ,
        'E',CAL_LITERAL,  0 ,
        'S',CAL_DATA,CAL_LITERAL,20,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
};
```

(calPtr) onto the stack, replaces it with calRet, and calls calExecute, an internal routine.

calExecute uses calParse to parse the CAL code, and the internal routine calMatch to look up methods in calContextList. It keeps calContext and calObject current, and it calls methods when needed. It also knows how to skip arguments after calling a method. A parse error, the end of the CAL program, or an intentional break ends execution.

The routine that fetches arguments for a method, calGetArg, has one argument, the number of the argument to get. It first looks for the argument in the CAL code that called the method. If the argument isn't there, it parses through the default arguments in the calContextList string. It sets calRet to the argument token or delimiter if it didn't find a regular or default argument.

The routine that looks up a variable, calDeref, has the reference string value as its argument. It calls calMatch to find the proper context and object in the calVarList. Then it examines every other token, looking for a match. It puts the variable token in calRet. Because the method that requested this variable may want to change it, calDeref makes sure that calRet.msg contains a pointer to the data portion of the variable.

METHODS

The file method.c contains the CAL methods. Macros at the beginning and two functions at the end provide routines for getting, checking, and comparing various kinds of arguments. I implemented most of the methods. However, I had to guess what the EIA intended for many of them because of the sparse documentation. The methods turned out to be pretty simple, but there are a lot of them. A manufacturer would include only the methods needed.

Remember that the "connection" between a method ID and a method occurs through the calContextList and calMethodTable. Having multiple versions of the same method for use in different contexts and objects is possible.

USING THE INTERPRETER

A program that uses the interpreter must have a method table array, a context list, and a variable list (see Listing 2).

To set up the interpreter, load calMethodTable with the address of the method table, load calMethodTableLen with the number of entries in the method table, load calContext with the address of the context list, and load calVarList. with the address of the variable list. To interpret CAL code, plug calRet with the address of the CAL code and call calEval.

The test program, test.c, has 14 separate CAL programs. It executes each one and verifies proper operation by looking at the appropriate instance variables. The test program also contains a little routine that can print CAL code in a human-readable format. The same routine can, of course, print the context and variable lists.

CONCLUSION

The tiny size of this interpreter amazes me. It has less than 5K of 80x86 object code, including all the methods. A lot of "implosion" occurred during development. I later rewrote the whole parser, complete with a half-dozen methods, in only four pages of 8051 assembly language! In the assembly language version, I was able to put the method address in the context list, eliminating the need for the method table array.

Over the years, I have embedded tiny languages in many projects, resulting in more versatile and easier to customize code. I've used David Betz's XLISP and a tinier LISP I wrote. While both are small, as languages go, they still involve thousands of lines of code and add a measurable overhead.

The CAL language, while not as expressive as LISP, is much smaller, faster, and flexible enough to handle most tasks. It's perfect for any kind of computer-to-computer communication. While I implemented it specifically for a home automation product I'm working on, I'm sure I'll be using it in many other projects.

Not one to leave well enough alone, I'm working on an assembler, disassembler, and structure editor for CAL. Wish me luck. ❑

Jeff Fisher is Chief Technical Officer at Ménage Automation Inc., a home automation manufacturer and retailer in San Jose, Calif. He can be reached on CompuServe as 71431.3343.

SOFTWARE

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

IRS

404 Very Useful
405 Moderately Useful
406 Not Useful

Temperature Monitoring

Add Some Temperature Sensors to the HCS II

When Steve encounters something that just doesn't sound right, he jumps right in to verify it for himself. He recently had a new HVAC system installed and had to find out just how well it really works.

CIARCIA'S CIRCUIT CELLAR

Steve Ciarcia

I thought of all kinds of cute titles I could have used to introduce this article, such as "Where There's Smoke, There's Fire! " or "Wood Burning: A Technical Dissertation." Authors often choose titles tongue in cheek, referring to their work with a sense of humor. Perhaps if I were writing for a literary journal rather than a computer reference magazine, I could still get away with it. Way back in the early days of ***BYTE*** I had a tendency to pick obscure titles. Would you believe that an article I titled, "Come Upstairs and Be Respectable" was a construction project for a serial communication link? Well, no one looking for serial project references can either. I apologize for those past indiscretions.

So rather than obscure the obvious by presenting a trite yet wonderfully eloquent title that means absolutely nothing, let me just say "Temperature Monitoring," as dull as that sounds.

A few years ago, many of you will remember, I added a glass-enclosed solarium to my house. At that time I described how I had specifically designed and insulated it for solar heating, but that I anticipated having to install an elaborate temperature monitoring and heat distribution system to fully utilize the benefits. As it turned out, passive solar heating in combination with a simple wood stove worked so well that it required no high-tech monitoring or control. A plain thermostatically controlled, through-the-wall vent fan managed temperature regulation completely.

If I stopped now, I would be leaving you with the impression that life in the solarium was a constant 72°F. Well yes, and no. As long as there was sun or the wood stove was on, it was a cheerful 72" during the winter. The difficult times were the spring and fall when the solar heating wasn't quite enough and full use of supplemental heating wasn't always required. Also, anyone who owns a wood stove knows that using it intermittently is both an aggravation and the primary cause of chimney fires.

This situation, and the realization that a frontiersman's life of chopping, lugging, and piling five chords of wood for the winter, loses its appeal quickly after the first "back to nature" year, suggested the immediate formulation of new plans. As soon as the winter was over, I had a 5-ton air conditioner and gas hot-air furnace installed. Because there was little alternative, they had to install it in the wood locker at the rear of the solarium. Ordinarily, using this location wouldn't matter to me, but the wood locker is unheated and is at ambient temperature. When it's 90°F outside, so is the air conditioner's evaporator. When it's -10°F out, so is the furnace.

Being a nosy engineer, I asked about the influence of ambient temperature. After all, don't people usually install this stuff in an inside closet or down in the basement (an impossible situation in my case)?

Without actually saying, "Go away boy, you're bothering me!" the HVAC guys said they were insulating all the ductwork outside and that I shouldn't worry. While most home installations were done a particular way, it was mostly for esthetics and cost. Commercial installations like mine "were built to take it."

All the time I was hoping that "take it" didn't mean to the cleaners, I couldn't help being bothered by the simple explanations these guys were handing out. Heating the metal parts of the furnace up from 0°F instead of 60°F (if located in a closet) had to cost some BTUs. Probably more important, lowering the temperature of the entire air conditioning system from 100°F to 70°F sounded grossly inefficient. Their "commercial installation" response was that a 5-ton (60,000-BTU) air conditioner could turn the solarium into a meat locker if I wanted it that way. I'd never even notice any ambient-related cooling losses.

Usually I'm not that skeptical, but this time I decided to check their opinion with some scientific investigation. While I'll have to wait until summer to collect actual measurements on the air conditioning, I decided to use this winter as the time to investigate my suspicions. I decided to monitor and record all the arguments in question and analyze the data to see who really was right.

Pertinent data collection in the solarium primarily involves monitoring light level and temperatures. In particular, inside solarium air temperature, outside air temperature, HVAC air inlet temperature, HVAC air outlet temperature, and ambient light level were most significant.

The wood stove became a greater consideration, however. With a wood stove that I might now use less regularly, I decided to add safety monitoring stack temperature and surface temperature probes. Stack temperature refers to the temperature of the exhaust gases and smoke going up the chimney. Surface temperature is the stove enclosure itself.

Thinking empirically as an engineer, you might theorize that a wood stove achieves optimum performance when it reaches maximum differential temperature. Unfortunately, a cherry red stove that barely melts ice cubes on the exhaust stack might be a better incendiary device than a space heater.

While it is true that the hotter the fire the more complete the combustion, any exhaust still contains some combustible materials. If the chimney (the metal portion inside the house is called the stack and the masonry or insulated metal portion outside is called the chimney) is cold, then this exhaust product condenses and collects on the inside. Cycling a stove frequently creates more of these occasions when the chimney is cool.

If enough of this combustible material collects in the chimney, it becomes an alternate burning source. If the fire in the stove becomes too hot, it can ignite the materials in the chimney and you have a condition called a chimney fire. The 2,000+°F heat of chimney fires has been known to penetrate cracks in the masonry, starting in-the-wall and roof fires.

I'm very careful and haven't had any problems thus far. Because I use about five chords of wood a year though, the potential for problems is greater than someone who lights a fire only on holidays. Properly operating the stove requires a relatively hot fire that keeps the stack temperature hot enough to reduce condensation of the exhaust. The stack temperature during typical operation is about 200–250°F. In a very hot fire it will go up to 350–400°F. On a couple occasions I'll call *run-aways*, when the stove got too hot, the stack temperature hit 500°F before I caught it. Of course any fire in the stack itself would increase that temperature substantially.

Surface temperature isn't a significant consideration. I'm only interested in it because the stove I have is lined with fire brick to hold the heat longer. I just thought it would be interesting to see how long it would do so and whether this heat contributes significantly to the stored solar heat.

Finally, these seven measurements should be as incorruptible as possible because they are important analog signals used in a control process. As a check on the monitoring unit itself, I added an eighth analog reference signal. Except for individual probe errors, any system ADC error will show up equally on all channels. Therefore, testing the accuracy of one channel checks all of them.

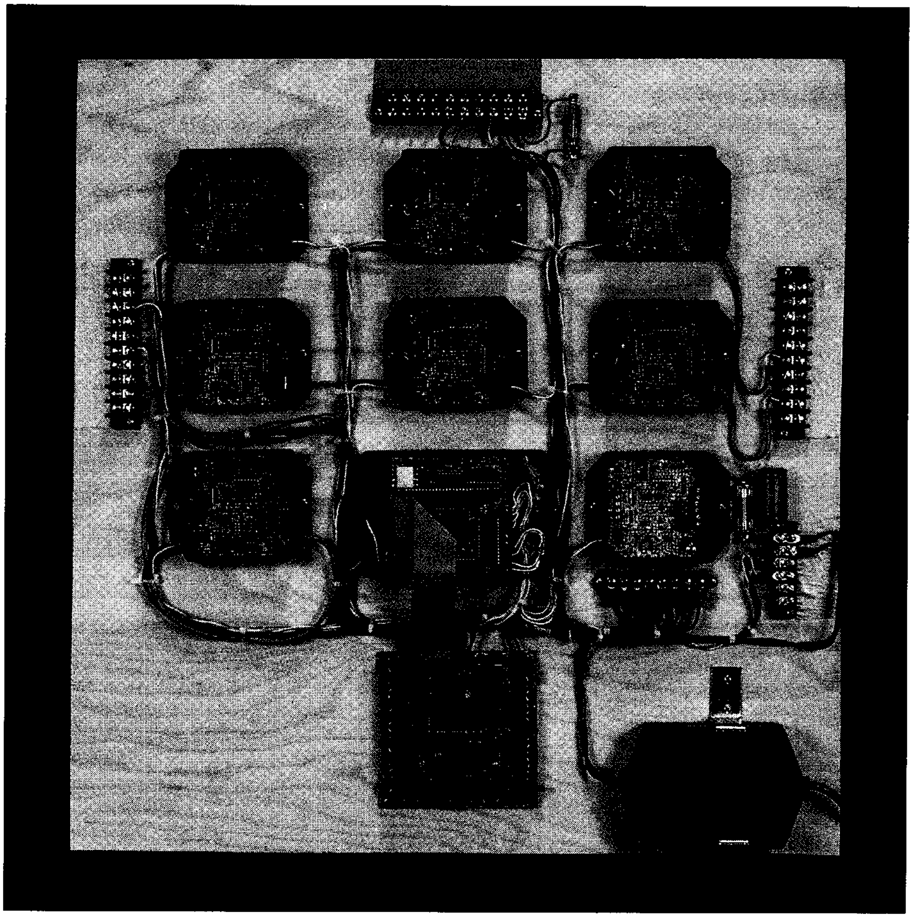


Photo 1—*Keeping track of eight signal conditioners and an ADIO-Link could become a rat's nest real fast. Mounting everything on a single piece of wood keeps things neat and makes installation easy.*

CIRCUIT CELLAR HCS

Ordinarily, building an eight-channel data logger for this one specific application would be no insignificant task. Fortunately, because I already have the Circuit Cellar HCS controlling the rest of the house, I only have to add some

	Analog Input	Range
CHAN0	Solarium inside air temp	0 to 100°F
CHAN1	Outside air temp	-20 to 105°F
CHAN2	HVAC air inlet temp	0 to 100°F
CHAN3	HVAC air outlet temp	0 to 300°F
CHAN4	Wood stove stack temp	0 to 500°C
CHAN5	Wood stove surface temp	0 to 500°C
CHAN6	Ambient light level	+0.5 V moderate light
CHAN7	Half range voltage reference	+2.500 V

Table l--The eight analog channels of the *ADIO board allow plenty of points to be measured to gef a good feel for temperatures in and around the solarium.*

network-connected hardware and a few program instructions to provide all the monitoring and logging required.

We have covered the architecture and hardware of the HCS previously and in considerable detail, so I will not do that here. I will only reiterate that the HCS has the capability to read analog as well as digital inputs.

Unfortunately, the nature of analog signals is that they cannot be transmitted over long distances as digital signals can. Without special shielding or conditioning schemes, the typical analog signal compatible with the HCS shouldn't use wires longer than a dozen feet or so.

The solarium heating system is about 75 feet from the HCS Supervisory Controller *(SC)* as the crow flies (as the mouse scurries?), but the wiring distance is more appropriately about 125–150 feet because it can't be run as a straight-line course. Therefore, I rejected direct analog wiring of temperature sensors to the SC.

Fortunately, we recognized that not all signals were going to be immediately local to the SC when we designed the HCS. To accommodate remote digital and analog I/O, the HCS uses an RS-485 serial network to talk to remotely located I/O-Links. These Links have specific functions such as LCD display, infrared remote control, badge IDs, analog I/O, digital I/O, and so forth. Up to 31 of these links can be connected to the SC.

The ADIO-Link is particularly suited to this data-logging task. It has eight 8-bit analog inputs (O-5 volts), four 8-bit analog outputs (optional, O-5 volts), and 24 digital I/O bits divided as 16 inputs and 8 outputs.

THERE ARE NO EASY TEMPERATURE MEASUREMENTS

Before I elaborate on the ADIO connection to the HCS, I need to discuss getting temperatures converted into levels suitable for 0-5-volt computer systems. A great many people imagine that you simply take a two-wire probe and screw it on the ADC input or at the most you use an LM34 direct-voltage-output integrated sensor and everything is solved. As a matter of practicality I'd say possibly, but my experience suggests that only the *other* guy gets to do it simply!

As I start describing the process of measuring a few inconsequential temperatures, take a look at Photo 1, which shows the equipment board that I had to mount next to the HVAC unit in the solarium. Granted, this whole mess connects to the SC via a simple twisted-pair RS-485, but life rarely operates in a range of O-5 volts. The purpose of all that junk is to make it look as though it does.

The complication is the measurement technique. When you convert analog signals into digital representations you quantize them into an approximation. The nearness of this approximation to the actual value depends on the number of digital bits used. For example, with an 8-bit converter representing a range of 0–1000 volts, a 69-volt input cannot be exactly recorded. With 256 incremental steps, each bit changes by 4 volts. Therefore, the ADC would read the 69-volt input as either 68 volts or 72 volts only. Increasing to 9 bits makes the approximation closer, but only a 10-bit ADC allows the reading of 69 out of 1000 volts directly.

While increasing the number bits may seem the only way to make accurate analog measurements, doing so is half the story. You can increase the number of bits, or you can reduce the range as an alternative. In the last example, an 8-bit ADC was 4 steps per increment when the range was O-1000 volts. The same ADC will provide 1 step per increment, directly indicating an input of 69, if the range is reduced to O-250 volts instead.

The most important consideration in analog signal measurement is balancing range and resolution. While 12-bit resolution on all measurements would be wonderful (and considerably more expensive), that is not the case. An 8-bit ADC is significantly more accurate if you scale the analog signals so the measured signal band most appropriately fits the 0-5-volts ADC span. The hardware used to multiply, offset, or otherwise modify these raw analog levels into 0- to 5-volt "computer-compatible" levels are called *analog signal conditioners.* They account for six out of the nine boxes on the equipment board. More later.

A VARIETY OF REQUIREMENTS

Rather than beat this theory into the ground, let me go back to the system I'm connecting to the ADIO board and analyze what signal conditioning is required. Table 1 shows the eight analog inputs I have designated and the temperature/voltage ranges I'd like to measure.

The next obvious question is how do I measure these temperatures. Do I use thermocouples, RTDs, thermistors, or what? My answer is yes.

In reality, given the ranges I've listed, there are quite a few options. You might even choose to use a different technique entirely. While I'll substantiate my choice, let me list the five most common measurement methods and discuss them:

1. Thermocouples

*There are six common thermocouple types. The two wires making up the thermocouple are each made of different metal alloys that, when twisted or bonded together, generate a voltage proportional to the applied temperature.

*Their operating range is -270°C to + 1800°C. Sensitivity at 25°C is typically less than 50 µV/°C, and accuracy is generally ±0.5°C with a good reference.

*Linearity is poor over wide ranges but reasonable if piecewise linearized

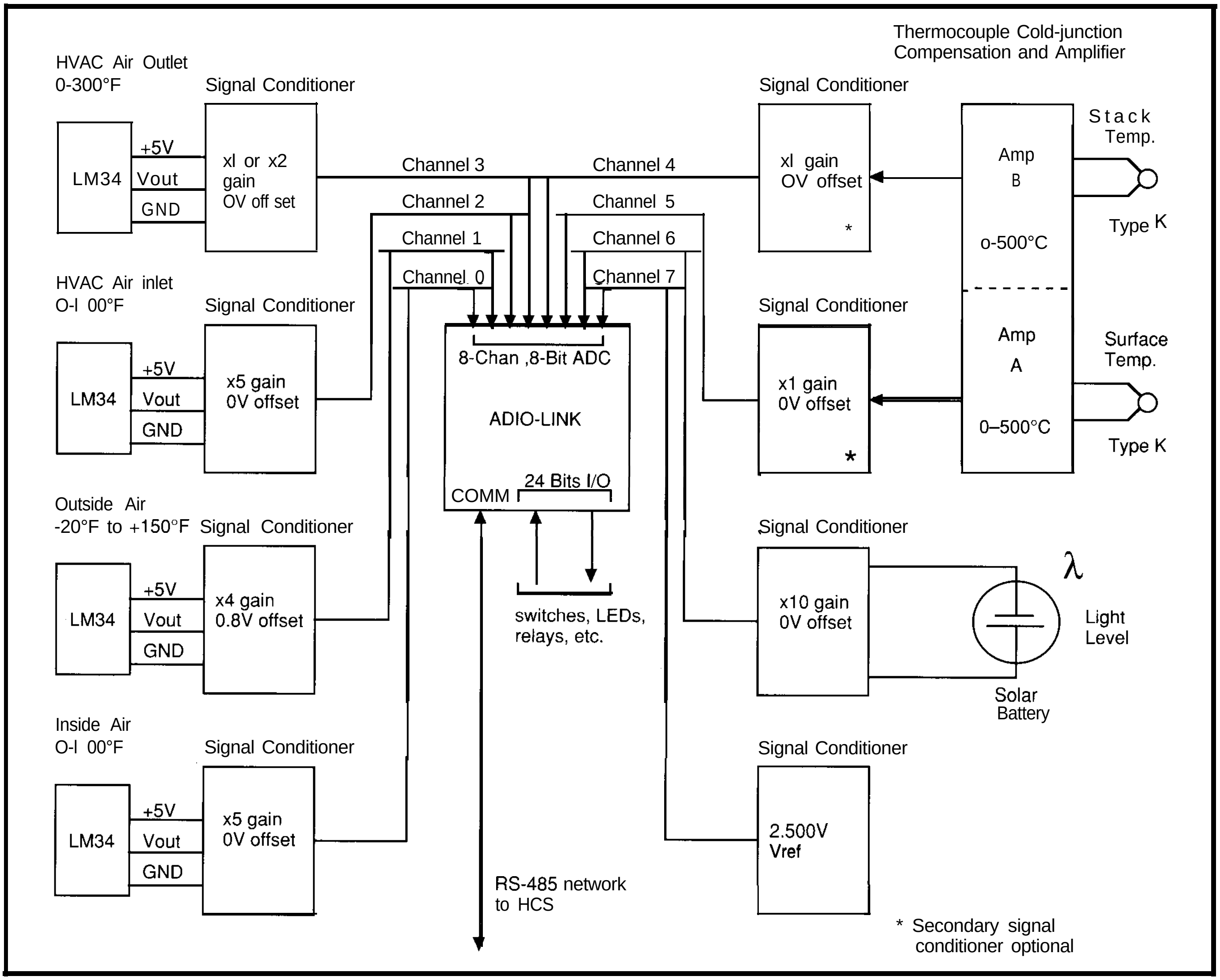


Figure 1—*Signal conditioners are used on all eight channels to allow individual gain and offset values for each channel. Each sensor is adapted to make maximum use of the ADC's input scale.*

Material	Sensitivity (µV/°C at 25°C)	Range (“C)	Swing Over-Range (mV)	Letter Designation
Copper-Constantan	40.6	-270 to + 600	25.0	T
Iron-Constantan	51.7	-270 to +1000	60.0	J
Chromel-Alumel	40.6	-270 to +1300	55.0	K
Chromel-Constantan	60.9	-270 to +1000	75.0	E
Platinum 10%-Rd/Pt	6.0	0 to +1550	16.0	S
Platinum 13%-Rd/Pt	6.0	0 to +1600	19.0	R

Table **2**—*There* are six thermocouple types, each *with its* own *characteristics.* Most *designers avoid thermocouples because using them has usually involved considerable circuitry.*

in 100” sections. Cost is $1 to $50 depending upon type and packaging.

*Using thermocouples requires an actual or mathematically derived correction referenced to 0°C. Also, they require extremely stable signal-conditioning components because of the low signal levels involved.

2. **Thermistors**

*Thermistors come in a variety of forms and materials. Basically they are temperature-sensitive resistors. As temperature rises, the resistance falls.

*Thermistors have an operating range of about -100°C to +450°C. Sensitivity at 25°C is generally 5%/°C for “regular units” while linearized composite units generally are 0.5%/°C. Accuracy is ±0.1 °C from -40°C to +100°C but can be as good as ±0.01°C at 0-60°C.

*Linearity is typically ±0.2°C for linearized composite units. Single thermistors are quite nonlinear. Cost is $2 to $10 for standard units.

*Thermistors have the highest temperature sensitivity of any common sensor, but special considerations are required for long-term use above 100°C. Thermistors require moderately precise signal conditioning and are susceptible to self-heating errors in the measurement process.

3. **Platinum Resistance Wire**

● Frequently called Platinum RTD (Resistance Temperature Detector), it functions like a thermistor except that when the temperature increases, the resistance of the RTD increases.

•Platinum RTDs have an operating range of -250°C to +900°C. Their sensitivity is approximately +0.5%/°C at 25°C, and accuracy is typically ±0.1°C. Special laboratory-grade sensors can achieve ±0.01°C accuracy.

*Platinum RTDs are nearly linear over wide temperature ranges, but they are very expensive compared to other sensors. Cost is typically $25 to $1000 depending on specs. Most industrial units are about $100.

*While Platinum RTDs have higher cost and lower sensitivity, they have wider ranges and better long-term stability than thermistors. They require moderately precise signal conditioning and are susceptible to self-heating errors.

4. **Diodes and Transistors**

•Since the first transistors and diodes were invented, users have faced an annoying tendency for the voltage drop across junctions to vary with temperature. Properly conditioned, this aberration makes an accurate temperature indicator.

*Junction sensors have a typical operating range of -270°C to +175°C. Their sensitivity is about 2.2 mV/°C (about 0.33 %/°C) while their accuracy is limited. In a range of -55°C to +125°C, the accuracy is ±2°C to ±5°C.

*Junction sensors are fairly linear. The average is 2°C over the operating range. The best feature about these sensors is their low cost, typically less than 50¢ each.

*While they are inexpensive, junction sensors must be individually calibrated and driven from a precision current source for best performance.

5. **Integrated Circuit Sensors**

● IC Sensors are the next circuit extension to transistor and diode sensors. With the signal-conditioning circuitry built in and the calibration done during wafer-level manufacturing, the result is a precise, completely packaged temperature indicator.

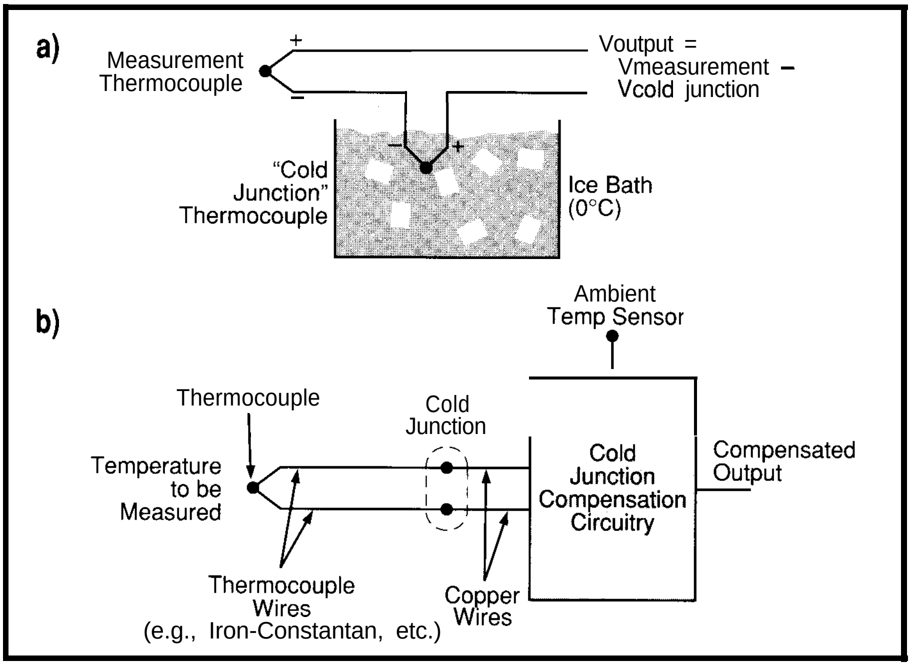


Figure 2—*(a) An ice bath used to be used to bring the thermocouple junction points to a known temperature to compensate for the voltage generated by the dissimilar metals in that junction. (b) Now, co/d junction compensation circuitry does the same without the mess.*

• The additional circuitry makes an IC sensor physically like other ICs, so its operating range is similarly restricted. IC sensors operate from about -85°C to +125°C. Within this range their sensitivity is typically 0.2–0.4%/°C and accuracy is less than half a degree. Cost is typically $1 to $10.

• IC sensors are quite linear because they contain the necessary circuitry and are precalibrated. Across the entire range linearity is usually within 1°C, but within a range of 0–70°C linearity can approach 0.2°C.

• IC sensors are moving into many applications usually reserved for thermistors because of their low cost and freedom from precision signal conditioning. Unfortunately, they do have a limited temperature range.

One final note about signal-conditioning circuitry. As a general term, signal conditioning usually means the entire circuitry from the sensor to the final ADC input-compatible level. In more specific terms, the signal-conditioning process includes separate stages that may or may not be required for all applications. For example, when using an RTD or thermistor, one conditioning stage converts resistance into voltage. Another stage level shifts and amplifies this signal to the desired levels.

Some sensors, such as the LM34 IC, which incorporates full signal-conditioning circuitry already, may only require minimal external level shifting or amplification to obtain a desired output. The process of signal modification, however minimal, is still called signal conditioning.

CHOOSING SENSORS

If you compare the list of six temperature ranges that I want to monitor in the solarium with the five types of sensors, you'll note a considerable overlap. Except where a specific accuracy or range limitation is the ultimate designator, choice of a particular temperature sensor is often just that: personal choice.

The list of six temperatures has two categories: high temperature and low temperature. The four low temperatures to be sensed have a cumulative range of -20°F to 300°F. The two high-temperature sensors have to cover 0-500°C (32-932°F).

From my list of sensor types it appears that IC sensors like the LM34 are the cost-effective, low temperature-sensing choice while thermocouples are best for high-temperature sensing.

Of course, connecting these probes to the ADIO involves much more than simply connecting them to the screw terminals-even the preconditioned IC sensors can require additional signal conditioning. Figure 1 is an overall view of the signal connections to the ADIO, and also diagrams the configuration and conditioning steps involved. Let me start by discussing the thermocouples.

THERMOCOUPLES

Table 2 shows the six thermocouple types. Most designers avoid thermocouples because using them has usually involved considerable circuitry. The physical characteristics that produce temperature-sensitive outputs from thermocouples must also be compensated for elsewhere within the signal-conditioning circuitry.

When two dissimilar metals are joined, a voltage differential proportional to the temperature is produced at that junction. If you know the types of metals, such as copper or Chromel, then you know the voltage generated and therefore the temperature at that junction. Unfortunately, thermocouples have more than one measurable junction and all junctions are equal.

The junction of a Chromel-Alumel thermocouple securely affixed to the wood stove's stack will indeed generate a voltage proportional to the exhaust gas temperature. Unfortunately, the Chromel-copper and Alumel-copper junctions will also produce voltages where they enter the signal amplifier. The net result is a probe with one junction where the measurement is taken and two junctions floating around at ambient temperature, creating a subtractive error.

Traditionally, the subtractive error was eliminated by physical means as shown in Figure 2. By holding the thermocouple connection points at a fixed temperature, in this case 0°C, then any generated voltage was attributable only to the measurement junction. Because this connection point was generally kept lower in temperature than the measurement thermocouple with crushed ice, it was called the *cold junction*. The process that corrected the connection point temperature to 0°C was called *cold junction compensation*.

This last line is significant. While for many years cold junction compensation truly involved the use of ice, modern day sensors have eliminated that practice. Just as you know the exact voltages produced by a Chromel-Alumel junction, you know how much error Chromel-copper and Alumel-copper junctions at a particular temperature produce. Simply measure the cold junction temperature, and correct it either mathematically with the same computer collecting the data or as an analog feedback signal in the signal conditioner.

Figure 3 is the schematic of a signal-conditioning circuit that both amplifies the thermocouple output as well as provides cold junction compensation. Using a type-K Chromel-Alumel thermocouple, the circuit is designed to produce an output of O-5 volts corresponding to 0–500°C.

The operation of the circuit is fairly straightforward, but it uses some sophisticated devices. The LT1025 is a monolithic cold junction compensator IC. The device measures ambient temperature and puts out a correction signal scaled for the particular thermocouple. While I have chosen to use a type-K device, you could easily change to a type-J by switching to the J pin on the LT1025.

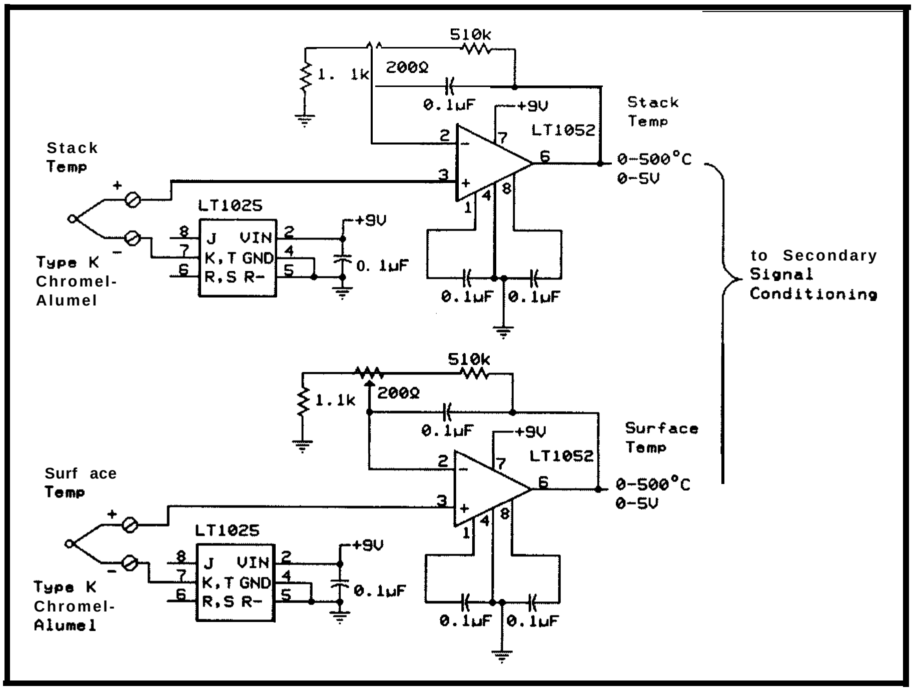


Figure 3—*The LT1025 cold-junction compensator adds a correction signal based on ambient temperature and known characteristics of the thermocouple. The LT1052 scales the reading to a useful range.*

The amplifier is not your plain vanilla op-amp. Considering that at 500°F your thermocouple is only generating 10.57 mV, you'll understand that thermocouple amplifiers need very low offset voltage, low drift, and low bias currents. The LT1052 is a special chopper-stabilized amplifier, which has a 5-µV offset, 0.05-µV/°C drift, and 30-pA bias current.

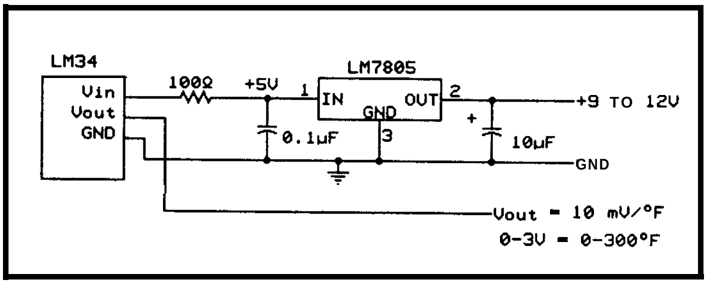


Figure 4—*Given* **a simple** *5-V power supply,* **the** *LM34 puts* **out a** *voltage* **that is** *directly proportional to its ambient temperature.*

The amplifier gain primarily determines the range of the thermocouple. At 500°C a Chromel-Alumel thermocouple has an output of 20.65 mV. If the signal-conditioner output is to be 5 volts at 500°C then the amplifier must have a gain of approximately 242. Similarly, you can instantly change your thermocouples to be 0–500°F by raising the gain to 473 (10.57 mV at 500°F). I'll decide which range makes better sense after I've used the system for a while.

This circuit does not directly correct offset or linearity errors, although they should be minimal considering the components used. Rather than further complicate the signal conditioning, I decided to take a dual approach. In the next section on using the IC sensors, I will describe a signal-conditioning circuit used to offset or amplify signals. One option of the circuit is to have a gain of 1 and an offset of 0, which functionally does nothing to signals passing through it.

The thermocouple signal conditioner I've described may need some external tweaking of offset or gain to make it more precise. Traditionally, all this linearization, offset adjustment, and tweaking circuitry has been included in the signal conditioner because the final conditioned signal was used to drive a direct indicator, such as a meter or recorder. For example, a 0-1-volt meter could not deal with a 0.5-volt offset. The traditional approach was to correct physically or otherwise modify the analog signal directly.

Today many sensors are connected to computers via ADC inputs. Once inside the computer, you can modify and manipulate these readings just as if they were in hardware. Rather than have an LT1025 cold junction chip, I could have used another ADC channel to measure ambient temperature (which we are, by the way) and mathematically correct the thermocouple reading at a rate of 40.6 µV/°C from the ambient temp to 0°C. Similarly, I could physically calibrate a probe by taking periodic readings at known temperatures and generate a correction table.

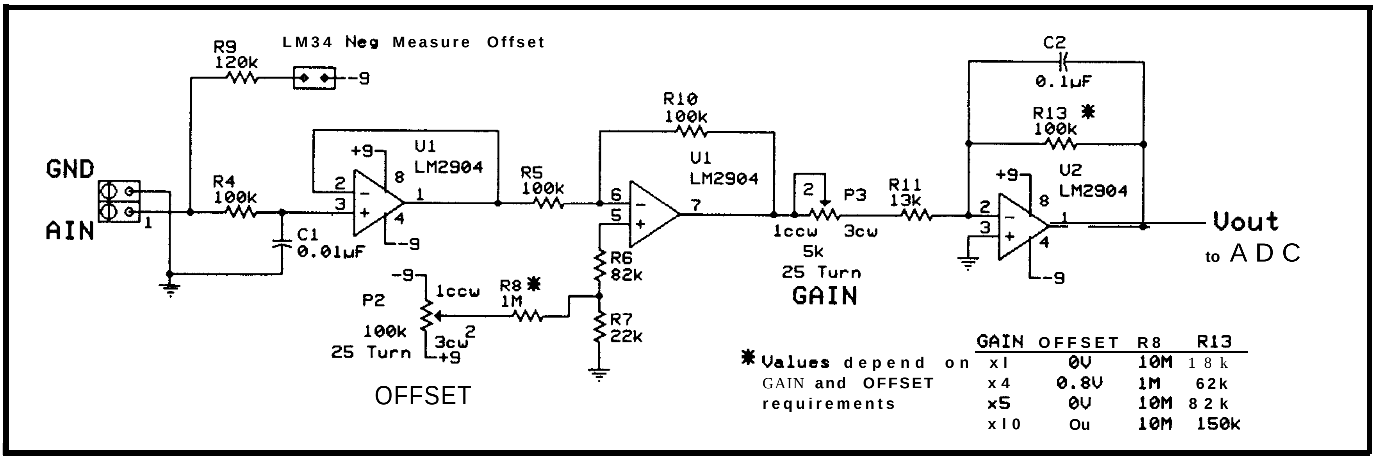


Figure 5—*The same signal conditioning circuit can be used for all the channels with some minor tweaking. The gain (first stage) and offset (second stage) are set by the values of two resistors and some trim pots.*

How much you want to do is open. I designed the thermocouple signal conditioner to provide basic information. In my system, its 0–5-volt output goes to a second conditioner (gain = 1 and offset = 0) and then to the HCS. Connected this way, I can choose to either tweak the signals with correction tables or adjust the potentiometer. Ordinarily, I'd prefer to keep it in the computer (easier than getting on a ladder in a snow storm to adjust the signal conditioner out in the wood locker), but I don't know the effect that long correction tables will have on the HCS's execution speed. This way I have both options available.

USING THE LM34 IS A MATTER OF SCALING

I can monitor the other four temperatures in a variety of ways, including using thermistors and RTDs. For the same personal preferences that cause some people to avoid thermocouples, I choose to avoid these in favor of using the LM34 IC.

The LM34A has a usable range of -50°F to +300°F (a companion device designated LM35 has a range of -55°C to +150°C). Illustrated in Figure 4, the primary benefit of the LM34 is that its output reading is a direct indication of temperature. At 105°F, the LM35 will indicate 1.05 volts. At 242°F it will show 2.42 volts. Similarly, at -18°F its reading will be -0.18 volts.

The only concern about using the LM34 or similar devices with a computer is resolution. It hardly makes sense to select a device for its 0.2°F accuracy and then take readings with a resolution of ±2°F. Of course if you only need ±2° accuracy, then you should acknowledge that up front, too.

The major mitigating factor in resolution is the ADC. The HCS uses an S-bit converter with a resolution of ±0.39%. As I mentioned earlier, that is only half the story.

One of the temperatures I want to monitor is the heated air out of the gas furnace. My guess is that it will be about 150–180°F, but I've designated a probe with 300°F capability. With the 0-5-volt input 8-bit ADC on the HCS, any reading from this sensor will have a resolution of ±1.9°. Considering the application, I don't consider this measurement too imprecise.

You can improve it by further signal conditioning, however. Figure 5 is the basic amplification and offsetadjusting circuit that I use for most sensor applications. The first stage is a high-impedance buffer; the second, an offset adjustment with xl gain; and third, a variable-gain stage. The starred components change depending upon the desired gain and offset required. In the case of the thermocouples mentioned earlier, I set this circuit for xl gain and O-volt offset.

If you understand that the output from the signal conditioner is not going to a meter or gauge but to a computer instead, then directly comprehendible readings are not required. While a 0-500°C sensor with 0–5-volt output sounds wonderful because you can easily check readings, a 0-250°C sensor with 0–5-volt output will provide twice the resolution for readings within its range. By using the circuit shown in Figure 5 to multiply the output of the LM34 by 2 (x2 gain with O-volt offset), you change the 0–300°F/0–5-volt sensor into a 0–250°F/0-5-volt sensor. With the same S-bit ADC, you can now get ±1° resolution.

The inside air and HVAC inlet air sensors are similarly amplified; this time by a factor of five. At 100°F the LM34 will read 1 .OO volt. At the output of the signal conditioner, the amplified signal will be 5.00 volts. Taking an S-bit reading of the O-100°F sensor will then in effect make it better than IO-bit resolution. The resolution of your O-100°F sensor will be approximately ±0.4°F. As far as the computer is concerned, the actual temperature is just the ADC count times 0.4” per count.

The outside air sensor has a wider range but is similarity scaled. Because Connecticut doesn’t really have vicious climate extremes, I felt that a sensor that covered from -20°F to +105°F would suffice. A range of -20 to +105 on the LM34 (-0.20 volts to +1.05 volts) is a full span of 1.25 volts. Using the basic signal-conditioner circuit again, multiply 1.25 volts by 4 to produce a higher resolution 5-volt output. However, to allow the ADC to read negative voltages, offset this reading by 0.8 volts. The resolution of your modified sensor becomes ±0.5°.

As far as the computer is concerned, it’s just numbers. It treats anything below 0.8 volts as a negative temperature with a reading of minus [(ADC count – 41) x 0.5]° and anything above 0.8 volts as positive temperatures multiplied by the same scaling factor.

The final light-level sensor is a variation on a theme. Instead of a temperature sensor, it used a solar battery. Such devices generally produce about 0.5 volts in moderate sunlight. To improve the resolution of this sensor significantly and allow better light-to-dark gradation level sensing, I used a ×10 amplifier.

IN CONCLUSION

You might have thought that wiring the signal-conditioner board would be the most demanding task, but I found doing so merely time consuming. Attaching the sensors and mounting the board in the wood locker in the middle of a Connecticut December confirmed my addiction to home control.

It will be a while before all the data is in, and no doubt Ken and I will have to have some discussions about having certain HCS math functions do some of the things I want to do. However, don’t doubt for a minute that I won’t stuff in some black box to provide the solution if I can’t wait.

While creating true closed-loop home control can seem a horrendous task to some, doing it a bit at a time not only chips away at the ultimate goal, but it provides sustained, albeit incremental, positive feedback.

Personally, I wouldn’t have it any other way. I like the enlightened living brought by electronics and home control. The only thing better than planning and designing it is actually using it. ❑

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

IRS

407 Very Useful
408 Moderately Useful
409 Not Useful

Putting the Wraps on CEBus

FEATURE ARTICLE

Ken Davidson

We've finally come to the end of the process of putting the CEBus specification together. Now that engineers can begin designing real products, what is being done currently and what can we expect in the future?

CEBus, CEBus, CEBus. You've seen me writing about it and its development for years now. "So where is it?" you might ask. I'm happy to report that we're finally reaching that seemingly unattainable conclusion to the series of committee meetings required to hammer out every last detail of the specification. We're finally seeing the dawning of actual product development. If you don't know what I'm talking about, let me give you some background.

A BIT OF HISTORY

CEBus is a home automation communications standard sponsored by the Consumer Electronics Group of the Electronic Industries Association (EIA/CEG) and written by a committee made up of representatives from most of the major and even more of the not-so-major companies involved with consumer electronics, telephones, computers, and alarm systems. Some names of those involved (both directly and indirectly) that spring immediately to mind include Panasonic, Sony, Leviton, Universal Electronics, Johnson Controls, AT&T, Apple, and Ademco.

CEBus, which originally stood for the Consumer Electronics Bus, started around eight years ago in an effort to combat the ever multiplying pile of hand-held IR remotes on the living room coffee table. The committee sought to develop a single communications standard that could control all manner of consumer electronics devices. However, as the committee started work on the standard, they quickly incorporated the entire house and several more communications media. While the name CEBus has stuck, it has grown to encompass a lot more than consumer electronics equipment.

The standard defines communications specifications for power line, twisted pair, coax, infrared, and RF, with fiber-optics to be added in the future. Based on the ISO/OSI seven-layer network model, the upper network layers are identical for all the media, while the lowest (physical) layer is unique to each medium. By using routers to cross-connect different

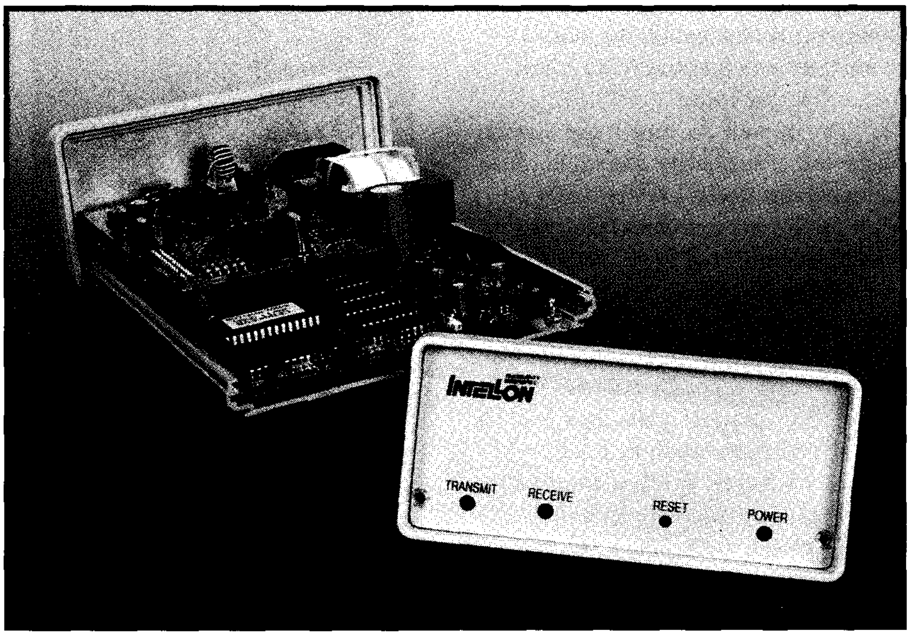


Photo I--The *Intellon CEBus* power line modem uses a special Motorola chip to handle the physical details.

```
111000111000111000110001110001110001110001100011100011100111
000111000110001110011100011000111001110001100011100111001110
001100011000110001100011000110001100011000110001100111001110
011000110001100111001100011001110011000110011100110011100110
011100110011100110011000110011001110011001100111001100110011
001110011001100110011001100111001100110011001100110011001100
```

Figure I--The *primary substate consists of a* **carrier** *spreading sequence that is made up of* **360** *"chips." The sequence* **is** *reversed to create a second substate.*

media in the house, having any CEBus-compatible device in the house communicate with any other CEBus device transparently regardless of the media connecting them is theoretically possible.

CEBus does not attempt to define how to achieve control in the house. It simply defines the medium by which devices communicate. Having a common link connecting all devices in the house makes the overall control job much easier. If you've been following the HCS II articles we've been presenting over the past year, then you'll realize that CEBus won't replace the HCS II; it will simply supplement and enhance it. By defining a "CEBus-Link," the HCS II Supervisory Controller will be able to communicate with and control any other CEBus-compatible device in the house.

CEBus is intended primarily for control, so it defines a control channel that operates at a rate of 10,000 one-bits per second for each medium. The length of a zero bit is twice that of a one bit, so the effective data throughput is actually somewhat lower than the above figure, but it is still more than adequate for typical home control purposes.

In addition to the control channel, the coax and twisted pair definitions include data channels, so you can send several independent channels of voice, music, and video over the same cables. Data channels for the other media are left as future enhancements.

A question I'm often asked is, "What cable should I run to be CEBus compatible?" You obviously don't need extra cable for power line, IR, or RF. For twisted pair, you should run standard four-pair (eight wires, 24 AWG) twisted-pair cable from a central point in your house to every room in the house (multiple runs to large or heavily used rooms). For coax, two cables should be run from the same central location to every room in the house or wherever you expect to have a TV or camera. Be sure to use double-shielded RG-6 cable and not wimpy RG-59. All the media use off-the-shelf connectors, so you're not locked into using some expensive proprietary connection system.

For more details about the CEBus network layers and how the physical media work, consult the "CEBus Bibliography" for a list of my past articles. Also be sure to check out the CAL Interpreter article elsewhere in this issue for a look at the CEBus application layer.

THE BEEF

Where am I leading with this? For a long time now I've been saying CEBus is right around the corner. Well, CEBus is finally here-at least the final specification is, anyway. About two years ago, the committee released the upper network layers and started releasing the physical layers as they were finalized. The last of the physical layers (RF) has been finished, and you can now purchase the complete specification. CEBus is now in such a form that companies can start designing it into their products without fear that someone is going to come along and make major changes. Dubbed IS-60, it is available from Global Engineering (see the "Sources" section).

RF FINALE

I've described in detail each of the other physical layers at one time or another in the past, so it's only fair I round out the media by going over the newly released RF specification. Similar to the power line standard, RF is based on *spread spectrum* technology from Intellon Corporation. While the power line used a *chirp,* or a modified frequency sweep, to encode data, the RF specification defines a somewhat complex scheme for spreading around the transmitted energy. For me to start at the highest level (within the physical layer) and

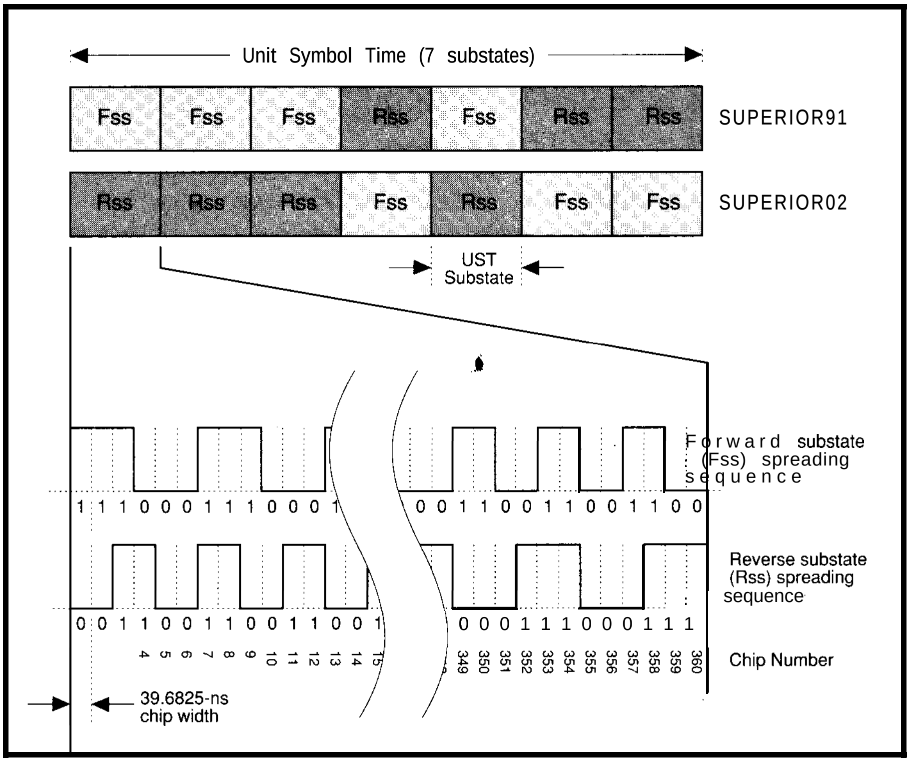


Figure 2—*The superior01 state is made up of seven substates and lasts one unit symbol time. The superior02 state is identical but has the* **order** *of the substates* **reversed.**

work my way down to the actual modulation scheme might be the easiest.

Rather than represent bits of data as distinct states (e.g., a 1 bit is five volts and a 0 bit is 0 volts), CEBus alternates between various states, with individual bits represented by the duration of those states (pulse width encoding). All packets are made up of four symbols: one, zero, end of frame (EOF), and end of packet (EOP). A one symbol (1 bit) lasts for a single Unit Symbol Time (UST), which is 100 µs. A zero symbol (0 bit) lasts two USTs; an EOF lasts three USTs; and an EOP lasts four USTs.

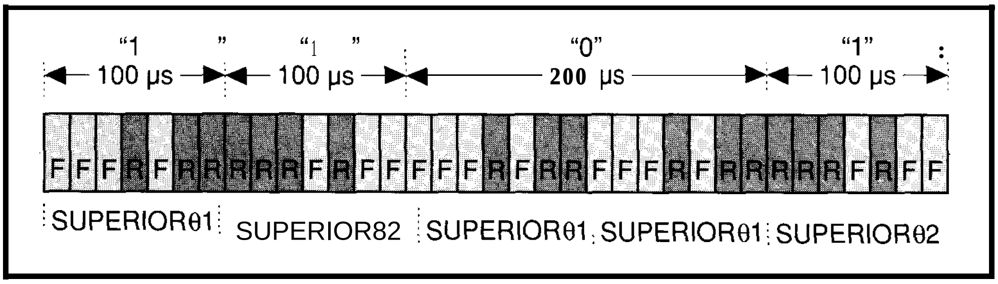


Figure 3—The data **portion** of **a** typical data packet is transmitted by alternating between two different states. Symbols are encoded by the duration of each state.

Each CEBus packet starts with a preamble as part of the CSMA (Carrier Sense, Multiple Access) protocol. During the preamble, which is simply a random number, the transmitter listens to the medium while it's transmitting to determine whether others are also transmitting. If so, it backs off and tries again a random amount of time later. Symbols sent during the preamble are encoded by alternating between a superior state and an inferior state (which is actually silence so the transmitter can listen for others). After the preamble, the transmitter knows it has control of the channel, so it alternates between two different superior states, known as ***superiorql*** and *superiorθ2*.

The superior states are made up of seven substates, each consisting of a specific carrier spreading sequence of 360 "chips" (not to be confused with the silicon kind). Figure 1 shows the sequence of chips that makes up a single substate (known as Fss, or forward substate). A second substate is also defined that is made up of the same sequence reversed (known as Rss, or reverse substate). A single unit symbol time of ***superiorql*** is made up of three Fss, one Rss, one Fss, and two Rss. ***Superior02*** is the complement, made up of three Rss, one Fss, one Rss, and two Fss (see Figure 2).

Finally, a third function, known as the Phase Modulation Function (PMF), modifies the substate spreading sequence to modulate the phase of the substate sequence randomly. The PMF is made up of the 15-bit sequence: 101011001000111. One bit of the PMF is XORed with each chip in a substate spreading sequence to cause all the bits in the sequence to be either inverted or unchanged. The next bit in the PMF modulates the next substate sequence. When the last bit of the PMF is reached, the sequence is repeated.

At the core is the 915-MHz carrier frequency, which is phase key shifted to generate two sidebands centered on the carrier frequency 10.5-MHz apart (center to center) with a nominal bandwidth of 2.1 MHz in each sideband.

Figure 3 shows how the data portion of an RF packet is transmitted. The modulation always alternates between the ***superiorql*** and ***superior02*** states, with the length of each state representing the symbol (as I mentioned above). When the symbol is more than one UST long, the same state is repeated multiple times.

The preamble is a little different. In order to differentiate it from the data portion, two modifications are made. First, only ***superior01*** is used (alternated with the inferior state) and has 2-µs gaps inserted between substates. As a result, one UST ends up lasting 114 µs instead of 100 µs. Second, the preamble EOP lasts 800 µs, as opposed to 400 µs during the data portion.

I've only scratched the surface here, and may have glossed over a few points too quickly for some. Be sure to get a copy of the real specification to get all the details.

TRAINING

One of the first requirements when introducing anything new is basic user training. The standard won't go anywhere without both manufacturer and consumer awareness. EIA and the CEBus committee have had a "publicity" subcommittee for years and have taken strides to increase that awareness. They had a CEBus booth at the Consumer Electronics Show for several years, where thousands of manufacturers and dealers got a first-hand look at the upcoming technology.

CEBus technology has also been to the NAHB Builders Show to give those who will be installing it in new construction a feel.

The Bright Home in Indianapolis gave consumers their first exposure to the system and wood builders and power company officials alike.

Grayson Evans, a longtime CEBus committee member and owner of The Training Department, has started running seminars and classes to give primarily manufacturers a good introduction to the specification and to show them how it can benefit their products. He reports that response to the seminars has been phenomenal, and if that response is reflected in new product designs, we should start seeing some exciting new CEBus-compatible products in the not-too-distant future. Seminars are run on a regular basis in major cities around the country and can be set up as needed for in-house training, so if your company is interested in learning more, give Grayson a call.

Tricia Parks of Parks Associates has also been busy putting together publicity events and events to pull together key players of the blossoming home automation industry. For the last few years, Parks has sponsored "Forum" gatherings that feature speakers, panel discussions, and debates revolving around home automation trends and issues. The most recent was Forum '92 that took place In September 1992 in Dallas. Forum '93 is coming up early this fall.

Another upcoming event sponsored by Parks is Habitech '93. Habitech is being billed as the first home automation industry annual trade show and is aimed at both professionals and consumers alike. "Habitech Professional" will be a two-day conference for those involved in producing and installing home automation equipment. "Habitech Weekend" will be a "media-sponsored consumer show with a rich program of seminars, demonstrations, and workshops for highly qualified end-user prospects." Habitech 93 takes place March 18-21, 1993, in the San Jose Convention Center. If you're in the area, be sure to check it out.

Other activities related to spreading the word about home automation in general include a new syndicated newspaper column written by (another) long-time CEBus committee member Dave Butler of Residential Micro Systems. Entitled "How to Automate Your Home," it is currently running weekly in numerous newspapers around the country, with the list of participating papers growing all the time.

The Home Automation Association (HAA) and the Canadian Automated Buildings Association (CABA) have for several years provided common meeting grounds for developers and manufacturers of home and building automation technology. If you are currently or are planning to become active commercially in building automation, or just want to find out more about getting into the field, contact one or both of these organizations.

BUT WHERE CAN I BUY...?

As any engineer knows (especially one who works for a large company), bringing a new product from the early stages of design to final release and shipping can take years. Because we're only now seeing the final CEBus specification, you're not going to see CEBus devices appear overnight. Such product development is also usually very secretive until just before final release, so trying to get details on CEBus products in the works can be next to impossible. I did manage to track down enough to make me excited about what we're likely to see in the future.

For close to a year now, Intellon Corporation has been shipping CEBus power line development modules based on the actual CEBus power line specification. While much of what I saw last year in the Bright Home was either based on preliminary specs or kludged up to work as CEBus will eventually work, these modules are the real thing. Intellon has developed a chip (manufactured by Motorola) that handles all the low-level power line details. Their power line modems include that chip plus an 80196 processor to demonstrate the technology and to allow a developer to use the interface chip in his or her own designs (see Photo 1).

I was able to borrow three of the power line modems and set them up around the editorial offices. While intended primarily for home use, where you have single-phase power and relatively short runs, I wanted to see how they'd perform in a rougher environment. Our building is much larger than a typical home and has far more computers and electronics plugged into the power lines than even Steve's house.

The evaluation software has three modes: Network, Transparent, and CAL. In network mode (the one I used], you can send packets of almost any length from any modem to any other modem and can specify what kind of network loading you want. Once configured, each node runs independently and keeps track of its own success or failure statistics. At the end of the test, the PC polls the nodes and presents a performance summary.

In one test that I ran overnight, over 100,000 packets were sent among the three nodes with transmission success rates close to 100%. And this was at the standard CEBus data rate of 10,000 one-bits per second. (Try that with X-10.) The scenario I set up loaded the line much more than would be the case in a typical house and sent more packets in a 16-hour period than most houses would see in a year.

In transparent mode, you use the power line modems as actual dumb modems. You connect a PC to each modem and can send information interactively between the PCs, including transferring files.

In the final mode, CAL, you can actually send real CEBus CAL packets to be acted upon by the receiving node. Each node has four digital inputs, four digital outputs, and one analog input, so they can function very similarly to one of our DIO-Link HCS II modules.

CEBus Bibliography

"CEBus: A New Standard in Home Automation," August/September 1989, issue #10, p. 40.

"CEBus Comes One Step Closer to Reality," April/May 1990, issue #14, p. 85.

"CEBus Update: How is the Health of EIA's Baby?" June/July, 1990, issue #15, p. S2.

"CEBus Gets Physical: The Standard Takes Two More Steps to Maturity," February/March 1991, issue #19, p. 103.

"CEBus Update: More Physical Details Available," June/July 1991, issue #21, p. 66.

"Home Control: Read All About It," December '91/January '92, issue #24, p. 98.

"Take a Tour of the Bright Home," February/March 1992, issue #25, p. 14.

"CEBus Goes Coax," February/March 1992, issue #25, p. 108.

The main difference is the CEBus unit doesn't need any extra wires even though it communicates at virtually the same data rate as the DIO-Link. (Don't bother comparing prices, though. The CEBus evaluation units are intended to be used for development, so they cost many times more than a DIO-Link.) If you're planning to do any CEBus power line development (or even development for other CEBus media), then this package would be a great way to get your hands dirty with some actual working equipment.

One of the first industries expected to pick up CEBus technology has very little to do with home automation: it's the power utilities. Demand for electricity is expected to continue to rise in the next decade and beyond, outstripping the utilities' capacity to generate it. In an effort to reduce the load, especially during such peak times as weekdays during the summer, power companies are looking at ways to *load shed* during peak times and offer consumers discounts on power after hours.

An application of CEBus power line communication actually being marketed by General Electric puts a CEBus power line transceiver inside an electric meter (dubbed the Phase3 Electronic Polyphase meter) installed on the side of a house. RF transceivers (not related to CEBus RF) mounted on light poles in strategic locations around the neighborhood communicate with the central office. The RF transceivers communicate through the power lines with the CEBus power line transceivers in the meters at the homes.

Using the network, the power company can then monitor electricity usage as often as necessary to vary the rate charged. Alternatively, the power company can send current rate information to the meter, which can relay it to smart devices in the house to allow the devices to decide when to run [or shut down) based on the current rate or to allow the homeowner to decide when to turn things on or off.

One application might include a dishwasher and an electric hot water heater cooperating with the meter to decide that the best time to heat the water and wash the dishes is in the middle of the night when rates are lowest. Another would be an air conditioner that resets its own thermostat to a slightly warmer setting during times of peak demand to lighten the load on the utility and lighten the homeowner's electric bill.

The complete system is called UCNet and includes voice as well as data communications to let the utilities keep in touch with their maintenance crews in addition to communicating with customers' meters. Be sure to refer to GE's product names rather than "CEBus" when requesting information.

THE TURTLE'S PROGRESS

The home automation industry certainly can't be faulted for rushing into things or moving ahead too quickly. As I've said before, a lot depends on consumer awareness and acceptance, which is going to take a long time. Until companies see consumer demand, they aren't going to commit a lot of resources. And until companies start pushing products, the consumer awareness isn't going to go up very quickly. Classic chicken and egg.

While progress has been slow, it has certainly been steady. We continue to see the development of new products, and the alert home automation enthusiast can still find plenty to keep the family and neighbors "ooing" and "ahing." With the final release of CEBus IS-60 and manufacturer awareness and consumer demand increasing, I think we're going to start to see a steady (but slow and, initially, expensive) stream of CEBus-compatible devices showing up in the coming months. If anyone comes across a new CEBus toy, be sure to let us know about it and I'll pass the information along to everyone else. ❑

Ken Davidson is the Editor-in-Chief of the Computer Applications Journal *and is a member of the magazine's engineering staff. He holds a B.S. in Computer Engineering and an M.S. in Computer Science from Rensselaer Polytechnic Institute.*

SOURCES

EIA, JEDEC, and TIA Standards and Engineering Publications
Global Engineering Documents
2805 McGaw Ave.
Irvine, CA 92714
(714) 261-1455
Fax: (714) 261-7892

Home Automation Association (HAA)
808 17th St. NW, Ste. 200
Washington, DC 20006

Canadian Automated Buildings Association (CABA)
M-20, 1200 Montreal Rd.
Ottawa, Ontario K1A OR6
(613) 990-7407
Fax: (613) 954-5984

The Training Department
5755 SW Natchez
Tualatin, OR 97062
(503) 692-8296

Parks Associates
5310 Harvest Hill Rd.
Suite 235, Lock Box 162
Dallas, TX 752305805
(214) 490-1113
Fax: (214) 490-I 133

Residential Micro Systems
14713 Pleasant Hill Rd.
Charlotte, NC 28278
(704) 588-2453

Intellon Corp.
5 150 West Highway 40
Ocala, FL 34482
(904) 237-7416
Fax: (904) 237-7616

G.E. Meter and Control
Bill Rector
130 Main St.
Somersworth, NH 03878
(603) 749-8578
Fax: (603) 749-8430

IRS

410 Very Useful
411 Moderately Useful
412 Not Useful

DEPARTMENTS

FIRMWARE FURNACE

Ed Nisley

Blindsided by Technology: The '386SX Embedded Firmware Project

Having long ago pushed the little 8031 to its limits, Ed has set his sights on a new development platform for future projects. While some may think his choice is overkill, you'll soon agree with his reasoning.

Suppose your project needs a high-performance **CPU,** lots of **DRAM,** comprehensive interrupt and **DMA** support, a real-time clock, nonvolatile configuration **RAM,** serial and parallel ports, and a megabyte-per-second I/O bus. Quick! How long will the hardware design take, and how much does the first unit cost?

Give up? How about five minutes and a hundred bucks!

Once again, the rules are changing.

COMMODITY COMPUTERS

The 8031 was designed (about **15** years ago!) as a microcontroller for projects that handled a few variables, communicated slowly, and didn't need much code. That it's still in use today is a testament to how well the designers did their jobs.

For small projects you just can't beat an 805 1. It is cheap, available on standard boards, and you can persuade it to do useful work quite easily. The catch is that you can design yourself into a corner where the next increment of performance costs much, much more than it should.

Recently, however, I've seen folks forcing 803 1s into tasks they just can't handle. A colleague produced an 803 1 hardware kludge with 2 MB (!) of code; the contract specified his CPU and EPROM choices. It worked, but, as the lady says, "Now, that just don't make no kind of sense!"

Consider that a standard '386SX system board now costs under $150 in single retail units. Money talks and it speaks clearly: you cannot afford to design a board with similar features for anywhere near that price. Are you listening?

It's too expensive. I don't need '386 performance; I'm using a '186. It's too big. I can't afford a '386 ICE. The I/O boards stick up. It uses too much power. There's no hardware documentation. I don't need a real-time clock. I can't justify embedding DOS. The BIOSes are incompatible. The vendors are unreliable. What a stupid idea!

I say again, the rules are changing. ***Circuit Cellar INK's*** prime tenet has always been, "Inside the Box Still Counts," and "Firmware Furnace" has taken you into the guts of firmware design with real examples from production products. That's not going to change, but now is the time for a bigger box!

Current '386SX boards are fascinating because you buy several orders of magnitude more performance than an 805 1 board for about the same price. From what I've heard on the BBS and over the phone, people are crying out for higher-performance systems; the 8051 has gone about as far as it can go. With cost not a barrier, there is little justification for stretching a teeny CPU beyond its limits.

As a result, I will use a '386SX system for some interesting "Firmware Furnace" projects. This series will still concern embedded firmware and will not be just another PC-programming column. There will be no arm waving here; I'm actually going to do firmware.

I'm also going to do it without going broke on development tools. In this column, I'll show you how to get started with nothing more exotic than a floppy drive; later on I'll get into EPROM boards, custom displays, and all that jazz. The code comes from the 8086 assembler you already have, but I'll slip into C in the next column.

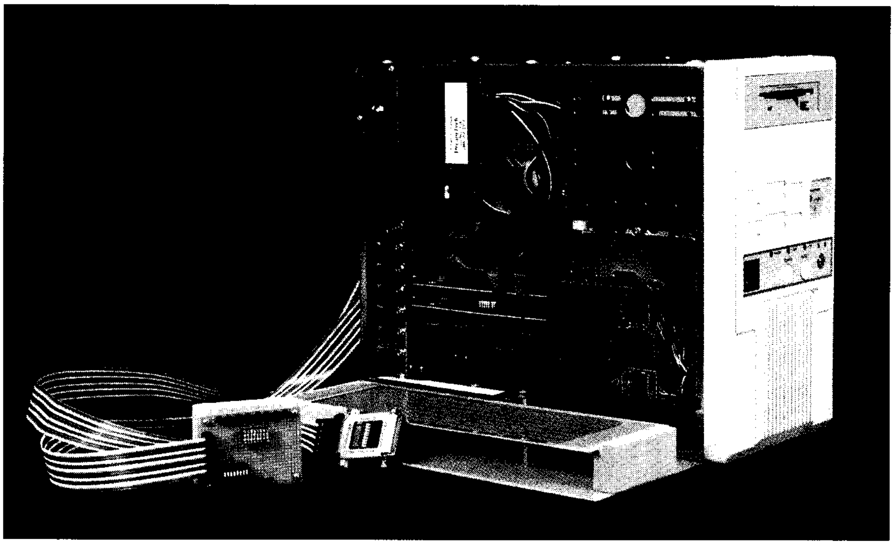


Photo I--The new ***Firmware Furnace*** *development platform* ***is based on a very inexpensive*** *'386SX* system.

Even if you don't need '386 performance, you'll get firmware ideas to build your own projects, hardware design tips to wire your system into the real world, and development techniques to get everything working. Best of all, you'll have some fun !

Finally, if you can't stand I/O boards sticking up in the air, I know a guy who'd love to build three-slot right-angle adapters just as those in small-footprint PCs.

ESSENTIAL HARDWARE

A '386SX system board is a commodity that you can buy nearly anywhere, but having a good source is nice. I talked to the folks at DreamTech, and they agreed to offer a special deal to "Firmware Furnace" readers who are working on these projects.

DreamTech's ***Firmware Furnace '386SX Package*** consists of a 33-MHz '386SX system board with 2 MB of 70-ns DRAM and an AM1 BIOS, a combination I/O board (with a parallel port, two serial ports, a game port, a mouse port, a floppy drive controller, and an IDE hard drive controller), a minitower case, and a power supply. If you ask for it by name, the package price is $259. (And, no, I don't get a cut!)

This package does not include a keyboard, display adapter, display, printer, disk drives, DOS, Windows, or other software debris, so it is not a complete PC system. Although as you will see, adding a 3.5" floppy drive is probably a good idea because doing so makes embedded firmware almost trivial.

Although the code from this column will run on any '386 PC, it is an Exceedingly Bad Idea to use your "real" PC as a guinea pig. A single slip of the scope probe or a minor programming error can put you out of business. The cost of the extra board is small compared to the aggravation of recovering your hard drive!

Besides, many of the firmware development techniques you will use require a PC to monitor the embedded system. Making things harder than necessary has no economical benefit; if you plan to build an embedded system, start working with the real hardware right away.

Photo 1 shows the development system I put together with the DreamTech hardware. I've added a JDR Microdevices 16-bit bus extender, built a card holder to support a prototype card, and chopped a hole in the case so it fits over the bus extender. On a LaserJet, the Bus La be 1 . L J file produced the yellow tag listing the I/O bus pins; BusLabel .ASC is the raw ASCII text so you can format it for your printer.

Incidentally, the two 3.5" drives next to the power switch are really blank drive bay covers. Tacky, no?

You must change the CMOS configuration parameters because all '386SX system boards are set up to be DOS PCs. Attach the display and keyboard from your main PC, run the BIOS setup routine, tell it to live without a display or keyboard, enable diskette booting, and tweak the date and time into your time zone. This process is a minor nuisance, but it's better than having to write the whole BIOS yourself!

BOOTING FIRMWARE

In my very first "Firmware Furnace" column, I described firmware as a kind of epoxy: mix the right combination of hardware and software and wait for it to set up like a rock. Although 805 1 firmware is usually associated with EPROMs, there's an easier way to inject firmware into our '386 system board-a floppy drive. Think of it as squishy firmware.

You may feel as though you're cheating, but the all-in-one I/O board I mentioned previously is under $40 (much less, in some cases), and a 3.5" floppy drive gives you **1.44** MB of storage for under $60. Price out the equivalent EPROM board and report back; perhaps your project can use a floppy after all?

I say again, the rules are changing.

No, a floppy drive isn't appropriate for applications that need rugged, all-weather, no-moving-parts hardware. But many (most?) embedded systems run in shirtsleeve environments where a floppy that spins only during system resets makes a lot of sense. Even better, data collection applications can now have PC-compatible data storage with essentially no overhead.

Also, contrary to what you might think, loading a program doesn't require DOS, either. Loading a **COM** file involves nothing more than copying the data from disk into memory, setting up a few registers, and jumping to the first instruction. **EXE** files are a tad more complex, but you'll flatten that problem in due time.

Our '386SX system board BIOS gets control after a hardware reset; tests and initializes the hardware; reads 5 12 bytes from Drive A, Track 0, Head 0, and Sector 1 into RAM at address 0000:7C00; and jumps to the first instruction. If that sounds as though you're loading a program, you're right; it's the standard bootstrap loader that starts DOS every time you turn on your PC.

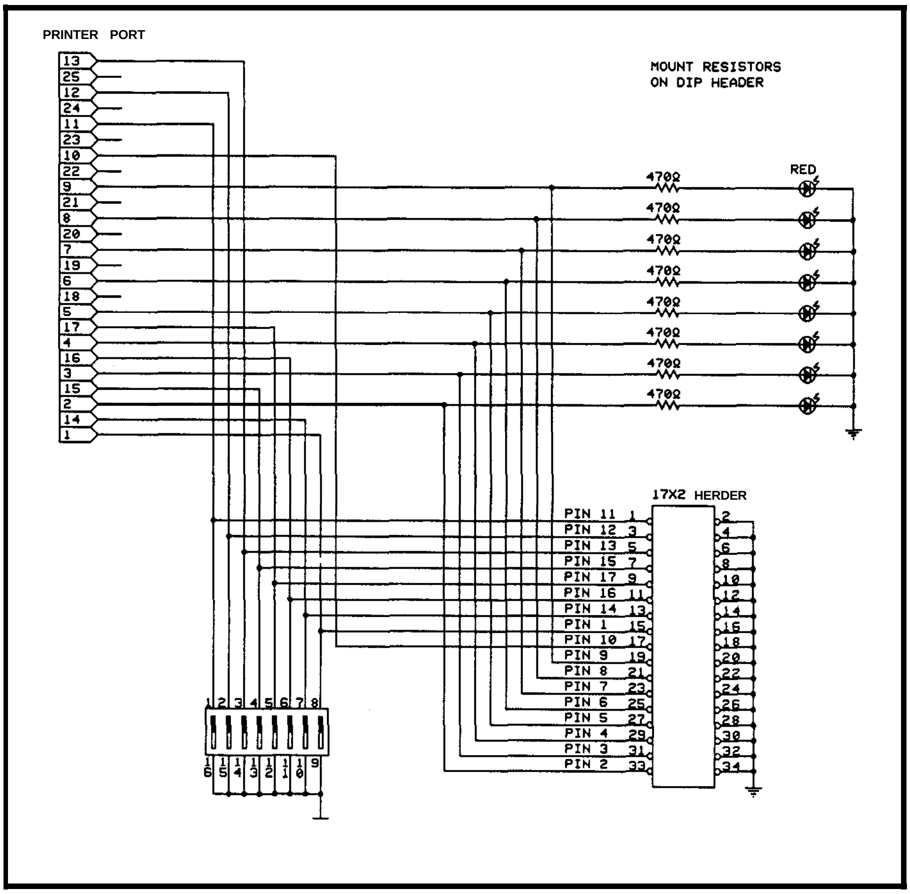


Figure 1-This circuitry *converts the printer port* **into** *one* **input** *and one output byte. The inputs are pulled up by resistors on the I/O card, while the outputs have (barely) enough drive for the LEDs. Note that the LEDs are ON when the outputs are HIGH, which is backwards from many LED drivers seen in these pages. The header provides convenient scope access, and will come in handy next month when I tackle performance issues.*

The BIOS neither knows nor cares what the program in that first sector does. In your case, you want to load an embedded program rather than DOS or OS/2, so you need a custom bootstrap loader that can find your program on the diskette and read it into RAM.

Now, here's the trick that makes this work all worthwhile. If your bootstrap loader knows how to interpret the DOS FAT file system, you can produce the embedded '386 program on a PC, use the DOS **COPY** command to put it on the floppy, stick the floppy in the embedded system, and hit reset. What could be easier?

THE BOOT'S STRAP

The first sector on each floppy disk contains both the bootstrap loader and a table called the Diskette Boot Record. Each type of diskette has a unique DBR, so it requires a custom bootstrap program to read its data. Fortunately, a little assembler magic can paper over the differences.

Listing 1 shows how this works. The **DISKSIZE** macro selects one set of constants and plugs them into the DBR variables. The values shown are used for a 720K 3.5" diskette. Although you could create a bizarre diskette format, I bet anything other than the standards will get you in big trouble with DOS.

Listing 2 shows how the bootstrap loader calculates the directory's location from the DBR values and reads the first sector into RAM. Each directory entry holds one file's starting cluster and size, so the loader uses the first entry to find the first file, reads it into RAM at address 0000:8100, and passes control to it.

There are a few restrictions; the embedded '386 program file must be the first one in the directory (although the loader will ignore a volume label), it must be contiguous (because the loader completely ignores the File Allocation Table), and it must be less than 64 KB long (as all COM files must be). In exchange, the loader sets up the registers so any COM program will work just fine. Fair enough?

Oddly enough, the standard DOS bootstrap loader has similar restrictions. The two DOS startup files (IO.SYS and MSDOS.SYS) must be the first two files in the 'directory and they must be contiguous. You can only jam so much intelligence into a 5 12-byte block!

The downloadable files for this column include four bootstrap loaders: BOOT720.SEC and BOOT1440.SEC handle the 3.5" formats, while BOOT360.SEC and BOOT1200.SEC cover the 5.25" field. You can use these directly or modify BOOTSECT. ASM to produce a customized version for your own system.

Listing 3 shows how to create a boot diskette for your '386SX system. The only tricky part is that DEBUG's L and W commands have changed slightly over the years; some versions use a sector range, while others use a sector start and count. Check your documentation to see which one you have.

THE FIRST PROGRAM

The canonical first program on any system displays a trivial message like C's "Hello, world" routine, but your '386SX doesn't have a display. Think small-your first 803 1 program probably blinked an LED, and that's the level you're at with this project. Unfortunately, a '386SX system board doesn't even have an LED!

Listing 1--The *Diskette Boot Record holds the information needed to read the rest of the diskette. Assembler macros define the constants starting with "N_" to create a boot record for each type of diskette. The bootstrap loader uses these values to read the DOS file directory and load the embedded system program.*

```
                IF      DISKSIZE EQ 720
                DISPLAY "Creating 720K diskette boot sector"
N-TRACKS        EQU  80
N_TRACKSIZE     EQU  9
N-HEADS         EQU  2
N_CLUSTERSIZE   EQU  2
N_ROOTFILES     EQU  112
N_MEDIAID       EQU  0F9h
N_FATSIZE       EQU  3

                ELSEIF
<<< other diskette data omitted here >>>
                ENDIF

N-SECTORS       =       N_TRACKS*N_HEADS*N_TRACKSIZE

                DB   'Firmware'      ; 03 OEM name and ver (8 chars)
SectorSize      DW   512             ; Ob bytes per sector
ClusterSize     D B  N_CLUSTERSIZE;  Od sectors per cluster
NumReserved     DW   1               ; Oe reserved sectors
NumFATs         DB   2               ; 10 number of FAT copies
MaxRootFiles    D W  N_ROOTFILES     ; 11 maximum root dir entries
NumSectors      DW   N-SECTORS       ; 13 total sectors on diskette
MediaID         DB   N_MEDIAID       ; 15 useless media descriptor
FATSize         DW   N_FATSIZE       ; 16 sectors in each FAT
TrackSize       DW   N_TRACKSIZE     ; 18 sectors / track (/ head)
NumHeads        DW   N-HEADS         ; la number of heads
HiddenSectors   DD   0               ; 1c number of hidden sectors
                DD   0               ; 20 number of sectors if > 32 MB
                DB   0               ; 24 internal DOS drive ID
                DB   0               ; 25 reserved
                DB   29h             ; 26 Boot signature
                DD   000FF00EDh      ; 27 volume ID number
                DB   'Firmware386'   ; 2b volume label (11 chars)
                DB   'FAT12   '      ; 33 file-system type (8 chars)
```

Listing 2—*This excerpt from the bootstrap loader reads the DOS directory, locates the first file, and reads it into RAM. That file may be any *. COM program written for the embedded '386SX system board.*

```
--- locate and fetch first directory sector
    this goes to 0000:7E00

      XOR    AX,AX                    ; start with zeros
      MOV    AL,[NumFats]             ; account for sectors in both
      MUL    [FATSize]                ; . . . copies of the FATs
      ADD    AX,[WORD LOW HiddenSectors]     ; these are surely
      ADC    DX,[WORD HIGH HiddenSectors]    ; . . . unused
      ADD    AX,[NumReserved]         ; boot sector and other
      ADC    DX,0                     ; . . . hidden sectors
      MOV    [AbsSector],AX           ; save start of dir
      CALL   CvtSectors               ; set up parameters
      MOV    BX,7E00h                 ; set up target
      CALL   ReadSector               ; and get one sector

    find sector address of data area just after directory
    we use directory size in bytes rounded to next whole sector

      MOV    AX,SIZE DirEntry         ; entry size in bytes
      MUL    [MaxRootFiles]           ; times entries
      MOV    CX,[SectorSize]          ; round upward
      ADD    AX,CX                    ; by sector size 1
      DEC    AX
      DIV    CX                       ; get sectors
      ADD    [AbsSector],AX           ; add to dir start

    find file's starting sector from cluster number

      MOV    AX,[ES:BX+DirEntry.FileStart]   ; first cluster
      SUB    AX,2                     ; . . . start5 at 2
      MOV    CL,[ClusterSize]
      XOR    CH,CH
      MUL    CX                       ; AX = sectors
      ADD    [AbsSector],AX           ; save for later

;--- convert file length from bytes to sectors
    we know the length must be < 64K bytes, so ignore high word

      MOV    AX,[WORD LOW ES:BX+DirEntry.FileLengthl
      XOR    DX,DX                    ; this is always 0
      DIV    [SectorSize]
      CMP    DX,0                     ; any remainder?
      JZ     HaveLen
      INC    AX
HaveLen:
      MOV    [SectorCount],AX         ; set up for reading

;--- now pull in the file to 0800:0100
    starting offset at 0100 allows nearly 64K worth of file w/o
    segment wrap

      MOV    BX,0800h                 ; set up target address
      MOV    ES,BX
      MOV    BX,0100h

NextSector:
      MOV    AX,[AbsSector]           ; set up starting sector
      CALL   CvtSectors

      CALL   ReadSector               ; and get one sector

      INC    [AbsSector]              ; step to next secor
      ADD    BX,[SectorSize]          ; . . . and next target

      MOV    CX,[SectorCount]         ; continue until done
      DEC    [SectorCount]
      LOOP   NextSector
```

#130

Listing 3—DOS DEBUG is the easiest way to create a bootable diskette for the embedded '386SX project. The diskette should have no hidden or system files, because the bootstrap loader simply pops the first file info RAM and runs if. Check your DOS manual to find out how DEBUG's L and W commands work on your system; the last digit may be either the final sector number or the number of sectors to transfer.

```
>format a: /f:720           create a DOS data diskette
<<< formatting stuff here >>> (you may use any diskette)
>debug                      fire up DEBUG
-l 100 0 0 1                read first few sectors from Drive 0
-nboot720.sec               specify our boot sector
-l                          load it in
-w 100 0 0 1                write one or two sectors to Drive 0
-q                          back to DOS
```

Figure 1 shows how to get started. Attach eight LEDs and eight switches to the I/O board's printer port. A '386SX is about one megaton of overkill for an LED blinker, but at least you can see if the program works

Listing 4 shows how COUNTSLO.ASM displays a counting sequence on LPT1 slowly enough to be visible to the naked eye. I used the BIOS WAIT routine to delay between counts; there's a wealth of handy routines in the BIOS code and letting them go to waste would be a shame!

I used Borland's TASM and TLINK to convert COUNTSLO.ASM into COUNTSLO.COM. The MAKEFILE renames it to COUNTSLO.BIN because I don't want to start the program by accident on my PC. The bootstrap loader doesn't care what the file's name is, so you can call it anything you'd like.

The MAKEFILE also puts the new file on the floppy in Drive A. Because the loader picks the first file, MAKEFILE executes a DEL A:*.BIN to wipe out any other embedded system programs, then copies the new BIN file to the floppy. While this process reduces the amount of typing needed to rebuild a program, be sure you have the right floppy in the drive before you start.

So here's the process briefly. Set up your '386SX board, build that little LED and switch board, copy the appropriate boot record and COUNTSLO.BIN to a floppy, pop it into the '386SX's drive, hit reset, and watch the lights.

That's all there is to it. Welcome to the wonderful world of embedded '386SX programming, minus all the aggravation and hocus-pocus you expected. It's all downhill from here!

The BBS files include several other simple programs to check out the switches and LEDs. They're simple enough that they should work on any PC, but some PC hardware isn't quite compatible with the IBM spec. For example, my Zeos '386SX laptop does not have pull-up resistors on the four control outputs. If the sample programs don't work on your PC, do a little volt meter probing before you claim a software problem.

Listing 4—The first embedded program displays a counting sequence on the parallel port so you can see something happening. The BIOS identifies the installed hardware so the port's address is in the usual low-RAM location and provides a wealth of handy utility routines as the calibrated delay used here.

```
        MOV     DX,0040h                ; aim at BIOS data segment
        MOV     ES,DX

        MOV     BX,2*(LPTPORT-1)+0008h  ; aim at port entry
        MOV     DX,[ES:BX]              ; get LPT port base address

        XOR     AL,AL                   ; start from zero...

ReShow:
        INC     AL                      ; tick it
        OUT     DX,AL                   ; send it out

        PUSH    DX                      ; save address
        PUSH    AX                      ; and count

        MOV     CX,COUNTRATE            ; delay for a while
        MOV     DX,0
        MOV     AH,86h
        INT     15h

        POP     AX                      ; recover count
        POP     DX                      ; and address

        JMP     ReShow                  ; continue forever
```

HOMEWORK

If you're like me, your PC knowledge slipped while you were working with diddly little CPUs. There's nothing wrong with that, but you can't specialize in one thing forever! Buckle down and do some research while you await your hardware.

The good news is that the PC is no longer uncharted territory, so much of what you need to know is ready for the reading. I've collected a list of my favorite books in the "Homework" sidebar, but surely I've missed some of the good ones.

The bad news is that these weighty technical tomes are in the $30–$60 range; you can deal a mortal blow to your budget with very little effort. Your local library may carry these books, but if you need them when they're loaned out to somebody else, well...

If your local bookstore can't supply your needs, give SBM Computers and Communications a call and tell Chris Waters I sent you. They carry or can get most technical books, some with a substantial discount. I've included other sources for some books that I know are hard to find.

Homework Bibliography

AT Bus Design
Edward **Solari,** Annabooks, o-929392-08-6

All the gory details of the PC/XT and AT I/O Expansion buses, documented with timing diagrams, signal descriptions, and cautionary notes. Information that you just cannot find anywhere else, written by somebody who knows whereof he speaks. This one is available from several mail-order electronics suppliers, as well as directory from the publisher: Annabooks, 15010 Avenue of Science #101, San Diego, CA 92128, (800) 462-1042

*System BIOS for IBM **PC/XT/AT** Computers and Compatibles*
Phoenix Technologies Ltd., Addison-Wesley Publishing, O-201 **-51806-6**

The straight dope on one vendor's BIOS implementation: good descriptions of what all the functions and data areas actually do. Although it may not precisely match up with the BIOS in your machine it'll give you a running start on the standard functions. It is considerably more verbose than the corresponding Official IBM BIOS documentation, which is not a bad thing.

The companion volumes may be of interest if you have the appropriate hardware: CBIOS for IBM PS/2 Computers and Compatibles and ABIOS for IBM PS/2 Computers and Compatibles. If you plan on protected-mode programming on PS/2s, the latter volume is the only source other than the Official IBM ABIOS documentation.

The Processor and Coprocessor
Robert **Hummel, Ziff-Davis** Press, I-56276-016-5

As it says on the back cover, "Finally-a single, concise technical reference on Intel's entire family of 80x86 processors and 80x87 coprocessors." Exhaustive detail on how the machinery at the heart of your PC munches instructions and data, with lots of information on protected mode operation and memory management. It includes a chip bug summary by CPU mask step, with workarounds.

PC Interrupts
Ralf Brown and Jim Kyle, Addison-Wesley Publishing, O-201 **-51806-6**

A huge tabulation of every hardware and software interrupt used by every PC program inside Mars orbit. Each entry includes a brief functional description, input and output register contents, warnings on conflicts with other interrupts, and occasionally some functional details. The text is available on-line from several sources, but I suspect the book is cheaper than the download charges.

The Programmer's PC Sourcebook
Thorn Hogan, Microsoft Press, 1-55615-321-X

A category killer book with (almost) everything you could possibly want to know about your PC, albeit in desiccated tabular format. There is nary a paragraph of narrative text so you'll have to depend on other sources for the "why" behind each function. This book is the one I reach for first when trying to decipher a connector, socket, interrupt, DOS/BIOS function, or chip pinout. Other books have the explanations, but this one covers the territory. I've found a few errors, but that's why you need more than one book.

The Zen of Assembly Language, Volume 1: Knowledge
Michael **Abrash,** Scott, **Foresman** Professional Books, *O-673-38602-3*

Abrash is the high priest of PC code optimization, and this book summarizes his magic. It's written for the venerable 8088, but includes hints and tips for CPUs through the '386. Believe me, if you follow his precepts you don't have to apologize to anybody for anything. This book may be extremely hard to find because ScottForesman imploded shortly after producing it (there was no cause-and-effect relationship). Give Bookstream a call at (602) 483-0192 to see if they have any copies left.

Microprocessors: A Programmer's View
Robert Dewar and Matthew Smosna, McGraw Hill, *o-07-01 6639-O*

A savage romp through CPU taxonomy. You get a detailed explanation of CPU functions and addressing modes, lengthy descriptions of the 80386 and 68030, the '387 coprocessor, as well as a chapter apiece devoted to the SPARC, Intel 860, IBM RISC, and Transputer RISCs. After reading about the 80386 protection mechanisms, you find a section titled, "Is All This Worthwhile!" starting with the memorable sentence, "The previous section is virtually incomprehensible." Don't miss it!

UPCOMING EVENTS

In the next column (next month!), I'll describe how to persuade Dave Dunfield's 8086 version of the Micro-C compiler to produce programs for the floppy drive loader, then I'll take a look at low-level hardware performance. After all, you need to know just how fast this beast will run before you decide how much you can do with it.

After that I'll get into the first real hardware project-a firmware development card with some EPROM, battery-backed/write-protected static RAM, a watchdog timer/power monitor, more serial and parallel ports, character LCD and keypad interfaces, and a few other handy widgets.. .along with the firmware to drive them. Covering this card in depth will take several issues, but it will simplify the rest of the projects.

Projects will include an interface to those big LCD graphics panels, a multichannel DMM to collect precise voltage data, voice I/O (it's easy with lots of memory!], stepper motor controls, and high-speed analog I/O. In fact, if the performance measurements in the next issue work out, cheap digital video is a distinct possibility.

On the software side, I'll use Dunfield's Micro-C for a while because it is the best cost-performer going. However, I'll eventually segue into Borland's C++ (with Paradigm's **LOCATE** utility to handle the EXE-to-HEX conversions) because Paradigm's **TDREM utility** lets me use Turbo Debugger over a serial link to dig into the innards of the firmware at the source and object levels. This method is even better than debugging on your own PC because the embedded program can't crash your system!

I finally have a peppy CPU with real instructions, lots of RAM, plenty of disk storage, and familiar compilers and debuggers. Now I'm gunning for interesting problems! ❑

SOFTWARE

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

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

Firmware Furnace '386SX Package
DreamTech Computers
5 175 Moorpark Ave.
San Jose, CA 95129
(800) 237-3263
(408) 996-2373
Fax: (408) 996-2383

80x86 Micro-C Developer's Special
Dunfield Development Systems
P.O. Box 31044
Nepean, Ontario K2B 8S8
CANADA
(613) 256-5820
Fax: (613) 256-5821

LOCATE and TDREM utilities
Paradigm Systems
3301 Country Club Rd., Ste. 2214
Endwell, NY 13760
(800) 537-5043
(607) 748-5966
Fax: (607) 748-5968

Technical books by mail
SBM Computers & Communications Corp.
2906 William Penn Hwy.
Easton, PA 18042
(215) 559-9999

16-bit bus extenders & protocards
JDR Microdevices
2233 Samaritan Dr.
San Jose, CA 95124
(800) 538-5000
(408) 559-1200
Fax: (800) 538-5005

IRS

413 Very Useful
414 Moderately Useful
415 Not Useful

Simple Steps for Easy Positioning

Controlling Stepper Motors

What do you do when you need precise motor control without a feedback loop? Use a stepper motor. Basic stepper motor control doesn't have to be hard, especially given the caliber of controller chips available.

FROM THE BENCH

Jeff Bachiochi

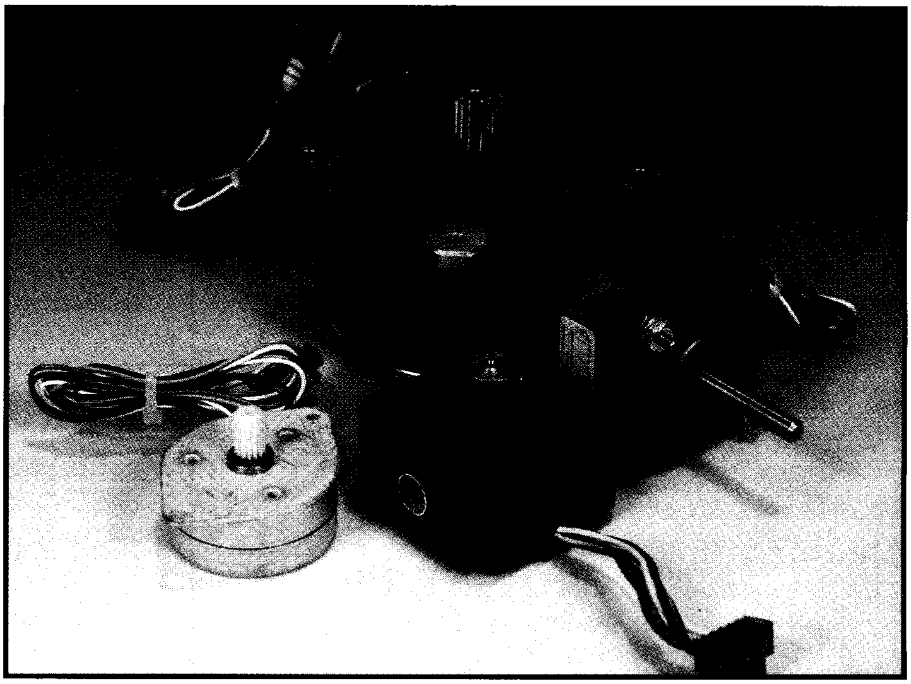
O adies and gentlemen step right up. That's right, just a little closer. You don't want to miss this amazing offer. You say you have positioning problems! (Get away from me kid, ya bother me.) Have I got the cure for you. It's not an ointment. Not a cream. Not even the ever-popular Elixir of Life. No friends, it's the all powerful stepper motor/translator driver. Yes, I can see the "pshaw" starting to form on all those lips, even as I speak. Well, let me assure you, you won't believe how easy movement will be from this day forward. No more aches, no pains, no complications. So come on in, step right this way.

CONTROLLING MOTION, DIGITALLY

Digital signals provide a simple interface for motion control using stepper motors. Steppers, being incremental by nature, offer a good compromise between accurate positioning and simplicity. You'll find them in many computer peripherals used today, such as printers, faxes, disk drives, and plotters. In fact, the stepper motors used in industrial robotics are responsible for the precision placement, machining, and assembly of these very same products.

Magnetic attraction between a motor's stator and rotor causes a rotation of the rotor until opposite poles align. When the coil sequence is advanced automatically, either by the line frequency or brushes and commutators segments, the continually advancing magnetic field draws the rotor through complete rotations. The stepper motor does not employ this automatic action. Instead, the user is responsible for each and every movement. The user can calculate how many steps it takes to move a desired distance or rotation because each movement is a precise step of known distance or angular arc.

The design's simplicity keeps the cost of implementing a stepper motor



The variety of stepper motoros available is almost endless. Motors come in all shapes, sizes, step angles, and available torque.

into a project low. Steppers generally run open loop without the costs associated with shaft encoders.

THE LINGO

The angular movement of a stepper motor is measured in degrees. The number of degrees the shaft rotates for each winding's phase change, or step, is its *step angle.* The most common step angles are between 1.8" and 18". The step accuracy is the difference between the theoretical and the actual step angle, which is generally ±5% of the step angle and is nonaccumulative.

A stepper that remains stationary while energized is exhibiting a resistance to change called *holding torque.* This torque is the maximum amount that you can apply to a shaft without causing it to rotate. Holding torque is measured while the coils are energized at rated current, while *detent torque* is measured with no coil current applied.

You can express torque in English or metric units. I'll be using the gram-centimeter (g-cm). Here are some equivalents:

1 oz.-in. = 72.0077 g-cm
1 lb.-ft. = 13,825.48 g-cm
1 N-m = 10,197.16 g-cm

Dynamic torque is the force a stepper motor produces while in motion. The smallest steppers have dynamic torques of less than 100 g-cm, while large frames can produce over 100 kg-cm. You may be more familiar with the term *foot-pound* (ft.-lb.) as used by the automotive torque wrench.

The user fully controls the step rate for stepper motors, usually given in steps per second (sps). Maximum step rates have a range of about 500-10,000' sps. Achieving high step rates while avoiding resonance bands often requires special *ramping,* or the gradual change between two different step rates. Resonance ranges occur where the rotor begins to oscillate at the natural frequency of the spring-mass system. Within this range, the available torque falls to a very low value, and the loss of synchronization may develop if the speed remains the same.

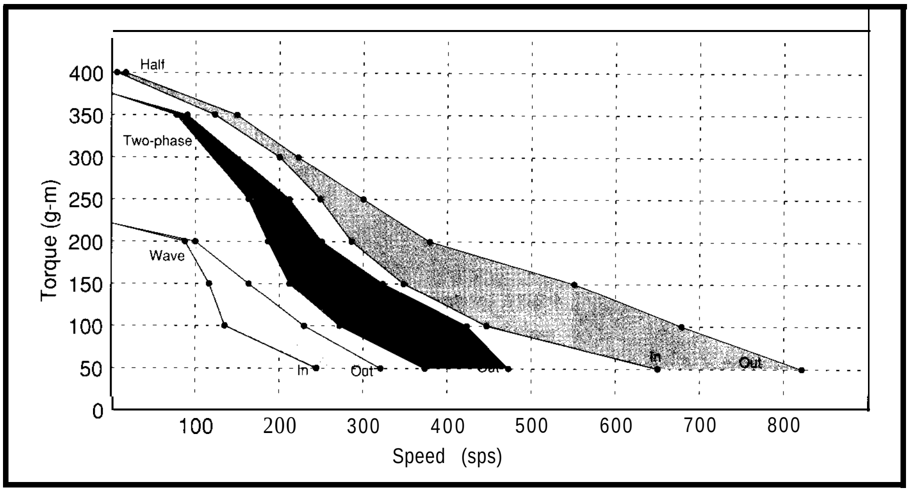


Figure 1—*For each of the three common step techniques, torque is inversely proportional to the step rate. Each also has a maximum pull-in rate and pull-in torque and a companion maximum pull-out rate and pull-out torque.*

As stepping rates increase, the available torque normally decreases. The *pull-in rate* refers to the maximum steps per second at which the stepper motor can start, stop, or change direction with a specific load without losing a step. The *pull-in torque* is the maximum load that can be started or stopped without losing a step. There is a maximum pull-in rate for each torque value and a maximum pull-in torque for each step rate. I've shown this relationship graphically in the torque-speed curve (see Figure 1).

Once a stepper motor has started, the pull-out rate refers to the maximum steps per second that the motor will drive a specific load and not lose a step. The pull-out torque is the maximum load a motor can handle at a specific step rate and not miss a step. I've also illustrated this relationship in the torque-speed curve.

BASIC STEPPER MOTOR CONSTRUCTION

There are three main types of stepper motor construction: variable reluctance, permanent magnet, and hybrid. The variable reluctance motor has a soft iron multitooth rotor. The rotor's closest tooth is attracted to the motor's wire-wound stator poles when a coil is energized. Because the rotor has no magnetic qualities, there is no attraction (detent torque) holding the rotor still while no power is applied, although it's capable of high step rates.

The permanent magnet stepper motor has a toothless rotor made into a permanent magnet. The rotor's poles are attracted to the motor's wire-wound stator poles when a coil is energized. The permanently magnetized rotor remains attracted to the closest stator pole even when no power is applied, giving this style of motor detent torque, but lower stepping rates.

The third and most popular style of stepper motor construction is the hybrid. The hybrid combines the qualities of the variable reluctance and permanent magnet rotors. This construction produces a stepper with high detent, holding, and dynamic torque while retaining the high stepping rates.

BIPOLAR VERSUS UNIPOLAR

Bipolar stepper motors are wound once on each stator pole. The magnetic polarity of the stator pole is switched by reversing the current in the coil. This reversing requires the use of a bipolar power supply or an H-bridge. It has higher torque (the complete coil is energized), but must use a more complicated power supply.

Unipolar stepper motors have double windings on each stator pole, one in each direction. Switching the magnetic polarity of the stator pole is accomplished by simply switching the current from one coil segment to the other. This step does not require a bipolar power supply, but offers less torque (the double-wound poles have only half of the turns energized at once).

THE UCN5804B UNIPOLAR STEPPER MOTOR TRANSLATOR/DRIVER

Four transistors and a bit of software are all that's necessary to drive a stepper motor's coils in the proper sequence, but it requires the processor's full attention. If your system has other things to attend to and you would like to keep everything simple, then use the single-chip approach. The UCN5804B from Allegro Microsystems has built-in sequencing logic for three types of step formats as well as direction control. Transistors are included to drive a unipolar motor up to 1.5 amperes from a 50-volt source. This 16-pin batwing DIP has internal clamping diodes and internal thermal shutdown (try that in software!).

I interfaced the 5804 to the PC's parallel port for easy experimentation (see Figure 2). Five digital output bits control the 5804's functions-two bits set the step format, one bit enables stepping, one bit sets the direction, and the final bit is a step signal. The port's strobe line is used as the step signal because it will produce a complete pulse (500 us in length) for each write to the parallel port, which gives BASIC easy control of the driver by simply LPRINTing. Notice that if the step format is hardwired, only two bits (direction and enable) are needed, allowing the user to interface up to four stepper motors to the same parallel port. No feedback comes from the 5804 (it runs open loop) so you don't need any input bits.

What's all the extra circuitry for? A safety interface that you may wish to construct. In applications where the actual physical limits to the movement will cause harm if exceeded, you can employ limit switches to disable further stepping in the unwanted direction. This circuit can use either normally open or normally closed switches to protect limit positions. If either limit switch is pressed, a signal is generated that will prevent additional steps from passing through to the 5804. When the direction line is changed, the step disabling signal is reset. You can eliminate this circuitry if you have the limit switches fed back as inputs and your software uses them to prevent stepping past the limits.

Another useful input is a home or center position indicator, which tells you when the stepper's load is in a known position. The schematic shows all the possibilities for completeness, though you may want to simplify it.

STEP FORMATS

The 5804 stepper motor translator driver chip supports three step formats: wave, two-phase, and half-step sequences.

The wave drive sequence energizes only one coil at a time. The sequence is either ***A-B-C-D-A*** or ***A-D-C-B-A,*** depending on the direction control bit. The wave drive power supply must provide current to only one winding at a time, making it less expensive. The rotor moves from one energized stator pole to the next. The torque produced is also small because it is proportional to the number of windings energized.

The two-phase drive sequence is much the same as the wave drive. The difference being that two poles are

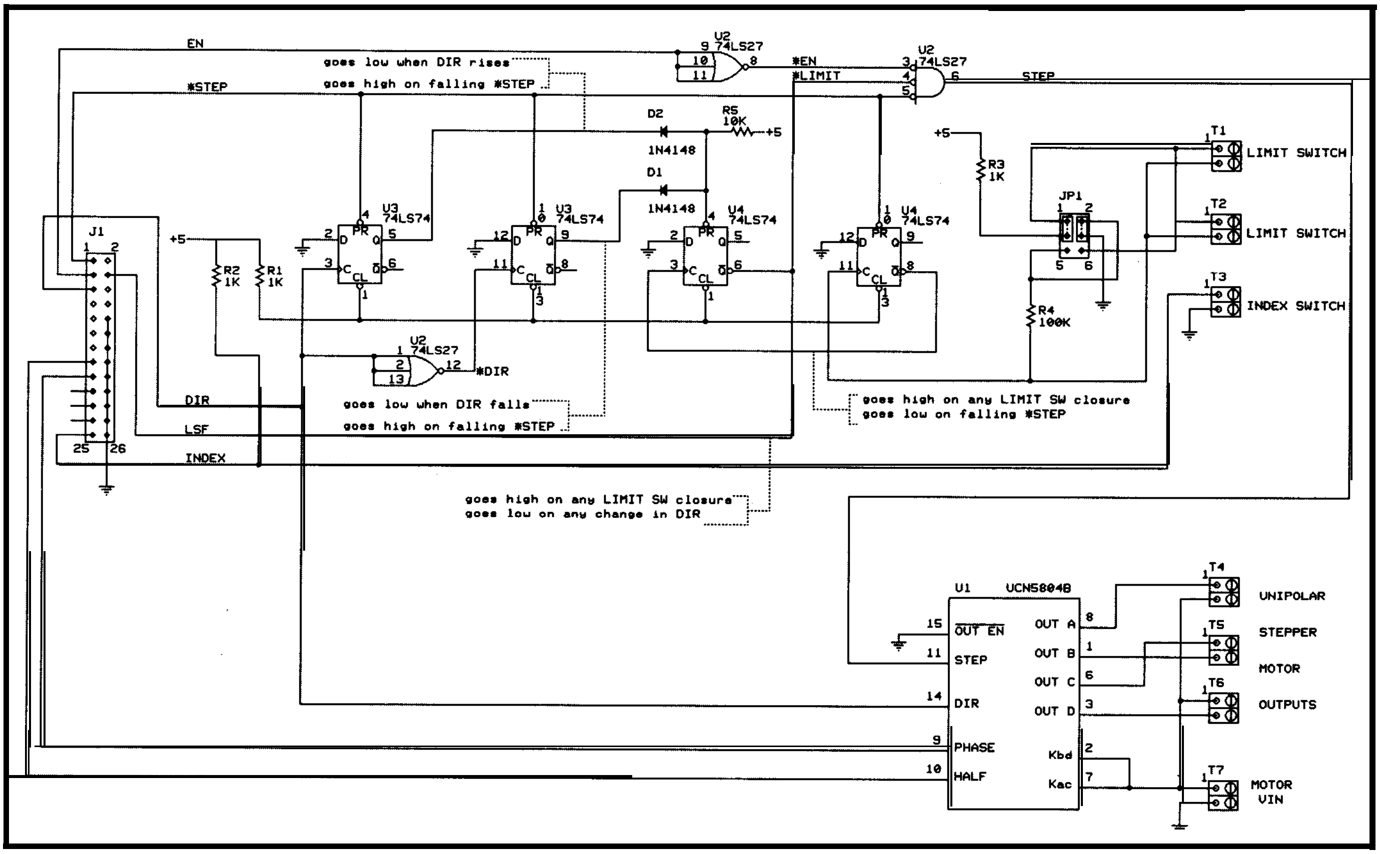


Figure 2-A typical PC parallel port makes an easy interface for experimenting *with stepper motors. The UCN5804B is the key to controlling the motors, while extra* discrete latches accept *input from limit switches to protect the* device being *controlled.*

energized at one time. The sequence is either ***AB-BC-CD-DA-AB*** or *AD-DC-***CB-BA-AD,*** depending on the direction control bit. The two-phase power supply needs to supply current to two windings at a time, making it more expensive. The rotor moves from "between two energized poles" (the attraction to both energized stator windings) to "between the next two energized stator poles." The tug of two energized windings increases torque.

Notice that the wave sequence and the two-phase sequence both move a whole step, while each starts at different points. The wave drive from pole to pole and the two-phase drive from half-way between poles to half-way between poles. Combining these two drive sequences creates the half-step drive sequence.

The half-step drive alternates between a single-coil wave drive and a two-coil two-phase drive. The half-step drive sequence is either *A-AB-B-BC-C-***CD-D-DA-A** or *A-AD-D-DC-C-CB-B-***BA-A,** depending on the direction control bit. The power supply must provide current to two windings even though every other step requires energizing only one, so it is more expensive like the two-phase drive. The torque increases because the steps are smaller and alternating ones have the tug of two windings. The smaller steps require two steps per whole step, increasing positional accuracy but cutting the step rate in half.

Listing l-Using a dedicated motor control chip results in a control program that consists mostly of user interface. The code that actually deals with the motor is very short.

```
10 Y=50
20 ENABLE=1: DISABLE=0
30 FORWARD=2: BACKWARD=0
40 WAVE=128: TWOPHASE=O: HALFSTEP=64
50 LPRINT CHR$(0);
60 GOT0 290
70 PRINT "SELECT AN ITEM"
80 PRINT: PRINT "1 ENABLE"
90 PRINT "2 DISABLE"
100 PRINT "3 FORWARD"
110 PRINT "4 BACKWARD"
120 PRINT "5 WAVE DRIVE SEQUENCE"
130 PRINT "6 TWO-PHASE DRIVE SEQUENCE"
140 PRINT "7 HALF-STEP DRIVE SEQUENCE"
150 PRINT "8 STEP MOTOR"
160 PRINT "9 END"
170 I$=INKEY$
180 IF I$="" THEN 170
190 IF I$="1" THEN STATE=ENABLE: GOT0 290
200 IF I$="2" THEN STATE=BACKWARD: GOT0 290
210 IF I$="3" THEN DIRECTION=FORWARD: GOTO 290
220 IF I$="4" THEN DIRECTION=BACKWARD: GOT0 290
230 IF I$="5" THEN TYPE=WAVE: GOT0 290
240 IF I$="6" THEN TYPE=TWOPHASE: GOT0 290
250 IF I$="7" THEN TYPE=HALFSTEP: GOT0 290
260 IF I$="8" THEN GOT0 380
270 IF I$="9" THEN STOP
280 GOT0 170
290 PRINT: PRINT "The motor is I';
300 IF STATE=0 THEN PRINT "Disabled, "; ELSE PRINT "Enabled, ";
310 IF DIRECTION=0 THEN PRINT "Backward, "; ELSE PRINT "Forward. ";
320 PRINT "set for ";
330 IF TYPE=0 THEN PRINT "2-Phase drive": GOT0 350
340 IF TYPE=64 THEN PRINT "Half-step drive" ELSE PRINT "Wave drive"
350 VALUE=STATE+DIRECTION+TYPE
360 PRINT "The hex value is ";HEX$(VALUE)
370 GOTO 70
380 PRINT:I NPUT "How many steps";S
390 PRINT "S to stop, + or to change step speed"
395 PRINT "step speed = ";Y,
400 FOR X=1 TO S
410 LPRINT CHR$(VALUE);
420 I$=INKEY$
430 IF I$="+" THEN IF Y>9 THEN Y=Y-10: PRINT Y,
440 IF I$="-" THEN IF Y<991 THEN Y=Y+10: PRINT Y,
450 IF I$="S" THEN X=S
460 FOR Z=O TO Y: NEXT Z
470 NEXT X
480 PRINT
490 GOTO 290
```

ROTARY TO LINEAR MOTION

The rotary motion of stepper motors is often translated into linear motion for positioning systems and can be accomplished in several different ways. Line printers can use a rack and pinion to move a print head across the paper. Some floppy drives use a cylinder and band to move the read/write head from track to track. Plotters might use a pulley and cord to position the plotting pen. NC drilling machines use gears and lead screws for linear positioning. Even line printers and fax machines use stepper motors and reduction gears to move paper through their mechanisms, which is a good example of how gearing can be implemented to increase the usefulness of a low-torque stepper. The gear ratio magnifies the torque (neglecting the efficiency of the transmission).

This process is an important tradeoff when designing with a stepper motor. As a poet once wrote, in this world, ".. nothing is had for nothing[1]." The trade is speed for torque or torque for speed. Any stepper has a maximum step speed and torque. You must determine the characteristics necessary of a stepper motor before you can choose one to do the job.

INERTIA

A body at rest tends to stay at rest, and one in motion tends to stay in motion. This law is extremely important when moving loads with a large mass. How fast does your automobile go from 0 to 60 MPH? Not instantly? No matter how hard you tromp on the gas, your vehicle takes time to accelerate to cruising speed. The constant force of those horses under the hood pulling or pushing you continuously add together to move the large vehicular mass. Coming to a stop is not instantaneous either, unless you're a crash dummy. Decelerating the hulking mass back down to 0 MPH requires continual force.

When using a stepper to move any large mass, the mass must be accelerated to obtain a desired speed and decelerated when bringing it to a stop under synchronous control. The mass must be accelerated or decelerated at a predetermined rate to ensure steps are not lost (under acceleration or deceleration). To compute this rate, use the motor's torque, the stepping angle, and inertial loading.

The object's mass is defined as its weight divided by the gravitational constant. Let me use an object with 1000 grams as its weight.

$$\text{Mass} = \text{weight}/g_c$$
$$M = 1000\ g / (980\ cm/s^2)$$
$$M = 1.02\ g\text{-}s^2/cm$$

The inertial loading or resistance to change in speed or velocity of this mass is based on the distance or distribution of the mass about point of rotation. The formula can be complex if the load has a complex cross-section. Using a solid cylinder with a radius of 1 cm for this example, the formula is

$$\text{Moment of Inertia} = (\text{Mass})\ (\text{Radius})^2$$
$$I = (1.02\ g\text{-}s^2/cm)\ (1\ cm)^2$$
$$I = 1.02\ g\text{-}cm\text{-}s^2$$

The torque necessary to move this inertial load is dependent on the angle of movement per unit time. If the step angle is 3.75° and the acceleration is 96 steps/s², then

$$\text{Torque} = (\text{Moment of Inertia})\ (\text{Acceleration})\ (\text{Step Angle})\ (\pi/180)$$
$$T = (1.02\ g\text{-}cm\text{-}s^2)\ (96\ steps/s^2)\ (3.75^\circ)\ (0.01745)$$
$$T = 6.4\ g\text{-}cm$$

By ramping a stepper motor capable of producing only about 10 g-cm of torque at a rate of 96 (3.75°) steps/s², you could move a 1000-g platter up to a speed limited to that defined by the torque/speed curve of the stepper. The most common uses for steppers utilize design parameters that match the stepper to the load without the use of ramping.

MEASURING THE EFFECT OF STEP FORMAT ON TORQUE

To make good use of the 5804's versatility, I wanted to experiment with the different wave formats to determine the torque differences between wave, two-phase, and half-step drives. I chose a Seiko STP4296L-50 stepper motor (available from Digi-Key as part number STP112) for my tests.

The test setup includes a platform that holds the stepper's motor shaft parallel with the floor at the edge of the bench. Since I was dealing with torques in the range of gram-centimeters, I attached a pulley with a radius of 1 cm to the stepper's shaft so the torque wouldn't require any interpolation. I also attached a piece of cardboard behind the pulley, marked it with full-step increments (3.75°), and stuck a pointer to the pulley to aid in identifying full-step movement. I connected a thin, steel wire to the pulley at a point on the circumference and tied the opposite end to a clear plastic bag that will hold weights. The setup is similar to raising a bucket from a water well.

The pull-in and pull-out rates of the stepper motor are measured for each increasing increment of weight hung from the pulley. The test is repeated for each of the three step formats, with the results plotted. A pulse generator is applied to the step input of the 5804, giving me manual control of the step rate. The pull-out rate is recorded when the speed is increased until the weight can no longer be moved without losing synchronization (a step). The pull-in rate is recorded when the speed is decreased and the motor again acquires synchronization. This test is a bit less than perfect regarding accuracy, but my real interest here is in the relative torques between the different stepping formats. My test results are plotted in Figure 1.

For small motors, power supply costs are not a big factor. In some larger applications they might be a significant part of the project. In this case, a savings could be made by choosing a step format based on power supply simplicity. I neglected power supply considerations on purpose. A stepper motor supply can vary from simple L/R resistive limited supply to a much more complex chopper style. Let me save these, which by the way include microstepping, for another time. For now, grab hold of that broken piece of equipment, strip out the stepper motor, and give your PC the power to position. ❑

Jeff ***Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on the Computer Applications Journal's engineering staff, His background includes product design and*** *maufacturing.*

[1] Arthur Hugh Clough, ***The Bothie of Tober-na-Vuolich, pt. VIII, 1848.***

SOURCES

Allegro Microsystems, Inc.
115 Northeast Cutoff, Box #15036
Worcester, MA 01615
(508) 853-5000

American Precision Industries
4401 Genessee St.
Buffalo, NY 14225
(716) 631-9800

NMB Technologies, Inc.
Motor Division
9730 Independence Ave.
Chatsworth, CA 913 11
(818) 341-3355

Seiko Epson Corp. (from Digi-Key)
(800) 344-4539

Sigma Instruments, Inc.
170 Pearl St.
Braintree, MA 02184
(617) 843-5000

Stock Drive Products
2101 Jericho Tpke.
New Hyde Park, NY 11040
(516) 328-0200

I R S

416 Very Useful
417 Moderately Useful
418 Not Useful

Smart LEDs

The Hard Way, the Soft Way, and the Right Way

The ubiquitous LED can be found in countless display applications. New display modules and driver chips are making the job of adding a highly readable display to your next project even easier.

SILICON UPDATE

Tom Cantrell

As silicon intelligence permeates systems, designers making the tradeoff between hardware and software face more choices than ever. Traditionally organized design teams, in which hardware and software groups work separately, are subject to a disadvantage best described by that old saying, "If the only tool you have is a hammer, everything looks like a nail." Each side may lobby that their way is the *right* way, both ignoring possibly more optimal "firm" (combined hardware and software) solutions.

Besides depending on basic functional requirements, your choice must also consider *make versus buy* and *time-to-market* realities. Another important consideration is software's consumption of developmental cost and time in contrast to hardware's predominate effect on unit cost.

All these considerations mean the same functional requirements may lead to completely different designs, depending on business issues. Even for a seemingly simple function like an LED display, designers face an ever-growing spectrum of hard, soft, and firm alternatives.

THE PAUSE THAT REFRESHES

Often, an application requires only the simplest of displays like a two-digit "O-99" counter. The ubiquitous seven-segment LED (which is actually eight-segments if you need a decimal point) can handily represent the digits 0 through 9 and other characters as well (within the limitations of its sparse matrix).

The brutally straightforward approach is simply to allocate seven lines per display, driving each LED display segment as you would a discrete LED. In this case, the driver software is little more than a lookup table (specifying the segments associated with each digit) and an OUT statement to set the corresponding output bits. However, the unit hardware cost explodes for more than a display or two, what with the need to replicate output lines, high-current drivers, and current-limiting resistors for each.

Enter the widely used time-multiplexing scheme. It relies on the simple capability of an LED at $Y \times X$ brightness, which is lit only I/Y percent of the time, to impersonate an LED at constant X brightness. Figure 1 is a good example of this technique, showing how little more than 8 bits of output/driver/current-limiting resistor can handle three seven-segment displays (plus, in this example, four discrete LEDs). However, saving hardware ("pay later") does require software ("pay now") in the form of an interrupt handler as shown in the flowchart in Figure 2.

At each interrupt, the handler outputs data (i.e., one of the three seven-segments or the discrete LEDs as a group) and enables the display in a round-robin fashion. Though each display is on (or refreshed] just one-quarter of the time, a simple matter of boosting the current effectively compensates for this limitation.

Actually, the software itself isn't very intimidating, and the amount of bandwidth consumed for refresh isn't excessive (even at 1 kHz, the handler should only consume a few percent of processing time). The real problem is that the refresh frequency needs to be fairly high in order to avoid flicker. Personal taste determines the point at which flicker becomes annoying, but you should probably shoot for no less than 100 Hz per display (i.e., the interrupt frequency is equal to the number of displays x 100 Hz).

The only potential problem with this scheme is when you're dealing with a lot of other real-time interrupts. In that case, you have to decide which interrupts have priority and how much

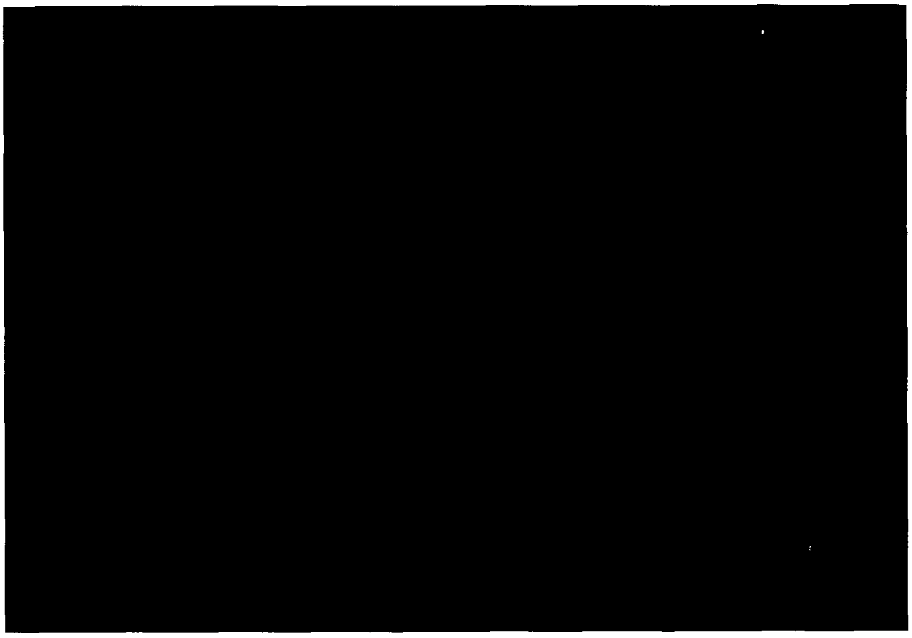


Photo 1—*LED modules made up of 64 LEDs are popular in scrolling displays and other applications where large letters are important.*

latency each can tolerate. If you give the LEDs high priority, you boost the latency of lower priority interrupts by the LED refresh handler execution time. On the other hand, if you give LEDs low priority, watch out for unexpected flicker should the refresh interrupt get locked out by higher priority tasks.

I'm not saying it won't work out—indeed, the widespread use of this technique proves it often does. A large, stable company with lots of engineers and money can afford to study the situation thoroughly in the interest of minimizing unit costs.

On the other hand, pretend you're the VP of engineering at hot start-up GO-PUBLIC Corp. Now, you may be VP of engineering, but, as usual for cash-starved garage shops, you're also the only engineer. You really don't have the time, money, or staff to fool around with messy details; you need something working yesterday. Worse, the specs (scrawled on a napkin, of course) are sure to change before GO_PUBLIC finally gets funding. A carefully crafted LED refresh scheme with detailed timing analysis may get trashed by a simple last minute request: "Gee, adding four more digits wouldn't be that hard-would it?"

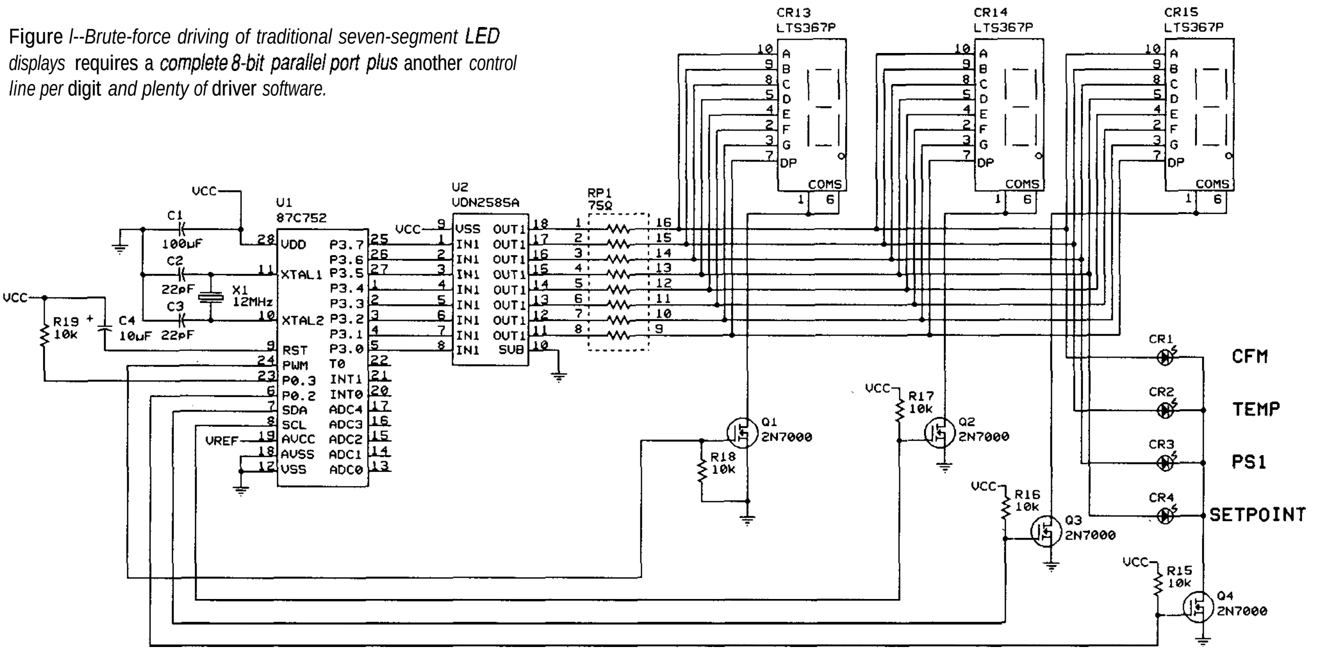
IC THE LIGHT

What you need is a chip that combines the high-current drivers, refresh logic, and an ASCII lookup table. Lucky for you, Maxim has just announced the MAX7219.

Conceptually, the chip works the same as the CPU-based driver/transistor setup. Eight-segment drivers connect to all the LEDs in parallel, with eight-digit drivers enabling individual displays. The '7219 has a built-in oscillator for refresh timing. It eliminates your worries about flicker because the oscillator runs at about 10 kHz, refreshing an individual display at least 1,000 times per second (i.e., 10 kHz/eight digits = 1,250 Hz). More important, the '7219's capability to handle refresh means you can kiss those CPU interrupts goodbye.

Designing with the '7219 could hardly be easier, as shown in Figure 3. Notably, the device uses a TTL-compatible three-wire [CLK, LOAD and DATA-IN (DI)] shift-registerlike interface. This feature saves CPU I/O lines and, in the case of remote

Figure 1—*Brute-force driving of traditional seven-segment LED displays requires a complete 8-bit parallel port plus another control line per digit and plenty of driver software.*



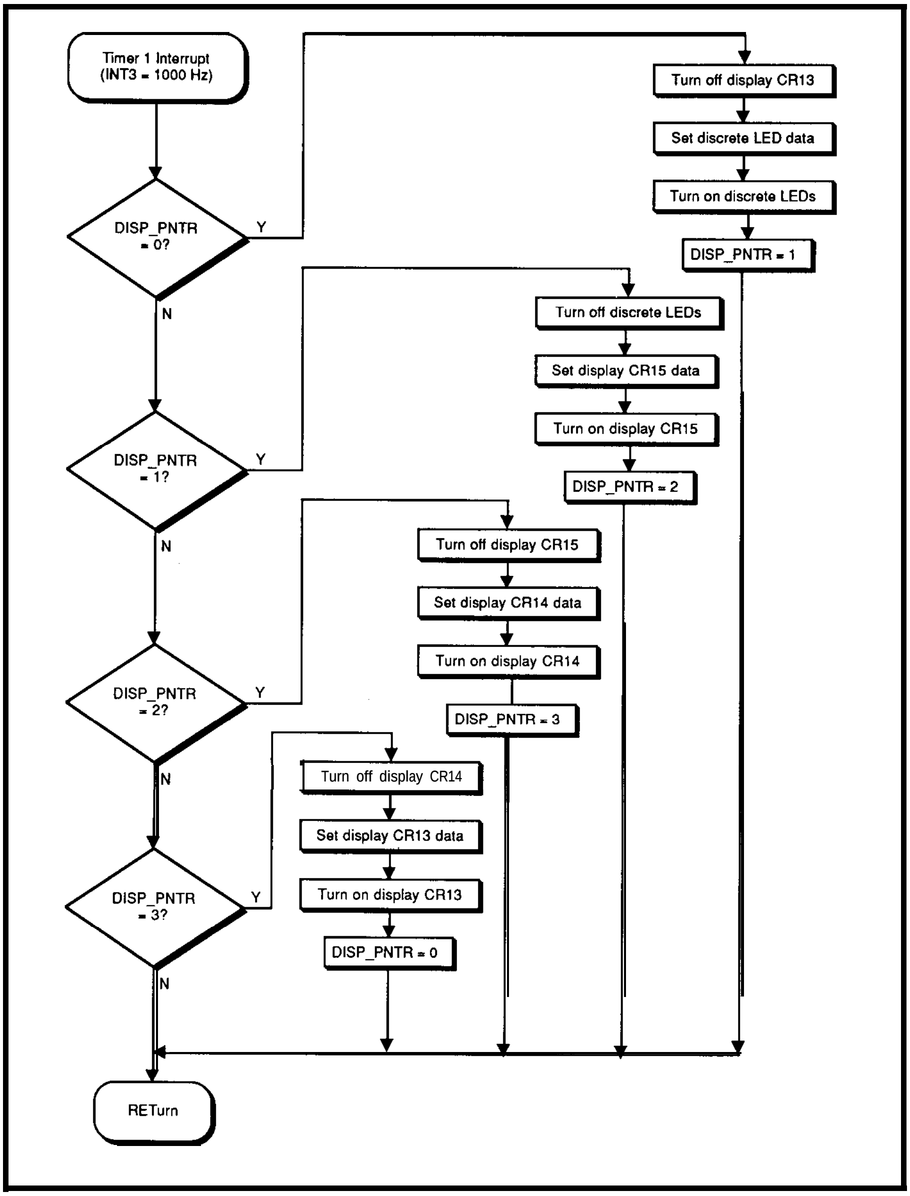


Figure 2—*The* interrupt *handler necessary to refresh the seven-segmenf LED displays in Figure 1 doesn't add a lot of overhead, but can be a nuisance during development.*

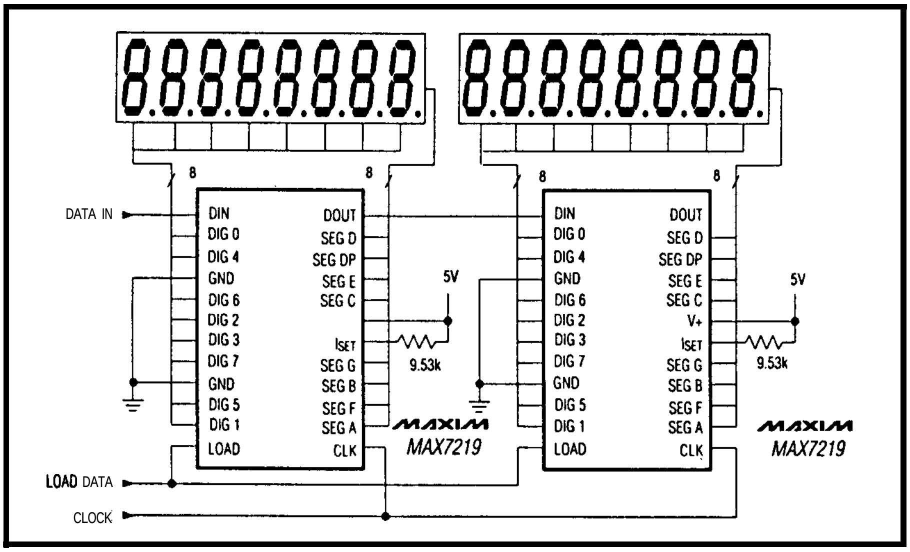


Figure 3—*The MAX7219* needs just *three* connections to *the host computer. Multiple chips may be* daisy chained to *handle an* unlimited *number of displays.*

mounting, cuts cable hassles. The shift-register approach also facilitates daisy chaining '7219s to handle more than eight displays.

Otherwise, all that's required is a resistor on the ISET pin to limit the segment drive current to match particular LED specs. Naturally, you can connect a potentiometer to ISET, providing manual brightness control. However, you may not need one because the '7219 includes on-chip 16-level brightness control using a driver current-controlling PWM.

The device also integrates a *CODE B* lookup table, eliminating the need for it in the driver software. CODE B is a 4-bit binary code, so it supports 16 different characters: "O-9," blank, minus sign, ***H, E,*** *L,* and *P* (I'm not kidding). However, it also supports a ***nondecode*** mode, which bypasses the table to allow direct control of individual segment drivers. A design combining the '72 19 with discrete LEDs (like the Kingtech 8 x 8 matrix shown in Photo 1) and nondecode mode could serve well as the basis for

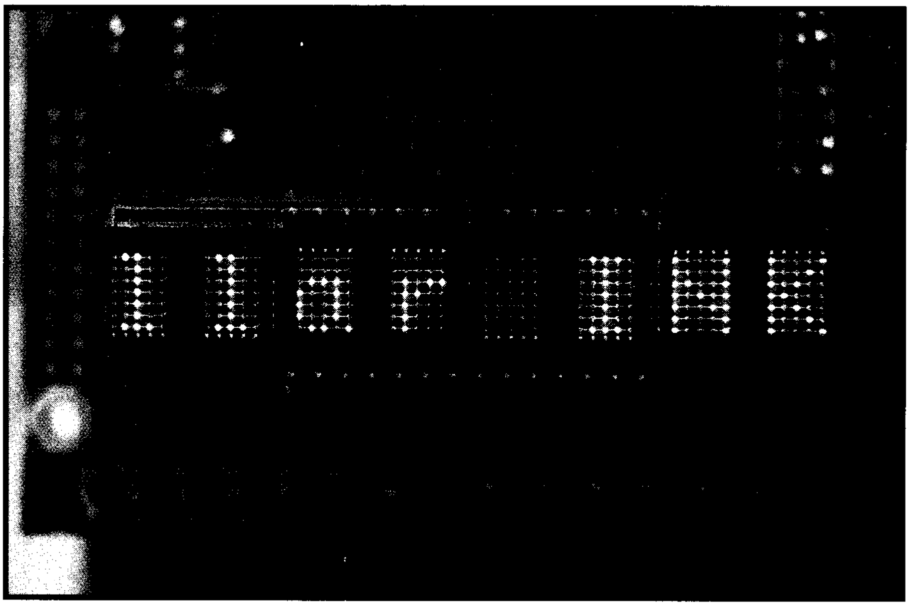


Photo 2—*The* ***Hewlett-Packard*** *HDSP-21* Ix is a *complete* eight-character, stackable, 5 x 7 dot *matrix, smart* LED display.

a variety of marquee-type displays.

YOUR NUMBER'S UP

The '7219 design is flexible when it comes to the number of displays supported, but you're still vulnerable to one final jerk around. Now, the big shots at GO_PUBLIC have decided they would like to have an alphanu meric display rather than just a numeric one.

Giving alpha applications a shot with seven segments is possible. For instance, hex digits are feasible if you

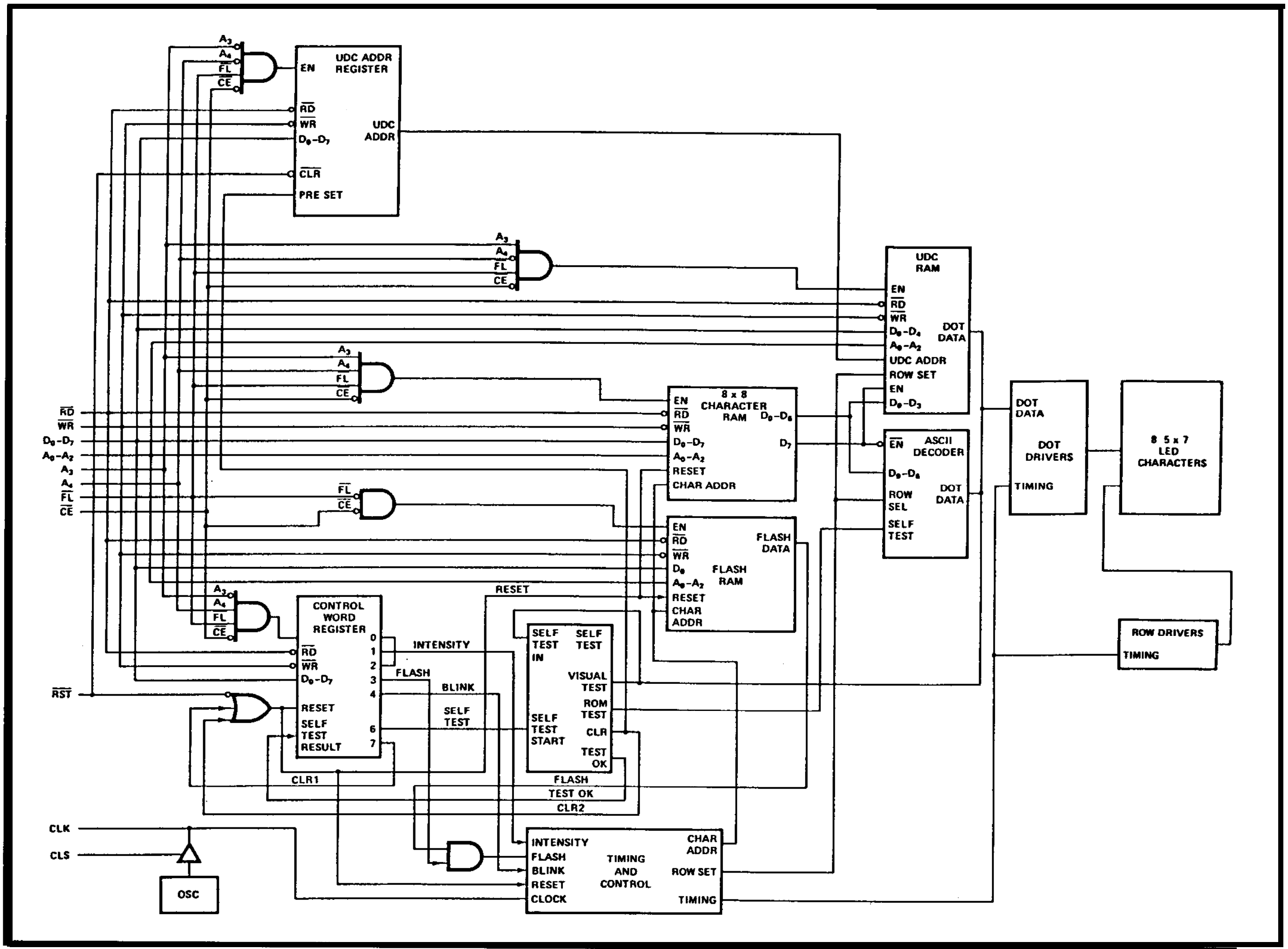


Figure 4—*The HDSP-211x includes much the same logic as the MAX7219, including refresh, decoder, programmable brightness, and more. One big difference is the display uses a parallel interface instead of a serial interface.*

can accept a rather tacky looking mix of upper and lower case (i.e., AbCdEF). However, many characters just aren't possible, notably those including diagonal strokes like *K, M, N, R, V, W,* X, Y, and Z. Furthermore, many of the other characters come out looking like numbers; for example, *S* = 5, G = 6, and of course 0 = 0. Looks like it's time to head back to the lab.

HP comes to the rescue with the HDSP-21 lx (see Photo 2), a complete eight-character, stackable, 5 x 7 dot matrix, smart LED display. The block diagram (see Figure 4) shows that the HDSP-21 lx includes much the same logic as the MAX72 19, including refresh, decoder (5 x 7 ASCII versus the '7219's seven-segment CODE B), programmable brightness, and so forth. One difference is that the HP display uses a parallel 8-bit interface versus the '7219's clocked-serial scheme. (Note that Maxim plans to offer the '7220, which is a '7219 with a parallel CPU interface.)

Of course the real claim to fame for the HDSP-21 lx is its capability to cram in 280 LEDs (eight characters x 35 LEDs each), and the whole gizmo is packaged with a standard 28-pin 0.6" DIP footprint. The '21 lx is available in all the usual LED colors including orange ('2110), yellow ('2111), red ('2112), and green ('2113).

Beyond the basics, the '21 lx also includes a couple of other key features. The *Flash RAM* shown in the block diagram consists of 8 bits, each of which enables and disables the blinking of an individual character. Meanwhile, the *UDC (User-Defined Character) RAM,* as its name implies, is a 16 x 5 x 7 bitmap memory that allows definition of up to 16 user-defined characters. Now, because you're attuned to the concept of *defensive design, you* realize that this feature will save you if the bosses find an overseas (non-ASCII-speaking) investor.

When using UDCs, heed HP's warning: "Illuminating all 35 dots at full brightness is not recommended." Indeed, they only guarantee full voltage and temperature range operation for 20 dots per character, which corresponds to the most dot-consuming ASCII character, "#." Depending on voltage and temperature, a full-brightness display composed of eight #s can consume from 1.5 to 2 watts. Turning on all 280 LEDs at full brightness would push power into the 34 watt range, more than likely leading to a terminal case of IC heatstroke.

LEDS WRAP IT UP'

You can finally relax because you have a design that's impervious to the number of characters, color, and font whims of your superiors. Or is it?

Sure enough, the boss comes rushing in and says they finally found a rich retiree with a big trust fund to invest in GO_PUBLIC. There's only one small problem-the guy's eyesight is about as good as Mr. Magoo's and he's griping that nobody will buy the "durn thing" if they can't read the display.

At this point, you might be ready to end it all, with a soldering gun no less. Never fear, simply plug in an HDSP-250x for characters 7 mm high (versus 5 mm for the '21 lx), which are readable in most conditions from a full 10 feet away. Other than the character size, everything else including the pinout, timing, and so forth is exactly the same.

The moral of the story is that sometimes the easiest way is the hard (as in hardware) way. ❑

Tom Cantrell holds a B.S. and M.B.A. from UCLA and has been in Silicon Valley for more than ten years working on chip, board ,and systems design and marketing. He can be reached at (510) 657-0264 or by fax at (510) 657-5441.

CONTACT

Maxim Integrated Products
120 San Gabriel Dr.
Sunnyvale, CA 94086
(408) 737-7600

Kingbright USA
225 Brea Canyon Rd.
City of Industry, CA 91789
(714) 468-0500

Hewlett-Packard Company
Inquiries
5301 Stevens Creek Blvd.
P.O. Box #58059
Santa Clara, CA 95052-8059
(800) 752-0900

419 Very Useful
420 Moderately Useful
421 Not Useful

Embedded Timepieces

EMBEDDED TECHNIQUES

John Dybowski

Since most embedded controllers deal with real-world events, a real-time clock is usually a necessity. There are lots of options available, with the hardware/software tradeoff usually deciding which one is used.

Sometimes embedded systems benefit from their capability to keep time. Often the capacity to do so is a requirement. As usual, you can keep time primarily through either a software-based approach or with hardware (implanting an electronic timepiece into the system in question). Through the use of a software-based clock, you can keep system overhead to an absolute minimum, an important consideration.

At the other end of the extreme, parts exist that contain all of the constituents required to implement full-function nonvolatile real-time clocks, including such functions as built-in crystals, backup power sources, and power-fail protection circuitry. In some cases, devices include useful functions such as nonvolatile memory controllers, nonvolatile memory blocks, or system watchdog capabilities.

A multitude of timekeeping devices are available today because a large number of manufacturers have jumped into the real-time business. These guys are good, and differentiating among the many options turns out to be one of the most difficult parts of the selection process. It's an engineer's market in which you can probably find exactly what you're looking for, but first you have to define your requirements.

CHEAP TIME

With all the real-time clock chips on the market, you may still find the need to play it cheap. Cheap usually means using a software-based approach because apparently the word's out that software doesn't cost anything.

Therefore, given the proper timebase, you can set up the required software timers to divide down the master clock successively into the elements that comprise real time as we count it. Of course, what you have may not be what you need, so you may have to do a little front-end work to provide a timebase frequency that divides down accurately to the values you're seeking. By using a standard 32.768-kHz watch crystal, a couple of CMOS inverters, a divide-by 32 counter, and several resistors and capacitors, you can build a 1024-Hz oscillator.

The resulting output of this circuit, pictured in Figure 1, yields a square wave with a period of 976 µs. Although this value may seem a bit strange at first, it turns out to be one of those righteous binary numbers that produces the desired outcome. Divide the period by 256 and the result is a solid 250-ms timebase, which you can further divide down using software to generate the ensuing larger counts needed. The 976-µs repetition rate of the clock is close enough to 1 ms to be suitable as a timebase for the system's interrupt-driven master timer, so it may be the only one the system requires.

Another approach that results in a highly accurate timebase suitable as a reference for counting real time is to use the 60-cycle component that rides upon the AC power often used to power the embedded device.

Here you can use an inexpensive optocoupler to isolate the incoming AC and translate it to a ground-referenced TTL signal. Clean up the coupler's output by putting it through a Schmitt trigger before routing it to an interrupt or input port pin. Although the frequency of the 60-Hz signal will have variations over the short term, the power company makes adjustments on the fly and the resulting accuracy ends up being very good over time. The resulting period of the 60-Hz signal is 16.66 ms. If you have an interrupt that runs faster than this rate, sampling the 60 Hz is easier than tying up a separate interrupt to count these pulses. As before, simply dividing this master signal in software results in a software real-time clock. Figure 2 depicts the circuitry needed to implement the 60-cycle front end.

The big disadvantage when using these software-based clocks is losing time if power is lost or the processor resets. This loss is usually a problem, but for some applications, especially those running on-line to a host computer, a fresh time download may be only a power-on message away. Using the crystal oscillator is a perfectly suitable low-cost solution for applications using uninterruptable power to power the system.

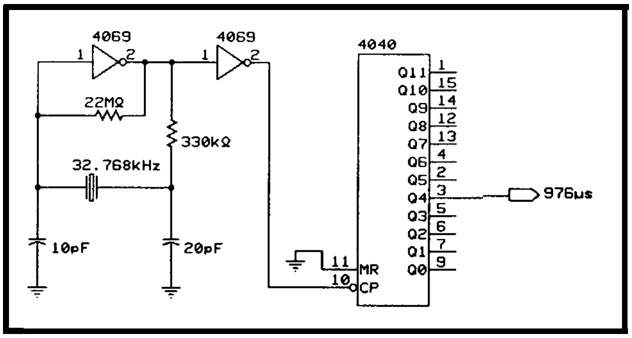


Figure 1— One of the simplest *methods for providing a system clock is to use a* standard *watch crystal and* divide *the frequency with a counter to obtain a* square *wave* with *a 976-µs period.*

SERIAL TIME

Dallas Semiconductor has a small serial clock that is inexpensive and stingy on the I/O pins. Packaged in an eight-pin DIP, the DS1202 provides a real-time clock/calendar and contains 24 bytes of static RAM. Three pins carry communications to the chip: a reset input, a clock input, and a bidirectional data line. You provide a 32.768-kHz crystal, and if you want nonvolatile operation, a couple of low-dropout diodes and a backup power source. Access is fast for a serial device, because the minimum clock's high and low times are 250 ns, which results in a maximum clock rate of 4 MHz.

It supports reading and writing the individual clock and RAM locations, along with a burst mode of access for both read and write operations to itself and RAM, in which the entire contents of the selected data region are transferable in a single-operation sequence. A write-protection capability is provided for added security as well. That the chip contains built-in power-fail detection circuitry, and access to the part relies on clocking the right bits at the right time along with the proper operation of the reset line, not to mention the additional write-protection feature, means that you don't need to add any additional circuitry to inhibit chip access when powering the system up and down.

However, the most impressive news is the clock uses only 300 nA at 2 volts, and this figure is a maximum, not typical! This feature is truly a marvel of CMOS design when you consider that not only is the device running a 32-kHz oscillator, but that a group of internal counters is continually operating as well. Operational current at 5 volts is only 1.2 mA. You might as well get rid of your fancy backup power schemes and just lash a lithium cell to it with this kind of power consumption.

Actually, I find that the amount of power consumed by this device alters the way I normally view the backup power source's function. Look at it this way. The chip essentially consumes no current; it just needs to see some voltage to be happy! If you consider the use of a common BR1225 lithium cell for backup power, you come to the realization that the DS1202's current drain does not appreciably affect the life of the cell.

Under the very light load that the DS1202 imposes on the lithium power source, the output voltage remains above 3 volts throughout the lifetime of the cell. However, load curves involving such light currents are seldom plotted due to the long times involved in taking these capacity measurements. Because lithium consumption is not really a consideration, the limiting factor turns out to be the time before the cell obviously physically disintegrates. With this part, you're set for the long haul. Figure 3 shows the simple interface requirements of the DS1202.

You can easily code support routines for the DS1202, especially when using processors with good bit-manipulation capabilities like the 805 1. First you need a couple of low-level routines to do byte transfers between the processor and the DS1202 (see Listing 1). The routines in Listing 2 give you access to individual registers within the DS1202. Finally, the chip supports a burst mode that allows you to quickly transfer the entire contents of the DS1202's register area.

One feature I like to see on a real-time clock is a timebase output, even on such a simple device as the DS1202, because for me it satisfies a system requirement. Unfortunately, the DS 1202 does not possess this function. The eight-pin DIP's designation of one pin as a no-connect makes this omission unforgivable.

If you're looking for a serial timepiece and want something with a somewhat more standard interface, consider hanging a Signetics PCF8573 off an I^2C bus where you can chain a number of worthwhile peripheral chips, all on a two-wire bus. Along with the usual clock/calendar registers, this 16-pin IC contains some other valuable functions as well. Included on the chip is a power-fail detect circuit, a once-per-second and once-per-minute pulse output, and a set of loadable alarm registers that can trigger an output pin when the clock/calendar registers match. You can

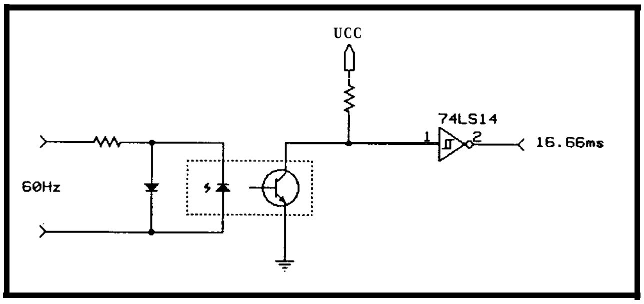


Figure *2-The AC power line is often an excellent timebase in embedded applications because the optoisolated interface is so simple.*

further specify whether to consider the date registers or not in the comparison by either setting or clearing an internal bit.

The PCF8583 provides most of the functions of the PCF8573, contains 256 bytes of RAM, and comes in an eight-pin DIP. Flexible alarm functions allow the selection of a number of output frequencies ranging from hundredths of a second to days on an open drain interrupt pin.

Where the DS 1202 is clearly designed to operate with a lithium backup power source, Signetics took a different approach with their I^2C clocks. With a typical backup current requirement of 3 µA and the capability to run down to 1 volt, a single NiCd cell provides backup power.

DELUXE TIMEKEEPERS

Although cheap is at times good, and a simple approach is preferable to a complex one, considering a more sophisticated tactic when evaluating peripheral functions for use in embedded designs is beneficial. This consideration is particularly advantageous when you can place multiple functions onto a single chip or, as in the case of real-time clocks, you can include typical support components in a single module with the timekeeper all set up and ready to roll.

For example, I usually favor a clock chip that includes an oscillator crystal deposited directly into the IC package. More than just a convenience, clocks that incorporate a built-in crystal possess the guaranteed oscillator accuracy in the AC operational specifications. Also, depending on your operation, the savings gained from ordering, inventorying, and stocking the required discrete components could be a significant factor. Select the required oscillator accuracy for your design and you can eliminate the need for manually tweaking trimcaps. Bench time can be expensive and a finger fumble could result in irate calls from the field and problems.

FIGURE 3

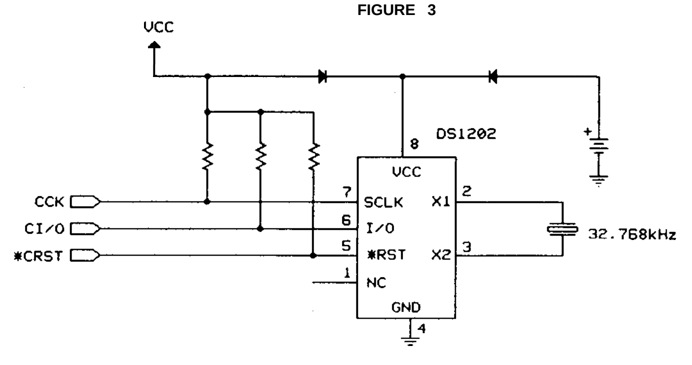


Figure 3—*Three I/O lines are all* **that's necessary to connect** *the DS1202 to an* **embedded circuit. A lithium** *battery and a* **handful** *of parts* **complete the design.** *Note that the* **diodes** *should be germanium.*

Finding completely self-contained real-time clock modules that have their backup **power** source encapsulated along with the crystal and other support components into DIP packages is now common. The advantages extend beyond convenience and the savings of board space. With the small load currents drawn from the lithium power sources by modern clock chips, the cell life in these applications is essentially the shelf life. Under these conditions, encapsulating the lithium cell in epoxy extends the cell's life expectancy.

All sorts of useful functions are now finding their way into real-time clock modules. Some of these include nonvolatile controllers for external RAMS, processor watchdog functions, meaningful amounts of resident nonvolatile memory, as well as power management functions. Although some of these parts do get rather pricey, considerations like limited printed circuit board real estate do come into play. General rule of thumb: if you can't get it in the box, you can't sell it.

If you're shopping for a real-time clock with RAM, a number of options exist. The Dallas DS1287, based on the MC146818 clock from Motorola, contains 50 bytes of nonvolatile RAM. This familiar timepiece has spawned a number of replicas from, among others, Benchmarq, SGS, and now even Motorola. The Benchmarq version, the BQ3287, contains 114 bytes of RAM instead of the usual 50, while retaining compatibility in all of the other areas.

Listing 1--**Support routines** *for the DS1202 can be very short, especially when* **using a** *processor with good bit-manipulation support like the 8051.*

```
;TRANSFER THE SELECTED BYTE FROM THE DS1202
;OUTPUT: ACC CONTAINS THE RECEIVED BYTE
;
READ_BYTE       PROC
                SETB     DATA-BIT        ;SET DATA BIT TO INPUT
                MOV      R0,#8           ;8 BITS TO READ
RB1:
                CLR      CLOCK-BIT       ;ASSERT CLOCK
                MOV      C,DATA_BIT      ;GET DATA BIT
                RRC      A               ; AND POSITION
                SETB     CLOCK-BIT       ;RELEASE CLOCK
                DJNZ     R0,RB1          ;JUMP UNTIL DONE
                RET
                ENDPROC

;TRANSFER THE SELECTED BYTE TO THE DS1202
;INPUT: ACC CONTAINS BYTE TO TRANSMIT
;
WRITE_BYTE      PROC
                MOV      R0,#8           ;8 BITS TO WRITE
WB1:
                CLR      CLOCK-BIT       ;IDLE THE CLOCK
                RRC      A               ;POSITION BIT
                MOV      DATA_BIT,C      ; AND OUTPUT
                SETB     CLOCK-BIT       ;CLOCK IN THE BIT
                DJNZ     R0,WB1          ;JUMP UNTIL DONE
                RET
                ENDPROC
```

Listing 2—*The individual registers in the DS1202 can be accessed using the low-level routines in Listing 1.*

```
;READ A BYTE FROM ONE OF THE DS1202 CLOCK REGISTERS
;INPUT: ACC CONTAINS ADDRESS TO READ
;OUTPUT: ACC CONTAINS SELECTED BYTE

RTC_READ    PROC
            SETB   RESET-BIT      ;RELEASE RESET
            RL     A              ;POSITION ADDRESS
            SETB   ACC.0          ;INDICATE A READ OPERATION
            SETB   ACC.7          ;MUST BE 1
            CLR    ACC.6          ;INDICATE CLOCK ACCESS
            CALL   WRITE-BYTE     ;INITIATE THE OPERATION
            CALL   READ-BYTE      ;GO READ
            CLR    RESET-BIT      ;ASSERT RESET
            RET
            ENDPROC
```

Motorola really left themselves open on this one. Although the MC146818 was fairly well thought out, at least for the time, the implementation left something to be desired. The MC 1468 18 consumed too much power, and the initial version really wasn't designed to be battery backed. The MC1468 18A corrected some problems, but other nagging difficulties remained. Because this clock eventually found its way into most PC/AT-style computers, others soon noticed the market potential and decided to make some improvements. Unfortunately, Motorola must not have paying much attention. Only recently has Motorola made any significant improvements to their initial design.

While 50 or 100 bytes of nonvolatile RAM may be adequate for maintaining some setup and configuration information, many applications require much more nonvolatile storage capacity. The question when selecting one of these real-time clocks with built-in RAM comes down to what is an acceptable means of access to the memory.

Although Dallas does have RAMified clocks providing RAM that is accessible in a standard fashion and suitable for use as general-purpose processor memory, some of the parts require some not-so-standard access mechanisms. Methods such as paged memory and the multistepped latching of address information are a couple examples of the games processors must play to get at the RAM in some of these products.

For example, the DS1387 contains 4K of RAM along with a DS1287-type clock in a pin-compatible package with the DS1287. The wrinkle is that apparently Dallas lashed the RAM to the clock, somewhat loosely, then added a couple of address latches and some control circuitry before burying the thing in epoxy.

What you get is the basic DS1287 core communicating to the outside world via its standard pins, with the RAM's data bus and address latches tied to the common data pins. The control signals for the two address latches, along with the RAM's read and write lines, tie to some previously unused pins. You must provide three separate write strobes that qualify with chip selects for the address latches and the RAM write signal as well as a chip select that qualifies the RAM read signal. To read from or write to the RAM, write the lower eight address bits, write the upper four address bits, then do the actual read or write operation. Figure 4 shows a way to interface the DS1387 to a processor using I/O mapping.

Getting at the RAM isn't very complicated, but you do need a multistepped approach. The code in Listing 3 illustrates the method I applied using an 80188 processor with the DS1387's RAM.

First, I provide a routine to latch the 12-bit address. In the second routine, the data byte must also be strobed into the data register. Finally, combine the routines to do something useful.

PHANTOMS

Phantom clocks, as the name implies, coexist in the same address space as other memory devices such as a PROM or a RAM. These serial devices often incorporate power monitoring and control circuits and backup power sources that can nonvolatize collocated static RAM chips.

Parts like the Dallas DS 12 16 operate as a socket residing in a memory site on the printed circuit board into which a memory device is plugged. For the most part, these clocks remain disabled and stay out of the way while passing the incoming chip selects and data directly to the piggy-backed memory device. To seize control of the clock, issue a serial 64-bit key to unlock it.

Proceed by initially issuing a read cycle to any memory location to start the internal pattern recognition sequence. This action initializes a pointer to the first bit of the 64-bit comparison register within the phantom device. Next, 64 consecutive write cycles must be executed, which can be to any location (usually a scratch location in the memory space is reserved for this purpose) in order to establish communications with the clock. With a correct match, chip selects to the memory device are blocked and the clock is enabled. Then, data can be transferred in 64-bit streams to and from the clock.

Although great for retrofitting existing designs, phantom clocks do have their drawbacks. Depending on how the memory is allocated in the system, disabling interrupt processing throughout the entire access phase to the clock may be necessary. This step might prove troublesome in many real-time applications, because not only do you have to establish the 64-bit key prior to communicating with the clock, but the subsequent 64-bit data transfer assumes that all communications to the socket are intended for the clock and not the memory device. As usual, what is perfectly acceptable for one application might be totally unworkable in another.

REAL TIME

As you know, many embedded designs incorporate real-time clocks. Often these clocks are present only in order to tell time. At times, the clock chip is used to display the date and time on a local display device or to timestamp data records before they are placed in memory for later processing. For these operations, the values dispensed by real-time clocks are just what you need. However, some activities suffer from the ambiguity inherent in the notation commonly used by real-time clocks. This ambiguity results from the cyclical operation of the counters that make up the real-time clock, making relating events that occur in linear time to the time kept by a real-time clock somewhat difficult.

This difficulty becomes a problem when scheduling events that occur at precise intervals for specified periods of time is necessary. For example, consider a system programmed to control various lighting systems and alarms based on a downloadable schedule. In such a system, responding to the system sensors differently depending on the time of day and day of week may be desirable. Furthermore, you may want to modify the system's response on holidays. You end up with a strategy that perhaps would ignore the motion-detecting sensors during normal daylight hours then switch to turning on lighting systems in response to early evening activity. After a certain time, activating the sensors would raise an alarm condition.

There is a problem that becomes obvious when you begin coding for all the circumstances likely to be encountered. The familiar cyclical character of timekeeping causes a lot of trouble when you try to coerce your controller into making reason out of the ranges of time that cross 24-hour boundaries, as well as the boundaries between days, months, and years. The situation becomes more complicated when you need to allow for periods focused around some specific point in time.

For instance, you may have to permit activities such as entry onto the premises around certain time, say perhaps a period of ±15 minutes around 5:00 P.M. Things get a little tricky when the time is centered around 00:00.

Although you can code around these special cases, doing so makes for a lot of special processing. Generally, making things as simple as possible is best, but when dealing with any nontrivial matters, there's a limit to what you can attain simply. The next best thing is to encapsulate the complexity, get it thoroughly tested, and then can it. Then you can get on with the business of coding your mainline cleanly without regard for all the special cases. Of course, this step involves the selection of the overall design approach early in the design cycle.

You can take a couple simple steps to relieve some of the ambiguity inherent in the units issued by real-time clocks. For example, set the clock to operate in 24-hour mode, and you have a linear span that counts the hours of a day. Counting the date using Julian units of measure, in which the days of the year are numbered sequentially, can also help to alleviate some of the vagueness related to counting months and days. Although a step in the right direction, you still have to consider the boundary conditions that still exist when the days and date wrap.

MACHINE TIME

This idea of Julian time seems right, and there is no reason why you can't extend this principal to obtain the linear reckoning needed, but you must go further to convert the date from the real-time clock to an entirely linear format. You can make this conversion easily using either a combination of lookup tables and arithmetic or just the tables alone. Select the approach that best suits your processor's capabilities.

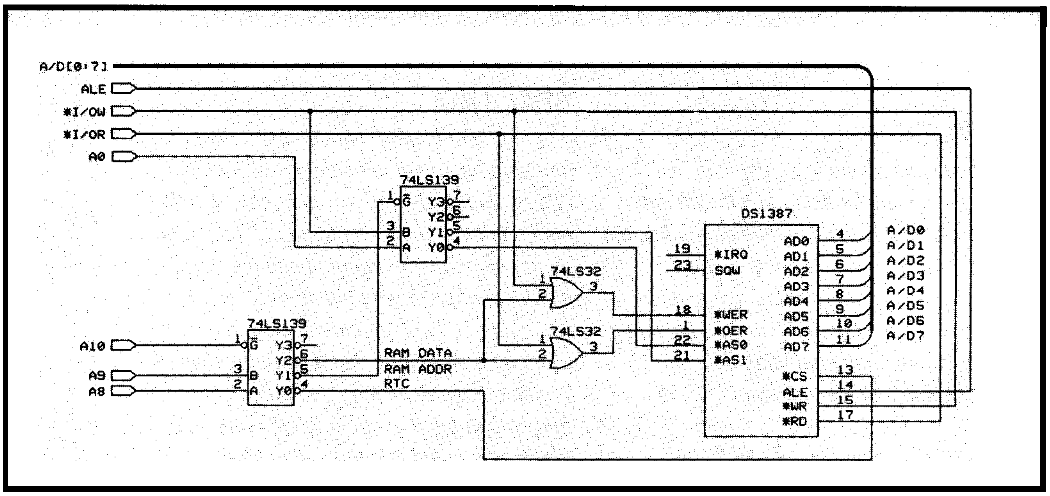


Figure 4—*The DS1387 clock module may be connected directly to the processor's bus. You must provide three separate write strobes qualified with chip selects for the address latches and RAM write signal and a chip select that qualifies the RAM read signal.*

Listing 3—*The DS1387 contains RAM that may be used by application programs, but accessing it requires a multistep approach (80188 code shown).*

```
;WRITE ADDRESS WORD BYTE TO DS1387'S ADDRESS LATCHES
;ADDRESS DECODING IS ASSUMED TO BE CONTIGUOUS FOR LOW/HIGH BYTES

;INPUT: ADDRESS IS IN BX
;
ADDRESS_WR      MOV     AX,BX           ;POSITION ADDRESS
                MOV     DX,#NRAM_ADD    ;SELECT ADDRESS LATCH
                OUT     DX,AX           ;LATCH THE 12 BITS
                RET

;WRITE DATA BYTE TO DS1387'S RAM AREA
;ADDRESS MUST BE ALREADY LATCHED AT THIS TIME

;INPUT: DATA BYTE IS IN AL
;
DATA_WR         MOV     DX,#RAM_DATA    ;SELECT DATA REGISTER
                OUT     DX,AL
                RET

;WRITE A STRING TO THE DS1387'S RAM AREA

;INPUT: SOURCE POINTER IS IN DS:SI
        BYTE COUNT IS IN CX
        DESTINATION POINTER IS IN BX

;OUTPUT: AX CONTAINS STANDARD COMPLETION CODE

STRING_WR
                JCXZ    SW2              LEAVE IF NULL STRING
                PUSH    DX
SW1
                CALL    ADDRESS_WR       LATCH THE ADDRESS
                INC     BX               NEXT DESTINATION ADDRESS
                LODSB
                CALL    DATA_WR          WRITE THE DATA
                LOOP    SW1              LOOP UNTIL DONE

                POP     DX
                MOV     AX,#0           ;INDICATE NORMAL COMPLETION
                RET
SW2
                MOV     AX,#-1          ;FUNCTION ABORTED
                RET
```

Regardless of the approach, the logic to this conversion proceeds as follows:

- Determine if you are in a leap year, then convert the month of year to days.
- Once you have this number, add the day of month to arrive at the Julian date.
- Don't stop now. If you count the number of minutes per day, you arrive at the number 1440. You can easily compute the corresponding number of minutes into the year, the Julian minutes. However, the numbers do get large, so limiting the range for practical purposes may be advantageous.

Armed with this knowledge, you may structure a schedule in two ways. First, pick an absolute starting point for a basis, and reference all time using Julian minutes, but, as I've said, the numbers get big pretty fast. Using a second approach, select some arbitrary time for a reference point and then count minutes.

Obviously the advantage of the second method allows you to adjust and reuse the schedule by simply redefining a new starting point. You also can count over 45 days' worth of minutes using a 16-bit integer to schedule one month's activity without overly burdening an 8-bit processor. With this scheme, a typical schedule might consist of start and stop times comprising 16-bit values each. Alternatively, you could define a point in time using a 16-bit value with the second value denoting a range centered around the time.

Whether the system accepts a schedule download in this format or if the conversion from standard time notation is performed locally, the memory storage requirements are greatly reduced from what would be required if storing conventional time. This consideration could be an important one because it would free up valuable RAM for other purposes.

As I mentioned, there are advantages to having canned routines that perform dedicated functions such as time conversions. Once you have these routines running, they perform functions that you perhaps did not have in mind originally. For example, you could use the conversion routine to calculate the elapsed time of events. Using this method, you would begin by recording the starting time of the event. When the end of the event is detected, this time is used to calculate the number of elapsed minutes directly, using the stand algorithm, without regard to any boundary crossings. Subsequent conversion to standard units of measure would be performed next. An example of the useful fallout of having canned the complexity in the first place. ❑

John Dybowski is an engineer involved in the design and manufacture of hardware and software for industrial data collection and communications equipment.

IRS

422 Very Useful
423 Moderately Useful
424 Not Useful

Let Me Explain

As the information age continues, CD-ROM technology has made a wealth of data available to desktop computer users. Russ introduces us to several patent databases and sets the stage for his new column.

PATENT TALK

Russ Reiss

I have found the readers of the *Computer Applications Journal* to be some of the most creative designers anywhere. However, even the most creative people can become stagnant, and seeing how someone else solved a problem can stimulate the flow of ideas. This creativity boost is one of the reasons why patents (and this magazine) exist.

Beginning this month, I will be bringing you abstracts and other interesting tidbits related to actual patents issued by the United States Patent Office. Some will be "hot off the press," having been issued just months before I bring them to your attention. Others will be "golden oldies" that have a particular interest or value. Before I discuss a specific patent, let me explain how and why patents are awarded and where I get my information.

In order to receive a patent for an invention, you must disclose fully the techniques you used. You must make this disclosure with enough detail that one "skilled in the art" can reproduce the invention. In exchange for this information, you receive exclusive rights to use, sell, or license your invention for a specific period of time.

Clearly, full disclosure is necessary in order to determine the validity and uniqueness of a proposed patent, but public disclosure of the patent serves many other purposes. It informs those who might be interested in licensing this patent of its existence. It notifies those with similar ideas that a patent has already been issued. Public disclosure also serves to educate designers and inventors as to what the state of the art is in any given area. Besides stimulating new approaches that ultimately lead to new ideas, which are the precursors of new industry, jobs, and a healthy economy, inventors benefit from this practice because it prevents duplicate efforts.

To a great extent, these laudable goals remain unachieved. Unless you belong to a large organization that can afford a dedicated patent research staff, this system probably has not helped you. How can the individual or small company keep up to date on patents? Certainly reading the Patent Office's official *Gazette* incessantly and waiting weeks or months for copies of pertinent patents that you sent away for is not very practical.

Once again computer technology comes to the rescue; this time in the form of CD-ROM drives and powerful database and search software. Now the entire United States Patent Office database is at your fingertips.

PATENT DATA ON CD-ROM

Until recently the variety and quality of published information databases have not kept pace with the ever-falling prices (and access times) of CD-ROM drives. One reason for this situation is the degree of effort required to produce a CD-ROM title. Plus, a task never ends when it involves dynamic data; each week new information must be entered and new CDs produced, which is certainly the case with Patent Office data.

A few years ago I purchased the CASSIS database, which the Patent Office issues and some CD-ROM clearinghouses may resell. CASSIS's biggest drawback is it is obsolete at publication (the Patent Office produces new CDs only quarterly). However, the cost and size are right; a single disc containing the data costs about $200.

The CASSIS database only provides patent abstracts. Even this abbreviated form of information would be adequate if it was complete, but it is not. Abstracts exist in this database for only the most recent patents, meaning those granted since two years prior to publication. Patents older than these are simply listed by title, author,

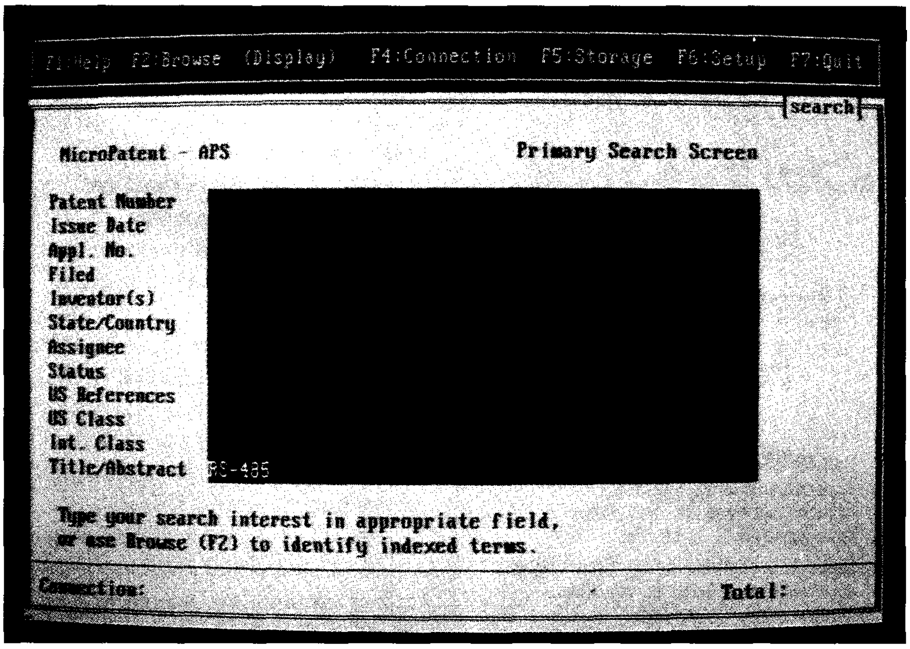


Photo 1--The *Automated Patent Searching software allows the user to search for patents based on a variety of fields.*

classification, and so forth. This resource is a useful and affordable one, but it really falls short of full patent-searching capability.

Another patent database and search system is the APS (Automated Patent Searching) system from Micro-Patent of New Haven, Conn. While APS is their "low end of the line" product, it is quite impressive. Unlike CASSIS, the APS's CDs contain *complete* abstracts of every patent. Each month a subscriber receives a new CD with all patents for the year up through those issued a week or two before publication. At the end of the year, you keep the December install-ment, containing the entire year. Prior years are available (about three years to a CD), beginning with 1975.

MicroPatent also offers more advanced [and costly) patent databases. The most complete database is PatentImages, containing complete copies of each patent, including drawings. You can view them on high-resolution, full-page monitors, and print out hard copies that include the drawings using laser printers.

This system is not the best for individual and small-business users for three reasons. First, MicroPatent sends you a multitude of CDs; you receive about two new ones per week for an approximate total of 100 a year. With that number, having the right disc on-line when you need it is unlikely, even with an expensive jukebox-type CD changer. Second, the cost is prohibitive ($5,500 a year for a subscription). Finally, the amount of information provided is limited. Only patents from 1990, 1991, and 1992 are currently available. PatentImages would be a bargain for a company requiring instant access to many recent patents; however, for the reasons I've stated, I've chosen to use the APS database.

THE ESSENCE OF A PATENT ABSTRACT

So patent abstracts are available conveniently on CD-ROM, and knowing about them will increase the flow of my creative juices-what does an abstract look like, anyway?

To best illustrate the features of a patent abstract, I'll describe a simple search using APS. Say that I'm curious what patents exist regarding the RS-485 serial bus protocol. I might simply enter into the APS search screen the search parameter RS-485 as is shown in Photo 1.

In a couple of seconds (using a Hitachi CDR3600 350-ms CD drive on a '486/33 PC), the number "1" appears next to the request, showing that only *one* patent exists in the database having exactly the text "RS-485." (In

Patent Number	4,728,754
Issue Date	1988 03 01
Appt. No.	902246
Filed	1986 08 29
Inventor(s)	Fowler, Glenville C.; Tozer, David J.
State/Country	GBX
Assignee	Plessey Overseas Limited
US References	3,597,549 4,292,623 4,570,257
US Class	178/2C 178/2D
Int. Class	H04Q 3/00
Title	Interbus system

Abstract The interbus system provides the communication intercon-nection between communication systems to form a large telegraph packet switching exchange for example. The fast interbus system uses an RS-485 interface, however, the specific protocol used has particular speed advantages. The interbus system consists of a bus converter for each communication system intercon-nected by a bus involving 16 data bits, a pair of parity bits, clock, abort, transmit active, address strobe and grant return paths. The bus converters are chained by a grant link arrangement. The grant link is used to carry a grant pulse which is passed on around the chained bus converters to identify the master converter for an interbus system transfer. When the grant pulse is received by a bus converter wanting to transmit, that bus converter breaks the chain and performs the transmission required. When the transmis-sion is complete the completing bus converter surrenders its master state by generating the grant pulse.

Figure 1--Doing a *search on "RS-485" yields but a single patent. In this case, the patent deals with a token-passing bus system that happens to use RS-485 for the physical connections.*

fact, without the hyphen, ***no*** patents match this request.) Now, let me describe the abstract. Pressing F5 flashes the actual abstract itself on the screen, as is shown in Figure 1.

Notice that more than just the text of the abstract appears. At the top is pertinent information about the patent, which includes the patent number, date of issue, date of filing, the name of the inventor(s), and the ***assignee*** (usually the company that sponsored the inventor). Also listed are references cited by the author to related or similar patents and classification information.

Most of this setup is self-explanatory, but a couple of points require further explanation. Assigning the patent to the sponsoring company is a common employment requirement for engineers working for many companies. Then there are the classification codes, which are very important. If you wanted to "do diligence" in applying for a patent of your own, you would search for other patents in these and related classification areas. Here's what the codes stand for (drawn this time from the CASSIS database):

178—Telegraphy
178/2C—Way stations; party lines
178/2D—Round-robins

You could perform a search on these classifications to locate all pertinent patents in these fields. This process might be restricted by other search parameters such as date, assignee, and so forth.

Note also the origin of the patent, ***GBX (see*** Figure 1). If the patent had originated within the USA, a two-letter state code would appear. In this case it came from a foreign country (Great Britain) and these appear as two-letter country identifiers followed by the letter X.

By the way, a quick search shows that there are some 7,853 patents from Great Britain in the current 1990-1992 database, while there are 58,391 from Japan, and only two from British Virgin Islands. Interestingly, these design patents apply to an expandable valise and a barbecue grill. Such are the concerns of an island paradise! Of course these numbers include *total* patents on *all* subjects, not just on RS-485. Note also that those 58 *thousand* patents were issued to Japan in just a little over two and a half years!

Generally though, your initial interest is in the abstract itself. The inventor [or his patent attorney) writes the abstract, representing the assignee's summary of the patent's description. If what you see looks worthwhile, you should then go on to read the actual body of the patent. While this information is not available on the APS CD-ROM, you can write to the patent office for a copy of the

4,136,803	Method and mechanism for conversion of free access ice chests to pr
4,324,325	Apparatus for collection of metallic containers and method therefor
4,439,856	Bimodal bus accessing system
4,464,749	Bidirectional token flow system
4,488,232	Self-adjusting, distributed control, access method for a multiplexe
4,494,233	Method and apparatus for the detection and regeneration of a lost t
4511,958	Common bus access system using plural configuration tables for fail
4,551,721	Method for initializing a token-passing local-area network
4,553,234	Method and system of transmitting digital information in a transmis
4,556,974	Method for passing a token in a local-area network
4,571,675	Microprocessor device with integrated auto-loaded timer
4,577,060	Paging network
4,594,724	Bypass method and apparatus for debouncing a mechanical switching el
4,602,365	Multitoken, multichannel single bus network
4,607,256	Plant management system
4,641,308	Method of internal self-test of microprocessor using microcode
4,646,232	Microprocessor with integrated CPU, RAM, timer, bus arbiter data fo
4,649,535	Method and apparatus for maintaining a dynamic logical ring in a to
4,654,889	Multistar fiber optic network with improved access time
4,663,748	Local area network
4,667,323	Industrialized token passing network
4,680,581	Local area network special function frames
4,682,326	Method and apparatus for maintaining a dynamic logical ring in a to
4,707,830	Token passing LAN using a plurality of tokens
4,710,915	Loop transmission system having automatic loop configuration contro
4,725,834	Reliable broadcast protocol for a token passing bus network
4,731,880	Communication network
4,739,183	Local area network for vehicle
4,745,598	Method and apparatus for maintaining a dynamic logical ring in a to
4,747,100	Token passing network utilizing active node table
4,766,530	Token passing scheme for a predetermined configuration local area n
4,777,591	Microprocessor with integrated CPU, RAM, timer, and bus arbiter for
4,785,448	System for communicating digital data on a standard office telephon
4,785,449	Network system for data transmission among plural communications st
4,789,982	Method for implementing a token passing ring network on a bus netwo
4,799,052	Method for communicating data on a communication network by token p
4,817,082	Crosspoint switching system using control rings with fast token cir
4,819,229	Local area network priority control system
4,825,204	Address setting and data transmission system
4,845,609	Computer communications subsystem using an embedded token-passing n
4,866,704	Fiber-optic voice/data network
4,866,706	Token-passing local area network with improved throughput
4,891,639	Monitoring system of network
4,893,821	Scoring system for game apparatus
4,930,121	Network system using token-passing bus with multiple priority level
4,949,337	Token passing communication network including a node which maintain
4,964,120	Method of detecting a cable fault and switching to a redundant cabl
4,979,748	Token aligning three-dimensional strategy game and method of play
4,983,010	System for connecting light waveguide buses and coaxial cable buses
5,012,468	Master slave industrial token passing network
5,042,031	Double loop communications control system
5,048,014	Dynamic network reconfiguration technique for directed-token expand
5,070,501	Token passing type slotted loop network system with means for enabl
5,081,623	Communication network
5,093,828	Method of exchanging data
5,097,469	Passive monitor for broadcast communication network
5,115,437	Internal test circuitry for integrated circuits using token passing
5,119,378	Testing of integrated circuits including internal test circuitry an
5,121,919	Game playing device
5,123,107	Topography of CMOS microcomputer integrated circuit chip including
5,125,492	Token operated television timer
5,140,586	Token associated data network communications protocol

Figure 2—*Doing a simple search using the keyword string 'Token passing*" yields 62 patents. Note that some line: are truncated in this summary listing.*

patent. (Mail $3 per patent to Commissioner of Trademarks and Patents, U.S. Patent & Trademark Office, Washington, DC 2023 1.) Also, many universities are "depositories" of patents where you may view and obtain copies of them.

INTERPRETING THE PATENT

Certainly no one single patent involves the use of the RS-485 bus, but this example is the only one in which the author called out the text "RS-485" precisely. Reading the abstract, you can see that the topic of this invention is actually a token-passing bus system. If you were to do a further search on the keywords *token pass* * (note, wildcard identifiers are permissible), you would find a large number of matches. Among these are also some patents referring to board games, ice chests, and coin (token)-operated machines.

In this case, the wildcard search was too broad. Note that some **12** fields may be used to qualify the search (see in Photo 1). Simply using the search criteria *token passing* in the Title/Abstract field, you find some 62 patents match. Figure 2 presents the listing of these patent titles. With a few keystrokes you may call up the abstract of any one of particular interest or save them all to disk for printing or later reading.

You can see from the listing of token-passing patents that most apply to the topic you had in mind, but a couple still have to do with ice chests. They use *tokens* to activate a mechanism, and the word *passing* is somewhere in the abstract. You could have eliminated these possibilities by forcing the juxtaposition of *token* and *pass* * with no spaces between. Placing the text in quotes accomplishes this goal. Finally, say you're interested only in patents assigned to Plessey. Using these criteria, you'll find 113 patents in the 1990-1992 database for all divisions of that company.

I'm sure you get the idea of how powerful and flexible a tool this patent database is. In the future, I hope to bring you a collection of interesting and thought provoking patent abstracts, and I solicit your input as to what you'd like to see. Until next time, put your thinking cap on and come up with some *Patent Talk.* ❑

Russ Reiss holds a Ph.D. in EE/CS and has been active in electronics for over 25 years as industry consultant, designer, college professor, entrepreneur, and company president. Using microprocessors since their inception, he has incorporated them into scores of custom devices and new products. He may be reached on the Circuit Cellar BBS or on CompuServe as 70054,1663.

SOURCE

Patent abstracts appearing in this column are from the Automated Patent Searching (APS) database from MicroPatent, 25 Science Park, New Haven, CT 06511, (203) 786-5500, (800) 648-6787.

I R S

425 Very Useful
426 Moderately Useful
427 Not Useful

CONNECTIME conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(203) 871-1988—Four incoming lines
Vernon, Connecticut

If you avoid calling the Circuit Cellar BBS because you fear long-distance phone bills, we've made some improvements that may ease that fear.

First, we now have three V.32bis 14.4-kbps modems to make your connections f/y. These work with a// the latest speed, data compression, and error correction protocols (V.32, V.32bis, V.42, V.42bis, MNP5, etc.). Second, we now support the popular QWK message compression format. You should now be able to call the BBS, download a week's worth of messages, and hang up in less than five minutes (assuming you connect at 14.4 kbps). Uploading your replies should fake even less time. We have complete details on-line, so splurge on one call and download all the pieces to make your future calls very short indeed.

In the second of two discussions this month, we'll be taking another look at sensing AC current.

In many controller designs, the digital circuitry is the easy part. Making reliable real-world interfaces that can survive the real world and properly isolate the processor is the tricky part. The first discussion is just a small excerpt from a much larger thread in which many people chimed in with many suggestions. I found the snubber portion to be most interesting, so am including just that part. The solution to Ron's problem turned out to be driver chips that were undersized for the relays. Rep/acing them with higher-voltage parts solved the problem, but if still didn't negate the need for snubbers.

Msg#:64343
From: RONALD HORNER To: ALL USERS

I have developed a small Z80 controller with relay outputs. Everything works fine except when the relays are switching a load. The relays are rated for 5 A at 120 V. When they turn a load (such as a 120-V light bulb] on or off, the computer hangs up. More analysis revealed that the program in RAM becomes scrambled. The relays do not cause any problem when there is no load.

I have the relays (PC-type solder relays) mounted on the same board as the microprocessor and its associated components.

Question: when the contacts open and close, would they be emitting RFI that is strong enough to affect the TTL bus, resulting in a computer crash? Are there standard shielding requirements for this application? (I have seen other boards with relays that don't appear to be shielded?) Is there anyone out there who is familiar with these kind of conditions? Any books to recommend?

Msg#:64351
From: STEVE CIARCIA To: RONALD HORNER

Your problem is RFI. You are radiating trash that your computer is picking up. Make sure there is a suppression diode across the relay coil and you'll probably need snubber networks across the loads if the problem occurs mostly when the loads are attached.

Msg#:64848
From: RUSS REISS To: STEVE CIARCIA

I agree with you regarding the snubbers, but have you ever seen a GOOD reference on how to calculate their value? I've used hit-and-miss a good deal myself. I wonder if inexperienced people don't use them a lot because they don't know WHAT to use.

Basically, we're talking about a series RC network across the relay contacts to absorb the HF energy during the switching transient, and the associated arcing. Luckily, the AC frequency is low enough that the capacitive reactance is high at 60 Hz. But it is very low at the switching/noise/transient/spark frequency (hundreds to thousands of kHz?). The resistor is there to absorb the energy and avoid a dead short across the contacts (at high frequency). This is my understanding, with appropriate question-marks where I'm lacking. I'd imagine classical transient analysis would work if/when we are talking about DC circuits, and perhaps "filter analysis" for AC? But one really doesn't have a good handle on the "frequency" of the garbage being filtered out.

Msg#:64875
From: STEVE CIARCIA To: RUSS REISS

I agree that there must be a real way to calculate the values, but given the imprecise impedance usually connected to the contacts anyway, it's hardly worth the exercise. This is one case were empiric choice wins over calculated value. I generally use 27–47-ohm I-watt resistors and 0.1–0.33-µF caps.

Msg#:64880
From: RUSS REISS To: STEVE CIARCIA

I can recall a project in which we designed a micro-based ATM machine. It worked fine until someone switched on the auxiliary fluorescent light (which had nothing to do with the microprocessor system but just backlit an ad board). Actually, I think it was when the lamp turned off, the system would reset every single time!

We checked the wiring path, and it was as far from the signal and power leads as we could make it. Everybody was stumped! I ran to the parts drawer and pulled out a big kludgy old 400-V capacitor (scientifically determined by whatever was close to my fingers in the 0.1–0.5-µF section, and a big resistor-forget the value, maybe 100 ohms?). Twisted their leads together, crimped it across the lamp switch with pliers, and tested. Yup, all noise gone! No more reset, but many wide eyes and open mouths. They had been muttering things about spending lots of money on a better logic analyzer or storage scope or something. Hehehehe. It's nice to wear the Superman suit once in a while!

Msg#:64938

From: ED **NISLEY** To: STEVE **CIARCIA**

The dope from *The Art of Electronics* is that they like 100-ohm/0. 1 -µF snubbers for AC line work, but don't go into any design details: "The values shown are typical for small inductive loads driven from the AC power line."

They suggest using MOVs or Transorbs, which might actually be a better deal given that Digi-Key carries 'em for half a buck in onesies. The clamping voltage is well below the "spark" level, so I think that would take care of all the radiated/conducted hash: no arc, no hash.

Msg#:65087

From: PELLERVO **KASKINEN** To: RUSS **REISS**

Actually, Russ, the principles for defining a suitable snubber are not too strange. Why they are not more explicitly stated is due to the fact that the user does not normally know what his load is and what he/she would allow for the snubbered situation. Here are a few pointers..

A snubber can be used for a DC as well as for an AC circuit. If we are talking about AC (or repetitive switching of DC) we want to make sure that the steady-state current does not overheat the resistor. Typically the capacitor is limiting the current at the low frequency and high voltage, so the larger the capacitor the more power is generated. With this basic current limiting scheme, a lower resistance reduces the power dissipation. But why not reduce it to zero? Because you have to dissipate the inductive energy SOMEWHERE. Otherwise you just cause a ringing that has maybe double the original voltage as a peak value. A small capacitor value causes lower power dissipation in the resistor, but does not have the muscle to take the inductive energy in and keep the voltage (and frequency) low. On the other hand, a large capacitor, especially in situations where you have the snubber over a relay contact and control AC load, will let some amount of current flow through the load that supposedly is OFF-hardly desirable in every case.

Let's take a simple example. Assume we have a 60-V transistor used to switch a 100-mA, 24-V inductive load. The load obviously has a 240-ohm resistance (24/0.1 V/A). But how often do you know its inductance? For the sake of easy numbers, let it be 2.4 mH, producing a time constant of 0.0024/240 H/ohm or 10 µs. Now, we want to protect the transistor switch (and at this point I want to note that a diode protection slows down the decay of the load current by free-wheeling) so we want to do something with a snubber. How about a capacitor directly over the transistor? The energy in the coil (ignoring the internal resistances for a moment] will be transferred to the capacitor so $L\times I\times I/2 = C\times U\times U/2$. All you need to do is decide how much voltage your transistor will allow (48 V?), being a 60-V transistor to get the approximate value for the capacitance. Looks to me like 10 nanofarads. The resonant frequency would then be about 32 kHz and the "Q" factor about 2.

Another problem before I let go: The transistor also has a limited current capability. Pick 0.5 A as reasonable for our 0.1 -A load. So, what happens at time of turning the load on? The capacitor has 24 V on it, just like the transistor. Without any limiting, the initial current *might* be detrimental to the transistor (not really, with 10 nF and 24 V, but anyway), so we want to limit the current as said to 0.5 A by adding a 48-ohm resistor to the capacitor. Now, the voltage at the transistor during the turn-off will be increased by the amount of voltage drop in this added resistor (i.e., 4.8 V). You might compensate by increasing the capacitor size a bit. We are still below the rule-of-thumb values, but maybe you don't like the 48-V max anyway; maybe you like something lower and we quickly approach those stated values.

As you see, no magic, just reasonable engineering once you know what you try to do. Of course, AC is always more cumbersome to calculate and most likely you are talking about higher voltages at the same time, but no different in principle. Add the boundary conditions mentioned before and you are on your way!

We ran a discussion back in the April/May 1992 issue on sensing AC current to determine whether a motor was running. We again broach the subject of sensing AC current, but this time we want to make linear measurements on how much current is flowing.

Msg#:64898

From: DON HOUDEK To: ALL USERS

I need to know how to make an AC current sensor for an energy management control system. I need to measure each of the 110-VAC lines from the fuse panel. The current sensor will be connected to an ADC and monitored by a PC. I need to be able to measure from 0 to 30 amps.

CONNECTIME

Can I use the surplus ferrite toroids like the ones found in some surplus catalogs?. If I can use these toroids, how do I figure what gauge wire to use and how many turns? Also, do you have to use magnetic wire? Thanks for any information that you have to offer.

Msg#:64960
From: JAMES MEYER To: DON HOUDEK

Firstly, you can buy a better current sensor than you can make, and do it cheaper, too.

If you're determined to *make* a sensor, then here's a way. Ferrite is *not * suitable for 60-Hz current sensors. The easiest way to make one is to find a small open-frame filament transformer. A 120-volt to 6.3-volt one rated at 1 amp would be ideal, but almost anything with a 120-volt primary will do. In this case, the cheaper the transformer, the easier it is to modify.

In most cases, the low-voltage secondary is wound last. This makes it the first winding on the outside of the transformer. Use a knife to cut the paper insulation off of the windings. You should be able to see the large diameter wires of the secondary now. Carefully unwind the wire that makes the low-voltage secondary of the transformer. You might want to count the turns as they come off. Try not to break the small-diameter wires of the 120-volt primary.

When the secondary is completely removed you should still have the primary winding on the transformer and you will now have a good-sized "window" left between the primary and the iron "core."

Use some electrical tape to secure the primary winding lead wires and cover any exposed connections. At this point you should have a transformer with only the 120-volt primary left.

If you counted the turns coming off of the secondary, you can divide the number of turns on the secondary by the voltage and get the turns per volt of the transformer. Then multiply the turns per volt by the voltage of the primary, and you'll know about how many turns of wire there are on the primary. We'll need this to determine the calibration of the completed sensor. The proper way to count turns is to count one turn for every time the wire passes through both sides of the core "window." Don't be too concerned about 100% accuracy. Most transformers use between one and two turns per volt, and you can calibrate the sensor later.

The next thing is to add a "burden" or "load" resistor to your sensor. A current transformer (sensor) is only a transformer, and the same math applies to either a current sensor or a power transformer. When you use the sensor, you will pass the wire to be measured through the window in the sensor. This makes a transformer with either a half or full turn for the primary. The winding that used to be the primary is now the secondary. (Gee, this is tough without a lot of drawings, but stick with me, we're almost there.)

The turns ratio of a transformer determines both the voltage 'and' the current ratio between the primary and secondary. Our newly converted transformer has a one-turn primary and a (for example) 150-turn secondary. For each amp in the primary, there will be 1/150 A, or 6.66 mA, in the secondary. Also, for each volt in the primary, there will be 150 V in the secondary. The ADC probably won't like 150 V, so this is where the "burden" resistor comes in.

Let's say that we want to measure up to 20 amps with the sensor. That means that we will have up to 133.33 milliamps in the secondary. Unless we put a load or burden resistor across the secondary, the voltage will rise to whatever value is necessary to get that current to flow. Not a pretty sight if it's connected to a microprocessor! We need to convert that current into a reasonable voltage. Let's say we want five volts on the secondary to represent the full 20 amps on the primary. A resistor will do the trick. Five volts divided by 133.33 milliamps equals 37.5 ohms. That's the value to put across the secondary. There will be a little over a half a watt of dissipation in that resistor, so use at *least * a one-watt-rated resistor.

CONNECTIME

Now you have a voltage that is proportional to the current in the primary. But that voltage is AC. You need to rectify and filter it before you can measure it.

Msg#:65070
From: **SERGIO PICADO** To: DON HOUDEK

Contact F.W. Bell at (407) 678-6900. They've got a good selection of Hall effect current sensors. No sense reinventing the wheel. You're probably looking at their PI series. Ask them to send you a catalog and application sheet.

I've used them on a motor current project and am happy with them. Just put together a straightforward instrumentation op-amp circuit and you're in business.

Msg#:65088
From: PELLERVO **KASKINEN** To: DON HOUDEK

I'd like to compliment James Meyer for a very thorough presentation that needs only a slight correction and then some amplification. The correction is that the small transformers up to a few tens of watts have at least 10 turns per volt. In fact, a transformer with a l-square-centimeter iron core cross-section seems to be about 35 to 37 turns per volt. Only the really big stuff goes into several volts per turn. I remember a 60-kVA transformer we once used that had a 55-V secondary consisting of only about 12 turns. Like James said, it is not paramount to count the turns very accurately, although he did not mention the reason: The common practice for power transformers is to add turns into the secondary for compensating the losses in actual use. You may know that most any transformer produces over the nominal voltage when unloaded. For small transformers, the typical drop from zero to full load is 6 to 10%. Big ones may get along with less than 4%.

Other amplification follows. For measuring transformers, you do not make such large allowances. In fact, many transformers are rated with more than one burden figure, lowest one for highest accuracy, such as 5 VA for 0.5% (accuracy) class and 30 VA for 1% or somesuch.

The other issue is the rectifying of the secondary signal. Normally you do not want the nonlinearity that silicon diodes introduce. As James said, you may want 5 V as the target value. Two times 0.6-V deadband does not make it too easy to use the results, in addition to the basic problem that the rectified and filtered result corresponds to the

#145

average, not the RMS, value of the sinusoidal signal. You have two ways out: use a bipolar input range on the ADC and then doing everything laboriously with a fast computer, or you put the rectifier bridge (with PLENTY of safety margin in voltage and current ratings) BEFORE the load resistor(s). I still recommend oversizing and duplicating by way of a parallel connection of the resistors for safety.

As far as ferrite is concerned for a core material, some of them are actually somewhat suitable, though not too practical. The suitability comes from low-loss characteristics that promote accuracy and linearity. But their saturation flux density is so low that they become awfully bulky, needing 3 to 10 times the number of turns in the windings that an ordinary silicon steel core needs.

My recommendation is that you indeed go for a commercial current transformer. They use lower-loss steel than the power transformers and therefore offer better accuracy. But what you should try to look for is a transformer with 1 -A secondary rather than the more common 5-A standard issue. This cuts down the size and power of the load resistors and the rectifier diodes (the resistors by a factor of 5^2, i.e., by 25!).

We invite you to call the Circuit Cellar BBS and exchange messages and files with other Circuit Cellar readers. It is available 24 hours a day and may be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps.

ARTICLE SOFTWARE

Software for the articles in this and past issues of *The Computer Applications Journal* may be downloaded from the Circuit Cellar BBS free of charge. For those unable to download files, the software is also available on one 360K IBM PC-format disk for only $12.

To order Software on Disk, send check or money order to: The Computer Applications Journal, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your VISA or Mastercard and call (203) 8752199. Be sure to specify the issue number of each disk you order. Please add $3 for shipping outside the U.S.

428 Very Useful 429 Moderately Useful 430 Not Useful

STEVE'S OWN INK

The Sample Game

I understand the current problems in the economy can cause a cutback in support materials, but getting information from American electronics giants can be a exercise in futility. Companies that are spending millions in full-page ads describing their latest widget or chip on one hand have either eliminated the field service personnel who can explain its use and authorize samples or left support entirely to their distributors. Of course, in a similar economic move, the distributors' work force is now full of former used car and copier salesmen who fear losing their job so much that they only focus on the "big kill."

Calling virtually any of the major chip manufacturers for technical information on a particular chip results in a frustrating phone tag game. The communication starts like, "Sorry, you've got the photosemiconductor slash visible development group here in Chicago. Perhaps you want the photosemiconductor slash infrared group in California." When you get to the infrared group you hear, "Gee, this is the incoherent radiation group. Perhaps the coherent guys up in Idaho know something."

Of course, when you finally get to them, the answer is, "Sorry, the person in charge of that product is on the road for the next month meeting with potential OEMs. If you think your application might use 100,000 quantity or more, I can leave a message and I'm sure Mr. So-'n-So will get back to you. Until then I suggest you call distribution. They should know as much as we do."

The truth of the matter is distribution probably does know just about as much as they do -nothing. If I had wanted the masochism of dealing with a distributor, I would have started there.

While not all distributors are the same, the distribution game has become primarily one of customer qualification rather then information and application support. A cold call to a distributor often results in passing the buck: "We have a $350 minimum order or, if you fax us your request on letterhead, we'll forward your data sheet and sample request to the factory."

Dealing with distribution is like buying a car. You either have to pretend like the last thing in the world you want is a car, or you drive up in a Porsche with the attitude that you might drop them a few crumbs if they impress you.

Distribution is solely motivated by money and the potential for sales. If you get their attention at all, the first questions will be what company you work for and how much are you going to buy. The only proper answers are "500" and "10k."

Somehow the same lack of knowledge of electronics and economic principles extends to their concept of the business community and potential customers in general. Unless you claim to represent one of the companies in the Fortune 500 (hopefully giving them an order equivalent to winning the lottery), be prepared for a dial tone. Companies like Micro Ventures, or Entrepreneurial Electronics Inc. don't give them visions of the "deal of a lifetime."

Finally, the only way to break the 500 wall is the 1 Ok quote. Rather than the meek, "I need information approach," we try the, "This is purchasing and I wanna buy it fast!" approach. "Yeah, Micro Ventures. Don't worry about whether you know us or not. We buy $thousands from your competitors. What I want immediately is a data sheet faxed on part XYZ, price and delivery on 1–10k pieces, and how many you have in stock. Oh yeah, the engineers want a couple pieces Fed Exed so they know it works like ZYX."

Unfortunately, this latter technique works all too often.

If all new ideas came from 500 companies, there would only be 500 companies. That is not the case, and if we know it, why is it taking so long to sink in at the top?

Fortunately, smaller, more entrepreneurial companies have sensed the void and are filling the gap. I'm hopeful fast-moving high-tech direct-customer-support companies like Maxim and Dallas Semiconductor will force a continuing trend in the electronics industry. Much like their European and Japanese rivals, Dallas and Maxim support individual requests with a sense of urgency that makes you feel like a customer again.

It took General Motors a decade to learn a hard lesson about customer relations. Fortunately, the company was big'enough to learn and still live. We can only wish that Big Electronics gets the message in time.

Steve