CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

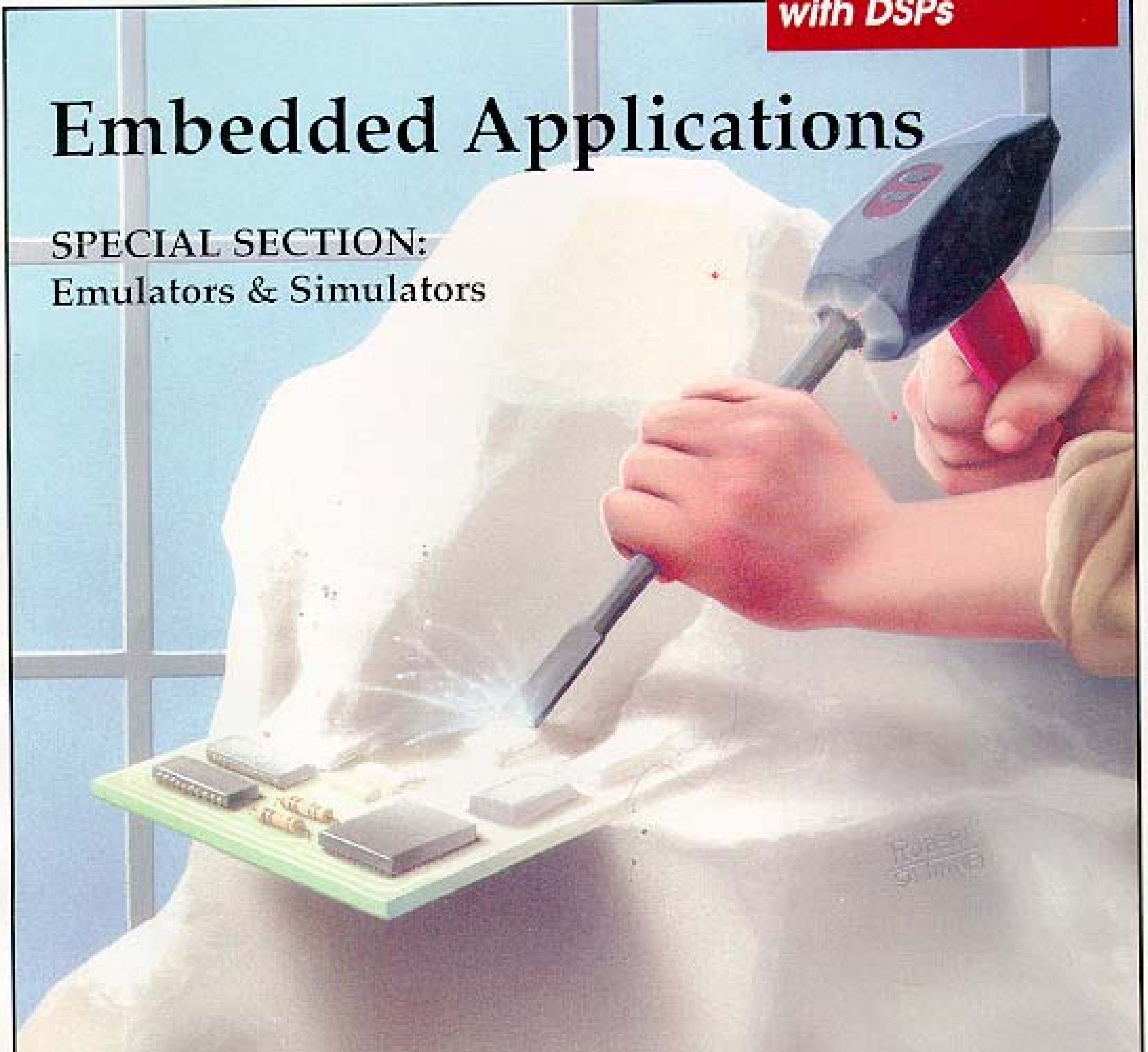
Writing Device Drivers

ISDN on Your Desktop

Number Crunching with DSPs

Embedded Applications

SPECIAL SECTION:
Emulators & Simulators



August/September 1991 — Issue 22

$3.95 U.S.
$4.95 Canada

08

0 74470 75349 0

The March of Technology

EDITOR'S INK

Curtis Franklin, Jr.

I don't often get the ideas for these editorials from entertainers, but the title you see above came from a speech given by Sir Peter Ustinov to thc National Press Club on June 18, 1991. He used thephraseintheanswertoaqucstionaboutthccausesofthe recent changes in eastern Europe. Regardless of your position on any number of subjects, two facts are incontrovcrtiblc: Technology is constantly changing, and the changes in technology affect the way we live. Voices have been raised in protest against the seemingly inexorable march of technological progress, but many of these latter-day Luddites have merely established technology as a shibboleth for rallying their followers: They use a broad brush to paint every technological advance as evil rather than attempting to understand how a particular technology effects their area of concern.

Of course, if you're reading this magazine of your own free will, you don't need me to tell you all of this. So far, I've been "preaching to the choir." I've got to convince you, though, that all of this does matter because the people who don't understand technology are reporting the news and making the laws in far too much of this world. Let me give you a couple of examples.

Early this year, a comprehensive act concerning terrorism was introduced in thc U. S. senate. One of its many provisions stated that the lawmakers wanted law enforcement officials to be able to look at any file on any computer in the land. One of the results of this would be to force every programmer working on data security or encryption to leave a "back door" through which law enforcement officials could look at data. The bottom line would be that data security, as we know it, would cease to exist since any security system with a back door is no longer secure. Fortunately, folks from several industry organizations went to the sponsor and explained why thc bill would have unintended results: The passage has been removed from the bill. Governmental bodies on every level are becoming involved with more and more technical and technologically advanced issues. In many cases they appear to be acting with no input from people who are wcll versed in the technology at issue. The results from any one of these cases could range from inconvenient to disastrous, and most of the population will not care because they are not experts and have no one to explain the situation to them.

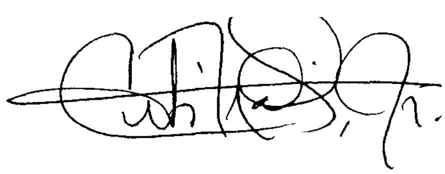
Ideally, the mass-media press would be able to make even thcmost complex issueclear to their readers. Onerecent incident has shown that **they** are capable of dangerous misunderstanding regarding technological issues. I'm referring to thc Great Prodigy Data Theft Caper.

The Wall Street Journal started the ball rolling with an article on a particular file that Prodigy's software keeps on the user's hard disk. Prodigy uses the file, about a megabyte in size, for storing graphics primitives so that their fancy graphics are merely slow and not glacial. Some genius took a look at this file with a sector editor and found bits and pieces of old data inside. They then decided that this was a nefarious plan by IBM and Sears to suck personal data off the hard disks of subscribers, and come up with a plan to make us all wear blue suits and buy Kenmore appliances. It was a great story, but it had the flaw of being completely ridiculous. Anyone who knows about MS-DOS and its putative file system realizes that deleting a file makes a change in thedirectory, not in thcdata sectors. Whether you run Prodigy or not, if you look at your disk with a sector editor you'll find bits and pieces of old files.

Furthermore, 1 refuse to believe that even IBM has the computing horsepower to suck in random chunks of an old guacamole recipe, a spreadsheet based on the assumption that I'm as rich as the Sultan of Brunei, and a letter to my great-aunt Kathryn and reach any conclusion at all. The story had no sound technological basis, and was blown complctelyout of proportion by people who hadn't the information to know better.

Whyaml tellingyou this? The readers of CIRCUIT CELLAR INK are in a unique position to make a difference in both areas, government and media. I frankly don't care which side of an issue you take, but when the issue involves computers, electronics, or other technologies in which you're conversant, let the officials involved know how you feel and why. In a similar fashion, it's all too easy to see a boneheaded factual error in a newspaper or magazine article on computer and laugh, but I've resolved that after I finish laughing I will write the editor of the offending publication and tell them how they erred. If it happens often enough they'll realize that they have readers who know enough about thc technology to care when the media makes a mistake.

Hardware and software design teach us that the details count. Keeping elected officials, bureaucrats, and the fourth estate informed and honest is mostly a matter of letting them know about the details they've missed. I think it's an effort worth putting forth.



CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

In This Issue...

FEATURES

14 **Using Device Drivers to Change the Rules**
by Chris Ciarcia
MS-DOS device drivers are the key to controlling PC hardware at the lowest level. Learning how to write drivers can open up your applications to greater functionality and flexibility.

26 **ISDN (S/T) Interface-Part 1**
General Review of Functional Concepts
by Steven *E.* Strauss & *P.K.* *Govind*
integrated Services Digital Network (ISDN) is making the transition from planning to reality. Understanding the nature of the network is an important first step toward using ISDN's potential. Next time: The hardware for making ISDN work with your PC!

32 **S-ART: Building the Network Software-Part 2**
by John Dybowski
Control networks are the sum of their hardware and software. The concluding part of this article talks about the software for tying the net together.

60 **Numerical Applications Using DSP**
Using a DSP Chip for High-Speed Numeric Processing
by Eduardo ***Pérez &*** *Dapang Chen*
Digital Signal Processors are frequently more than just signal processors, they're high-speed number crunchers in their own right. Knowing what they can do may open up new applications for these popular and powerful processors.

SPECIAL SECTION: Emulators & Simulators

42 **But it Worked with My Emulator!**
Why Emulation Isn't Reality
by Keith ***Wentworth***
Emulators are valuable tools when it comes to working the bugs out of a new design, but there are traps and pitfalls waiting for the unwary. Knowing what to look for when you use an emulator can reduce the surprises when you move your software to the final platform.

49 **Son of DDT: A New 8031 Debugger**
by Ed Nisley
The DDT-51 was a break-through project in low-cost debugging and development equipment. Now, it's time for the next generation. Find out what four years of input from users, field experience, and technical development have done for the personal embedded software development.

DEPARTMENTS

1 **Editor's INK**
The March of Technology
by Curtis Franklin, Jr.

4 **Reader's** INK-Letters to the Editor

8 *NEW* **Product** News

74 **Firmware Furnace**
Toolmaker's Toolworks
by Ed ***Nisley***

81 **From the Bench**
Reducing Power Consumption
Breathing New Life Into Data Logging
by Jeff Bachiochi

88 **Silicon Update**
Kynar To The Rescue
The Ultimate Sensor?
by Tom ***Cantrell***

95 **Practical Algorithms**
Summarizing Your Data
Properties of a Bounded Probability Density Function
by Charles ***P. Boegli***

104 **ConnecTime**—Excerpts from the Circuit Cellar **BBS**
Conducted by Ken Davidson

112 **Steve's Own INK**
A Standard Column
by Steve ***Ciarcia***

97 **Advertiser's Index**

Circuit Cellar BBS-24 Hrs. 300/1200/2400 bps.8 bits, no parity, 1 stop bit, (203) 871-1988: 9600 bps HST (203) 87 l-0549

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

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

READER'S INK

Letters to the Editor

MORE ON STANDARD DEVIATION

Charles P. Boegli provided us with a description of a tool in hisarticle "Adjustingstandard Deviation to Sample Size" in CIRCUIT CELLAR INK #20. It has a very powerful application if used in a more useful form. Some of us average data to get a "better" number. He has shown us that we should average at least 10 samples.

What is the quality of our data? The standard deviation used as calculated is a difficult-to-interpret number. What some of us would like to have is a measure of quality. If the standard deviation is presented in terms of percent of the mean value, we have what I will call the quality.

A quality of O.OOOOOOOOOO... would indicate a near perfect measurement. The number of zeros after the decimal point indicates how good the data is. A lower number of zeros means higher quality. Not being a mathematician, or having even taken statistics, the actual numerical significance of this number is unknown to me. It seems believable that it could predict the number of significant digits in the measured value.

In a DC (steady state) measurement system, where a stimulus is applied and an average value is measured, the procedure usually takes the form of: (1) Apply stimulus, (2) Wait x, (3) n: (Measure, Wait y}, (4) Average. If the quality of the data is computed along with the average value, programs can reject those values that are not measurable and/or unstable automatically.

While using the notion of data quality, I found a flaw in step 2. If the first stimulus is large and the subsequent stimuli small, the first data point would have a larger quality value, thus lower quality, especially if the samples measured are capacitive. The x wait time needs to be adjusted by a function of the magnitude of change in external stimulus if you want the steady state value.

The above information should prove useful for those characterizing electrical devices and those who write programs that need to determine the validity of a value measured by a sensor that is assumed to be in a steady state.

Ron Dozier
Wilmington, DE

***Editor's Note:** In CIRCUIT CELLAR INK #21 we published a letter by Mr. Norman Stanley which criticized the article "Adjusting Standard Deviation to Sample Size" from CIRCUIT CELLAR INK #20. The author responds:*

I appreciate the effort Mr. Norman Stanley has spent defending the status quo against my paper, and will reply as briefly as possible. Apparently I was overenthusiastic in saying that $(0/x)^{0.5}$ is imaginary when x is negative; as he correctly observes, it is zero. Since the expression is also zero when x is positive (as we both agree), the indeterminate expression obtained when $x=0$ likely also evaluates to zero. None of these methods is useful with a single sample, which we all knew anyway.

In hastilybrushingaway the raison d'être of my paper, Mr. Stanley illustrates the kind of personal discomfort I alluded to in its opening. My correction comes from a table in the identified recent (1985) source, from which I now quote: "When we wish to refer to the standard deviation of.. .a parent population, we use the symbol σ'...[which] is usually unknown. However, it is possible to estimate σ' by using.. .a series of samples as follows:

σ' = (sigma of a sample fo a given size) x $1/c_2$

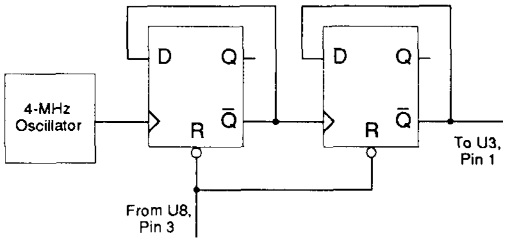
where c_2 is a factor that varies with the sample size." This correction is applied to a SD obtained with the divisor n, for which the symbol ois used there. I find no ambiguity in that statement. The table is found in many sources and by inclining "to view it with suspicion" Mr. Stanley casually sidesteps the central issue. Fitting the value of $1/c_2$ to the sample size by an expression that can be neatly introduced into that for ois my own work. Mr. Stanley correctly observes that this expression gives σ' (or, as he will have it, σ) even larger than *n-l.* It also took me a while to recover from the shock at calculating this value.

My source manifestly uses symbols other than those Mr. Stanley prefers, and he will find other usages in other texts. Perhaps there is less unity here than he suggests. I prefer to use SD and take the trouble each time to define it.

Charles Boegli
Blanchester, OH

TIMER DESIGN

I appreciated the article "A PC Stopwatch" in CIRCUIT CELLAR INK #20. Here is a simple improvement which everyone should include. If the 1-MHz oscillator is high when the gate goes high, the counter will be clocked immediately. A similar condition exists at the end of the gate, so that any gate can give either a true count or one more than the true count. This "bobble," as it is called, is quite mystifying if you don't know what causes it. The fix is to synchronize the clock to the gate.



Vem Wall
Phoenix, AZ

THE LOCAL OPERATING NETWORK

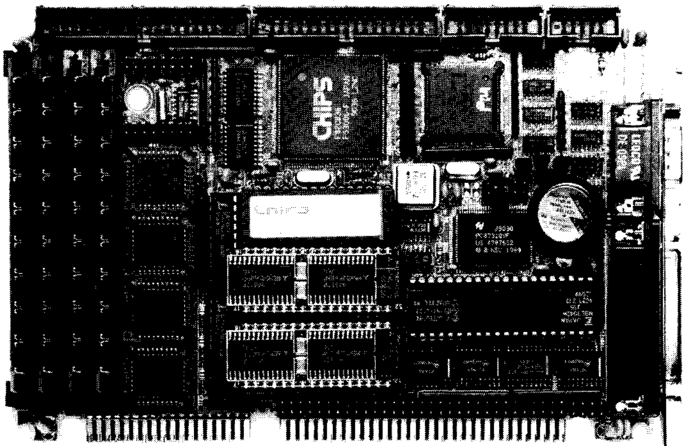
I read Ken Davidson's article "Echelon's Local Operating Network" in CIRCUIT CELLAR INK #21 with much interest. I also attended an Echelon Seminar on June 6, so I heard essentially the same spiel Mr. Davidson did, although there was no evidence of any "CEBus-bashing," and no condescending references to the various levels of expertise that users would display. Mr. Davidson's factual description of Echelon's technology is accurate, but I feel I sense some defensiveness in his conclusion, which reads like a commercial for CEBus.

I have no preference either way between LONs and CEBus. I do feel that an examination of the Echelon technology reveals that it is well thought-out and has much to offer. Its development system and tools are excellent but expensive ($14,000+), which will discourage experimenters from using it. I was personally impressed by the Neuron C language and the ease with which network and control applications can be written and debugged.

Whether LONs, CEBus, both, or neither will prove useful will be determined by users and the marketplace. I suspect both methods have a place in the scheme of things.

As far as standards go, they must be specified unambiguously so that all implementations are compatible and interoperable, and feel that it is vital to have an organization that validates the various implementations to ensure compatibility, as we do with VME Bus-compatible equipment.

Kenneth J. Ciszewski
Overland, MO



PALM-SIZE 386SX SINGLE-BOARD COMPUTER

An industrial-grade single-board computer, featuring a 16- or 20-MHz 386SX microprocessor, has been announced by Teknor Microsystems. Measuring only 7" x 4.7" (178 mm x 119 mm), the TEK-AT2 board runs either in stand-alone mode or on a PC/AT bus. The board uses all CMOS technology for low power dissipation and extended temperature operation.
Up to 16 megabytes of mixed DRAM are supported. An optional 1M-byte Flash EPROM and 1M-byte SRAM with battery backup are available, as well as optional piggyback support for an 80387SX math coprocessor. The TEK-AT2 contains both hard and floppy controllers, two serial ports, and one parallel port. Video support is available through an optional add-on board. Setup software in ROM and on disk is provided.

The TEK-AT2 includes software utilities, complete documentation, and full technical support. The board is priced at $725 in 100-unit quantities (16-MHz model with no memory).

Teknor Microsystems, inc.
C.P. 455
Sainte-Thérèse (Québec)
Canada J7E 4J8
(514) 437-5682
Fax: (514) 437-8053

Reader Service #500

PROCESS CONTROL/DATA ACQUISITION SOFTWARE WITH RULE-BASED FEATURES

A software package that features a rule-based, table fill-out application generator is available from Automated Control Systems. PACX (an acronym for Process Acquisition and Control expert) eliminates traditional programming by allowing a user to describe the application instead of coding it. The main part of the PACX system consists of the System Builder, an application generator that is a menudriven editor. The editor, which is the only user interface, consists of three parts: one used to describe the hardware, one to describe the application, and one for the artificial intelligence section. Based on this description, the editor creates a "Knowledge Base" which consists of a database of I/O points and a set of rules for control. The set of rules in the Knowledge Base is used by a real-time multitasking operating system. As the operating system runs, it uses drivers to read and write the I/O points of the controlled process. PACX has been designed such that at each given time, only the I/O points that are critical for correct control of the process are updated. This optimizes the operation of the computer for fast control and data acquisition.

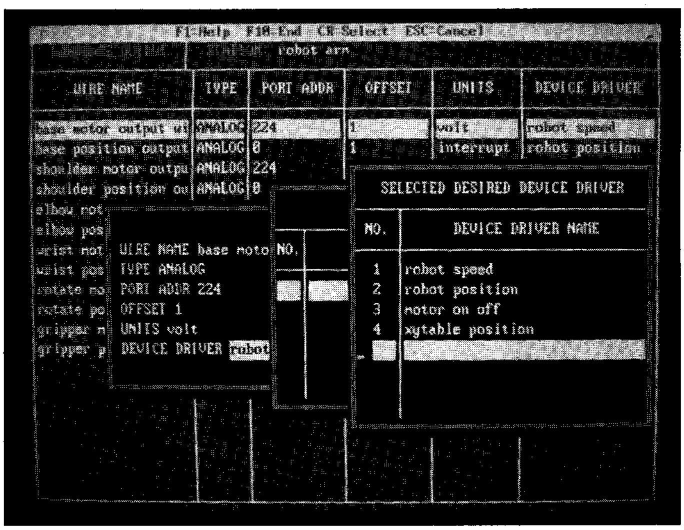
A complete set of mathematical operators is available for both digital and analog calculations. Inputs are checked against a mathematical expression for true conditions to drive the rule base to new rules. Outputs may also use full mathematical expressions which utilize any of the other I/O for calculating their settings.

A full set of timing operations including relative timers, wall clock time, and minimum and maximum for each rule is also available. During run time, full screen control is available to the user. Text-based menus, data entries, and messages may all be built and controlled using the System Builder, giving the operator full control for interactive-type applications.

The PACX Data Acquisition and Control Software is PC compatible and consists of the PACX System Builder ($349.00) and the PACX Run Time ($349.00). Available device drivers are $99.00 each, and customized device drivers can be obtained.

Automated Control Systems, Inc.
P.O. Box 49 • Provo, UT 84603-0049 • **(801)** 373-0678

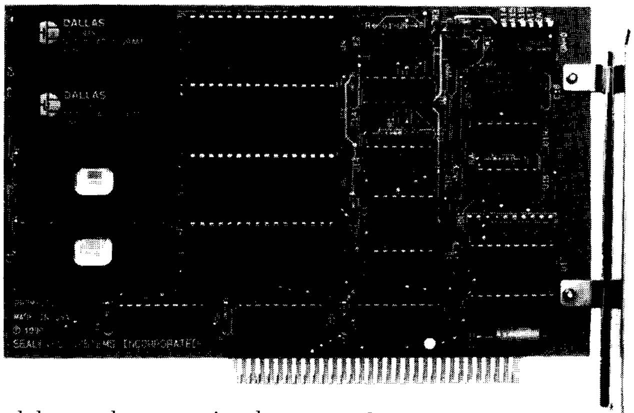
Reader Service #501



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

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

FLASH MEMORY DISK EMULATOR BOARD



Sealevel Systems has announced the PROM-III Flash Memory Disk Emulator Board. The board can replace both hard disks and floppy disks in embedded PC applications where critical data retention is required. The PROM-III stores up to one megabyte of data in flash EPROM. Flash EPROM allows erasure and reprogramming without the need to physically remove the chips. The PROM-III also supports battery RAM modules and conventional EPROMs.

The board works in conjunction with the PROMkit Software from Annabooks. Both DOS and application programs can be placed in EPROM and automatically booted. Applications include factory automation, dedicated PC controllers, diskless workstations, and secure network stations.

The PROM-III sells for $239.00 with OK, $319.00 with 360K of flash EPROM installed, and $389.00 with 720K of flash EPROM. Software to program the flash memory devices is included at no charge.

Sealevel Systems, Inc.
P.O. Box 1808
107 S. **Pendleton St.**
Easley, SC 29640
(803) 855- 158 1

Reader Service #502

DEVELOPMENT SYSTEM FOR EMBEDDED CONTROL APPLICATIONS

A complete prototyping board for developing embedded control applications is available from Rigel Corporation. The R-535 Prototyping Board with R-Ware features efficient software development and rapid hardware prototyping combined in a single integrated development environment. The R-535 uses the 80535 microcontroller, which is an enhanced version of the popular Intel 8032 controller. The R-535 can be used to develop programs for the 8031 family of microcontrollers. The R-535 uses external RAM during the development cycle. Once an application program is developed, the R-535 has the capability to permanently place the application program in EPROM using the on-board EPROM burner. With an application-specific program installed, the R-535 may be used to emulate an embedded controller.

The software development effort is greatly simplified by R-Ware's integrated host environment. The environment is completely menu driven. It includes an editor, cross-assembler, and debug utilities. The ROM-based software residing on the R-535 board complements the PC-based host software. Together, R-Ware handles all PC-to-board communications, program download, debugging, and EPROM burning. Besides components of the integrated environment, the ROM-based on-board software includes operating system utilities complete with user-accessible system calls and an ASCII terminal mode to be used when a PC-host is unavailable.

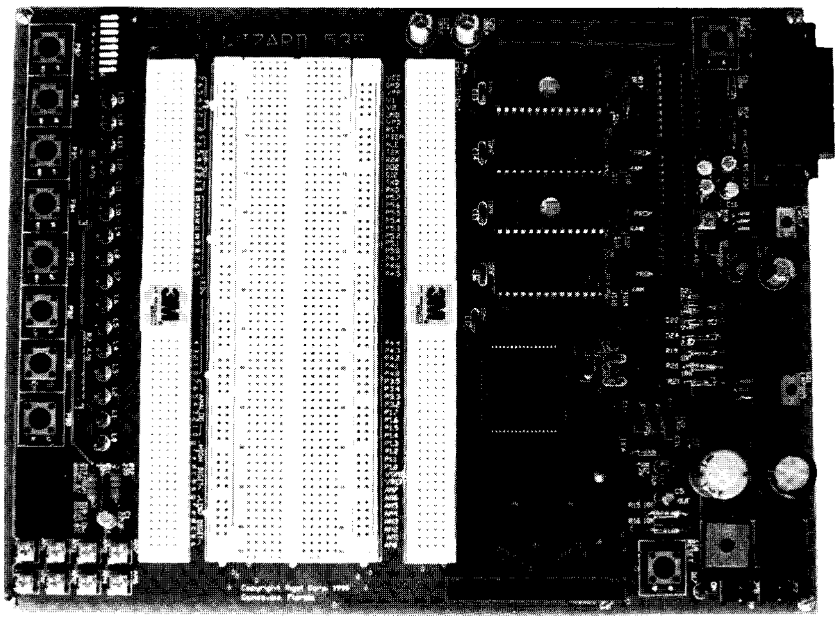
Prototyping components consisting of push buttons, DIP switches, LEDs, numerical displays, and potentiometers are used for emulating control application inputs and outputs. The large solderless breadboard terminal strip is used to construct application-specific circuits to be interfaced with the microcontroller. All microcontroller ports and all major system control lines are available through a 54-post solderless breadboard terminal strip. In addition, all system lines are available by two standard 40-pin headers. These provide flexibility for connecting prototyping components to the microcontroller lines, and for developing and debugging user-designed analog and digital application circuits.

The R-535/R-Ware includes a 100+ page user's guide with schematics, board layout, and control experiments with example software.

The R-535/R-Ware is available as a kit for $395.00 (includes all necessary components and an assembly manual) *or* assembled and tested for $495.00.

Rigel Corporation . P.O. Box 90040
Gainesville, FL 32607
(904) 373-4629

Reader Service #503



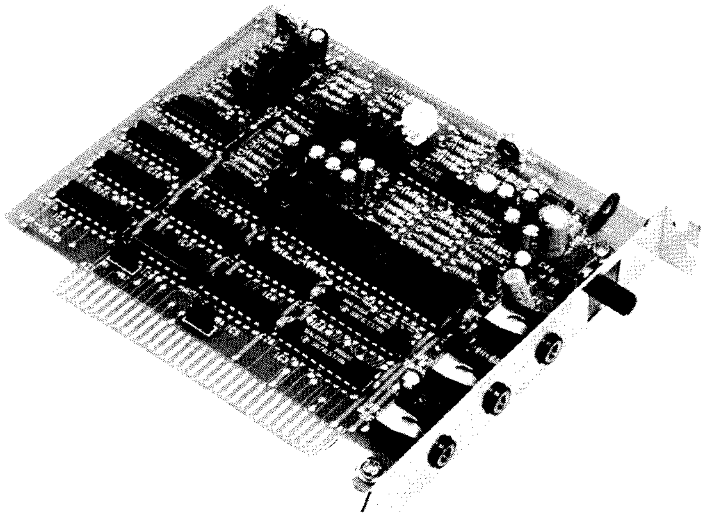
PC VOICE CARD FEATURES DATA COMPRESSION

A proprietary real-time voice data compression scheme that can usually reduce data size by about 30% without voice quality degradation is available in a voice card from Eletech Electronics. Compression is achieved by not recording the pauses between spoken words and sentences. Instead, a special code designating the length of the pause is recorded, reducing thousands of bytes of digitized "silence" to just a few bytes.

When voice data is played back, a period of silence is added whenever the special silence code is encountered. This approach results in virtually no voice quality degradation since only the pauses and not the words are compressed for storage.

This compression scheme has been incorporated on the **DigiCorder PC Voice Card,** an g-bit half-length card that fits any available slot in an IBM PC/XT/AT or compatible. Digital recording and playback at adjustable sampling rates are possible, and the unit's input supports standard microphones and tape players. The length of the recording (or the size of the data file) is limited only by the capacity of the disk drive.

A menu-driven utility program is included to provide functions of record, playback, and limited speech editing. Line command utility programs for recording or playback are also included to facilitate batch mode operation. All programs run under the PC-DOS operating system.



The DigiCorder sells for $99.00 and comes with a high-quality speaker and microphone, and utility programs and device drivers supporting Micro-soft C and QuickBASIC under DOS. Clipper support is a $79.00 option.

Eletech Electronics, Inc.
1262 Katella Ave.
Anaheim, CA 92805
(714) 385-1707
Fax: (714) 385-1708

Reader Service #504

COST-EFFECTIVE DATA LOGGING SYSTEM

A unique data-logging system that consists of a software package and an input module that connects to the serial port of any IBM or compatible computer has been announced by Woods Electronics. The Count Logger acquires data in real time by counting pulses which are applied to the input module, and sending the count to the screen, disk data file, and/or printer.

Each time a pulse is applied to the module, Count Logger tricks the serial port into thinking a valid RS-232 data byte has been received, and accepts the pulse as one count. The module shapes the input pulses so that any wave shape will be accepted as one pulse. To log randomly occurring events, the output of the event detector is connected to the input module. To log frequency, the output of a signal generator is connected directly to the module.

Any quantity can be measured by converting it to a frequency. For example, to measure temperature, a simple temperature-to-frequency converter is connected to the stock module.

Count Logger records the number of counts received within a precise user-specified time period ranging from 0.2 seconds to 1 year. The number of time periods can be set from 1 to 10,000,000 or indefinite. The starting time can be set to a future time. Each time period also records the actual time at which it occurred. As a result, each logged entry contains the data needed to fully analyze the content of the input pulses. The data is continuously analyzed by the Count Logger and sent to the screen for immediate use. The data file to which data is recorded is in ASCII format and can be read, printed, edited, or sent to your spreadsheet or database for further analysis or sophisticated display.

The output to screen, file, or printer includes user-inserted messages, time, length of sample period, number of counts in that period, frequency, total counts in all sample periods, average number of counts per unit of time, and error messages. An on-disk operating manual is provided along with error checking of machine and user-selected options.

Count Logger is available for $169.00 including input module and real-time program. Any two-pin jack can be connected to the pigtails of the input module. A DB-25-to-DB-9 connector is also available for $7.50.

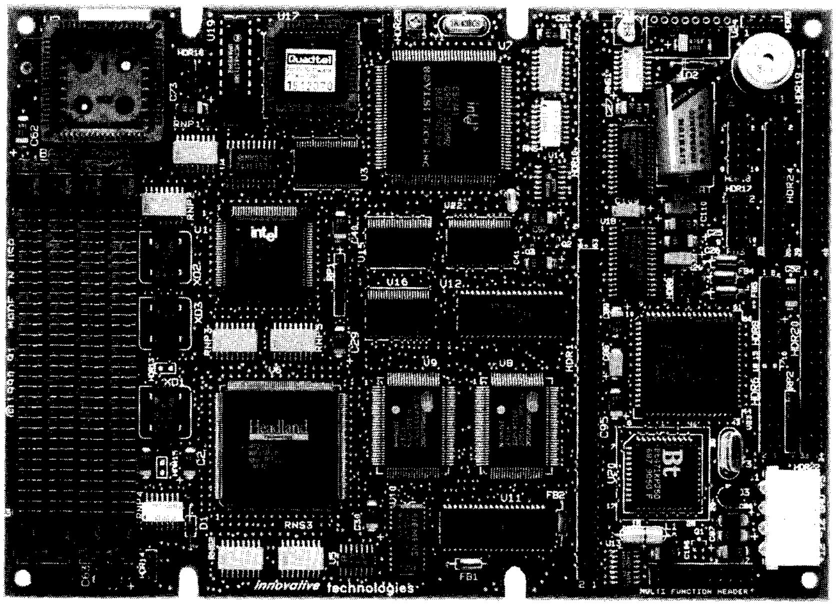
Woods Electronics, Inc.
4233 Spring St., Ste 117
La Mesa, CA 91941

Reader Service #505

ALL-IN-ONE 386SX-BASED BOARD

Innovative Technologies has introduced a new single-board computer system targeted toward the embedded applications market. The unit, which has been designated the it/sx, integrates all functions normally found in a complete AT-compatible system onto a single six-layer circuit board measuring 5.75" x 8"—a format popular within the embedded systems marketplace because it matches the footprint of 5.25" disk drives.

The it/sx is built around a 20-MHz 386SX microprocessor, and is completely compatible at the hardware level with the IBM PC/AT standard; equivalent software compatibility is ensured through the board's industry-standard Quadtel BIOS. The system currently accepts up to four megabytes of on-board dynamic memory, which will increase to sixteen megabytes by the fourth quarter of 1991.



An on-board VGA display controller supports analog, digital, and flat-panel monitors, and is backwardly compatible with CGA, MDA, Hercules, and EGA display standards. The Cirrus Logic chip set which lies at the heart of this controller is renowned for its grayscale rendering on monochrome LCDs; an optional Continuous Edge Graphics RAMDAC quadruples perceived resolution for even more stunning images on analog CRTs.

A floppy disk controller as well as an IDE hard disk interface also appear on the board. Each can support up to two drives, with the floppy controller allowing any combination of 360K, 720K, 1.2M, and 1.44M formats. The it/sx also supports the latest generation of floppy disk drives which employ perpendicular recording techniques to achieve storage capacities of 2.88M, and allows the system's second drive to be designated as a tape backup unit by modifying the controller's phase-locked loop when in tape drive mode. Alternatively, diskless operation may be elected through use of the system's "ROM disk" support, which allows implementation of a 128K-, 384K-, or 896K-byte on-board ROM disk.

Other standard features include two RS-232C serial ports, a Centronics-compatible parallel port, real-time clock, keyboard controller, and PS/2-compatible mouse port. An on-board math coprocessor socket is provided to facilitate 80387SX support.

The it/sx operates from a +5V-only power supply, and with four megabytes of memory, typically consumes less that five watts. A PC/AT-compatible ("ISA") expansion bus header allows connection of virtually any off-the-shelf expansion card; to enhance the compatibility of this header, the board allows pass-through of +12V and -12V from the power connector, and provides a socket for an optional on-board DC/DC converter which supplies the bus with -5V at up to 200 mA.

The complete is/sx system (without memory) carriers a quantity-one price of $1095.

Innovative Technologies
P.O. Box 90086
Houston, TX 77290
(713) 583-l 141

Reader Service #506

REMOTE COMPUTER CONTROL

Scheduling a PC to run applications on a totally unattended basis is now possible with a new hardware device from The Pendulum Group. The PowerPak allows a user to power on or off a PC and other devices from a remote location. The unit can also turn a PC on at a preselected time, and automatically turn it off after running an application.

The PC receives its power from the PowerPak, and the PowerPak receives its commands from the PC. Users can preset a time for the PowerPak to turn off the PC following a task such as tape backup, or the printing of a lengthy report. There is no need to leave a system on to perform late-night data transfers or other tasks.

Another application uses the PowerPak, an external modem, and a remote communication software package to allow access to the PC from a remote site. The user instructs the PowerPak to turn on the PC via a modem command, uses the communications package to do work on the PC, and then powers down the PC with another command.

The PowerPak (which includes Auto-Might, an event scheduling software package) runs on IBM PC/XT/AT and compatibles. It retails for $379.

The Pendulum Group, Inc.
333 West Hampden Ave., Ste. 1015
Englewood, CO 80110
(303) 781-0575 . Fax: (303) 761-2440

Reader Service #507

Reader Service# 148

Reader Service #155

FEATURE ARTICLES

page 14
Using Device Drivers to Change the Rules

page 26
ISDN (S/T) Interface—Part 1

page 32
S-ART: Building the Network Software—Part 2

page 60
Numerical Applications Using DSP

Using Device Drivers to Change the Rules

For most of us who work within the DOS environment, our only contact with device drivers comes when we load their images and then make the appropriate entries into our CONFIG.SYS file. Invariably, they seem to take on the form of some strange, cryptic, unintelligible, and necessary-"do I really have to have it?" definition line, which weall know represents a continued loss of those few remaining bytes of program space. As I have pushed my software against the all-too-well-known MS-DOS wall of oblivion, they have provided me withadditionalammunition tohandle programming situations where I needed to break the rules in creating some slick and fancy application.

In the traditional sense, device drivers are written to attach a new device to a system. They are program modules that insulate the computer system from attached hardware by providinga basic communications interface between the two. Of course, you could make your device look more like something your system's ROMs already support, but that seems to represent an inordinate amount of work. Instead, designing a piece of software that looks like an extension of DOS itself, while performing the required interface functions, seems more practicable. Of equal importance, I have found that device drivers can serve purposes other than hardware interfacing. They can serve as powerful tools for creating a window into DOS during the boot-up phase. A driver loaded into low memory directly above the operating system's file-control blocks and disk buffers is initialized early in the boot-up process, well before the AUTOEXEC.BAT file is executed. This early initialization can be very useful.

A LEARNING EXPERIENCE...

I recently developed a special security procedure that needed to be activated each time the system was booted. It was designed to deny unauthorized access to sensitive data even if a floppy system disk was used to boot the computer. It had to be absolutely uninterruptable since any disruption in its execution would always result in a total system crash. The code was designed to remap many of the basic DOS interrupt vectors and modify the hard disk's work partition area ownership, which would be set and reset at the beginning and end of each user (login/logout) sequence. So I needed to find a way to isolate the boot sequence (program operation) from the operator.

My first attempt was to place the code into the first line of my AUTOEXEC.BAT fileand set theBREAK switch to OFF. I expected it to run, do the necessary interrupt handling, and when security requirements were met I expected it to modify the disk partition table allowing access to sensitive data. Like most high-level programmers who typically ignore system "things," I quickly found out that a carefully inserted CTRL-BREAK, key pulsed during the boot sequence, invariably generated a Terminate batch file? prompt, allowing me to bypass the AUTOEXEC.BAT file. This was not acceptable. I needed my code to run in an unstoppable mode. I thendebugged COMMAND.COM and inserted a portion

FEATURE ARTICLE

Chris Ciarcia

of my own code to vector the COMMAND . COM procedure to my security code at the appropriate time. I gave that up when all I could see was years and years of versions as each new DOS came out. I then tried to figure out how I could get my application to run from within CONFIG.SYS. As it turned out, I was able to implement a procedure by creating my own operating system environment (using the CONFIG.SYS SHELL command) and then mirroring the COMMAND. COM functions and passing control to AUTOEXEC. BAT. It worked! For a while I was happy, but the new solution was a memory hog. I lost an additional 10K of my 640K above and beyond what I would usually use for COMMAND. COM.

Then I got smart-I learned to write device drivers. I worked out a simple procedure whereby a very basic device driver was loaded, which during its initialization phase masked out the keyboard by resetting the keyboard interrupt vector. This effectively disabled any unwanted keystrokes and attempts to circumvent the procedure using CTRL-BREAKS. My security code was then run as the first program within the AUTOEXEC batch file. It did its thing with the hard disk partition and it then reset the keyboard interrupt vector when it was ready to pass control to an authorized operator. Without the driver in place, the security code was ineffective and access to the hard-disk work partition was denied. This made sensitive files on the hard disk fairly safe from the random floppy disk system booter, provided they didn't come with Norton utilities on their disk. Then again, PCs were not designed to be secure, so all I needed to do was make it harder to break the rules.

As such, my use of a device driver to mask the keyboard demonstrates how it provided me with an access window into the DOS boot sequence. It was such a simple and powerful tool that I'd like to share with you some of the basics I learned about device drivers. I'd like to discuss how DOS loads and initializes a driver, its basic structure, and how I used it to mask the start-up sequence.

Address	Name	Size	Type
000000		000400	Interrupt Vector
000400		000100	ROM Communication Area
000500		000200	DOS Communication Area
000700	IO	002510	System Program
	NUL		DOS NUL device driver
	CON		DOS console driver
	AUX		DOS serial port driver
	PRN		DOS printer driver
	CLOCK$		DOS clock driver
	A:-C:		Drives A: thru C:
	COM1		System Device Driver
	LPT1		System Device Driver
	LPT2		System Device Driver
	LPT3		System Device Driver
	COM2		System Device Driver
	COM3		System Device Driver
	COM4		System Device Driver
002C10	MSDOS	008E20	System Program
00BA30	IO	00E650	System Data
	DRVR	000130	DEVICE= "our example"
	MICEMM4F	006AE0	DEVICE= my mouse driver
	ANSI	001180	DEVICE=
	LADDRV	000AE0	DEVICE=
		000820	FILES=
		000100	FCBS=
		003E70	BUFFERS=
		0008F0	LASTDRIVE=
		000CD0	STACKS=
01A090	MSDOS	000030	— Free —

Figure 1 -*A basic memory map and device driver chain as it appears on my CompuAdd 386-20 running DOS 4.01 with a 120-MB hard disk. The above map was obtained by running the DOS utility MEM with the /debug switch operable.*

THE RAW AND THE COOKED

Before continuing we should be aware that there are two types of device drivers used by DOS: character and block. They are fundamentally different and it's important that you understand that difference.

A device is defined as a character type if it is byte oriented. An example would be the typical printer port. All communication between the system and that device occurs on a character-by-character basis with the I/O being defined as "cooked" or "raw." In the "cooked" mode, DOS requests one character at a time from the driver and

[-1]	4 bytes-next driver pointer
[ ]	2 bytes-driver attribute word
[ ]	2 bytes-strategy routine offset
[ ]	2 bytes-interrupt routine offset
[ ]	8 bytes-device name

Figure **2**—*The components of a device header.*

buffers it internally. In the "raw" mode, DOS doesn't bother to buffer the data. Instead, requests for input are fixed-length character strings which are then passed directly to the driver with the return being made up of characters read by the driver. These character-type device drivers are assigned names (CON, AUX, LPT, etc.) that have a maximum length of eight characters. This constraint arises from the allowcd number of characters within the device header. We will discuss this header shortly.

Unlike the character device, the "block" device handles data blocks. An example is your tape backup drive or one of your disk drives. Each of these devices always transfers data in block sections. Device drivers which support these devices are assigned drive letters and become one or more of your system's logical drives (A:, B:, C:, etc.). As such, a single block-device driver can interface more than one hardware unit or map one unit into multiple logical devices (i.e., partitions).

CHAINS

DOS places each driver into system memory during the initial boot phase and chains them together to form a linked list. When an application program calls DOS Int 21h, DOS begins with the first driver in the chain and searches each, in a sequential fashion, until it finds a character or block driver with the appropriate name or designated logical drive letter. The NUL driver is **always** first in the list. It is then followed by the installable drivers loaded from CONFIG.SYS, followed by the internal DOS device drivers. But keep in mind, their memory location doesn't necessarily correspond to their position in the chain. The actual byte images of the installable drivers are loaded last after DOS's internal drivers. DOS just fudges the chain links so that the installable drivers come earlier in the search order than internal drivers. The advantage to this "organized" chain order, is that any of DOS's internal character device drivers can be replaced by an installable driver of the same name with the exception of the NUL driver. For example, ANSI.SYS can be used to replace the CON default system console driver so that any data streams directed to the standard output are routed through ANSI. On the other hand, block drivers cannot be replaced. DOS assigns logical drive letters to each block device as the driver is loaded from CONFIG.SYS on a "first-come, first-served basis" with its primary (DOS) block driver taking precedence over all others.

Much of this you have already experienced. Whenever you increased the capabilities of your system by adding a new board, you always had to add a new device driver. If you added a CD-ROM drive, a mouse, a local area network, or just a music synthesizer, this required the addition of a system interface: a device driver. But before we dig into the guts of a driver, it is important that you understand a little more about how drivers are involved in the basic DOS boot sequence.

FROM THE TOP

When you power up or reset a system based on the 8086 family of microprocessors, the microprocessor automatically starts program execution at location FFFF:0000h (a feature of the processor chip, not DOS). The ROM BIOS at FFFF:0000h instructs your system to jump to the beginning of the hardware test routines and the ROM bootstrap code. These hardware tests check the amount of installed memory and test which peripheral devices are available and operable. The ROM bootstrap initialization routine sets up some of the basic parts of the interrupt vector table which relate to hardware located by the POST. It then initializes the ROM BIOS tables at memory location 0400:0000h, refreshes the dynamic memory, institutes a search of the memory area between A000:0000 and F000:0000 in order to locate other ROM extensions (graphics ROM, etc.), and then marks them with a unique byte sequence which identifies them as ROM (see Figure 1). After all the found ROM extensions are initialized, the ROM bootstrap code finally starts the system itself.

The disk bootstrap code is now retrieved from the first sector (boot sector) of the system disk and executed. It looks back to the system disk for files IO.SYS and MSDOS.SYS (or IBMBIO.COM and IBMDOS.COM for PC-DOS), where IO.SYS must be

Bits FEDCBA98	76543210	Meaning
	1	Standard input
	0	Not Standard input
	......1.	Standard output
	......0.	Not Standard output
	.....1..	NUL device
	.....0..	Not NUL device
	....1...	Clock device
	....0...	Not clock device
	...1....	Special
	ooo.....	Reserved (set to zero)
	....	OPEN/CLOSE/Removable media supported
....1...	....	OPEN/CLOSE/Removable media not supported
...0....	....	Reserved (set to zero)
..1.....	....	IBM block format
..0.....	....	Other block format
.1......	....	IOCTL. supported
.0......	....	IOCTL not supported
1.......	....	Character device
0.......	....	Block device

Table 1 —*The Driver Attribute word.*

Offset	Length	Function
0000h	1 byte	length of the request header
0001h	1 byte	UNIT CODE: the device number for block devices
0002h	1 byte	COMMAND CODE: the number corresponding to the most recent command sent to the driver
0003h	2 bytes	STATUS: the status code word set by the driver after each call An error is indicated if bit 15 is set. If 0 then the request was completed successfully
0005h	8 bytes	reserved for DOS
000Dh	variable	data required by the driver

where the request header status word is defined as:

Bits FEDCBA98	76543210	Meaning
1.......	00000001	Write-protect violation error
1.......	00000010	Unkown unit error
1.......	00000011	Drive not ready error
1.......	00000100	CRC error
1.......	00000101	Bad drive request structure length error
1.......	00000110	Seek error
1.......	00000111	Unknown media error
1.......	00001000	Sector not found error
1.......	00001001	Printer out of paper error
1.......	00001010	Write fault
1.......	00001011	Read fault
1.......	00001100	General failure
.......1	....	Done
......1.	....	Busy
.00000 . .	....	Reserved
0	....	No error

Table 2—*The request header "Static Portion."*

the first entry in the root directory followed by the MSDOS . SYS file, especially since the file systems or disk structure utilities haven't been initialized yet. Information obtained from the BIOS parameter block (BPB located in the boot sector from byte 0Bh to 17h) provides enough information to locate and copy IO. SYS into low memory above the BIOS tables. Then either the boot program or IO. SYS will copy MSDOS . SYS into memory just above IO. SYS.

IO. SYS has two parts: a system manufacturer supplied BIOS component and a module called SYSINIT. The BIOS component, which contains the resident device drivers as well as hardware-specific initialization code, is started first. It checks the interrupt vector table for allowed hardware usage and then automatically deletes unneeded drivers. It then passes control to the SYSINIT module.

SYSINIT checks the available memory and relocates itself into high memory to make room for DOS, then copies MSDOS . SYS over (on top of)

```
;
; Driver: DRIVER.ASM
;
; globals
;
CR       EQU 0Dh                  ;carriage return
LF       EQU 0Ah                  ;line feed
MAXCMD   EQU 16                   ;max num of cmds for DOS 3.1
ERROR    EQU 8000h                ;SET ERROR BIT
BUSY     EQU 0200h                ;SET BUSY BIT
DONE     EQU 0100h                ;SET COMPLETION BIT
UNKNOWN  EQU 8003h                ;SET UNKNOWN STATUS
;
cseg     segment public 'code' ;start the code segment
         org    0                 ;zero origin
         assume cs:cseg,ds:cseg,es:cseg
```

We now set up the device header. I have chosen to define this as a "character type" driver with no special abilities. The attribute word is therefore set to 8000h. Refer to Table 1 for the attribute word settings.

```
;============================================================
;
;        driver header

DRVR_0   proc   far               ;set as a FAR procedure
         dd     -1                ;next driver pointer
         dw     8000h             ;attribute word
         dw     strategy          ;pointer to strategy
         dw     interrupt         ;pointer to interrupt
         db     'DRVR_0 '         ;device name
;
```

We must now set up the strategy routine. This calls for retrieving the segment and offset address ES:BX of the request header and saving them in rh_ES and rh_BX.

```
;
;____________________________________________________________

;        strategy routine
;
rh_ES    dw     ?                 ;RH segment register
rh_BX    dw     ?                 ;RH offset address
strategy:
         mov    cs:rh_ES,es
         mov    cs:rh_BX,bx
         ret
```

We will now set up the interrupt section starting with the "command branching table."

```
;============================================================
;
; Interrupt section: command branching table
;
d_tbl:
         dw     s_init            ;initialization
         dw     s_mchk            ;media clock
         dw     s_bpb             ;bios parameter block
         dw     s_ird             ;IOCTL read
         dw     s_nrd             ;read
         dw     s_inst            ;input status
         dw     s_infl            ;flush input buffer
         dw     s_write           ;write
         dw     s_vwrite          ;write with verify
         dw     s_ostat           ;current output status
         dw     s_oflush          ;flush output buffers
         dw     s_iwrt            ; IOCTL write
         dw     s_open            ;open
         dw     s_close           ;close
         dw     s_media           ;removable media
         dw     s_busy            ;output until busy
```

And we must now save the system's state and all of its registers on a stack.

```
;
```

(continued)

Listing **1**—*The* **first** *part of the assembler code includes instructions to the* **assembler** *and 'housekeeping' code to set up the initial state of the device driver.*

the IO.SYS initialization code. It then calls the MS-DOS (or PC-DOS) initialization code. The MS-DOS internal files are set up along with the appropriate interrupt vectors. **MSDOS.SYS** then starts initializing drivers by checking their status, initializing their hardware, and setting up the interrupts serviced by each driver. The BIOS block is also examined to determine the number of disk drives attached to the system and the largest disk sector size which is used to determine the disk sector buffer. Finally, MSDOS.SYS displays the DOS copyright message and returns control to **SYSINIT.**

Next, SYSINIT uses the DOS Int 21h file services utility to open the CONFIG.SYS file and allocate memory for disk buffers and file control blocks. Any referenced drivers are then loaded, initialized, and added to the chain list of drivers (see Figure 1). These drivers are processed and placed into memory in the order in which they appeared within CONFIG.SYS.

As each driver is loaded, SYSINIT calls the driver's "init" function to verify that its associated devices are present and operational. The driver then notifies SYSINIT of the address of the next higher block of available memory which is to be used as the physical location of the next device driver. As each driver is loaded, SYSINIT forms a linked list by filling each device header with the segment:offset address of the next driver in the chain, whereby the last driver is identified by the double word value FFh (-1). If new character drivers and resident drivers have the same name, the new drivers are listed so that they are found first when requested.

SYSINIT next closes all file handles and opens the console device (CON), the printer device (PRN), and the auxiliary device (AUX). SYSINIT then loads and executes **COMMAND.COM** which holds the necessary code for internal commands and for batch-file processing. After **COMMAND.COM** sets up the vectors for interrupts 22h through 24h, **COMMAND.COM** executes the AUTOEXEC.BAT file. Control is then transferred to the transient portion of COMMAND.COM and the DOS prompt is displayed. Now we're ready to mess up and crash the system.

TABLES AND ROUTINES

A driver is a relocatable memory image similar to a .COM file. But instead of being an executable program, a driver contains a special set of tables and routines that implement a device or "some characteristic behavior" required by the interface software. The order of the tables and routines is specific. This enables them to be recognized by DOS. As such, the driver is broken into three components: the device header, the strategy routine, and the interrupt routine.

Unlike a typical .COM program which uses the assembler ORG directive to start assembly at byte 256 (in order to leave room for the DOS Program Segment Prefix), a driver is ORGed at 0 with the device header being the first thing in the file. This is an 18-byte area divided into five fields (see Figure 2).

This header defines the offsets to otherimportantpartsofthecode.DOS therefore uses it to find out the device name and where the support routines reside. The first four bytes are filled in by DOS when it links the driver into the system chain. They are initialized to -1 (FFFFFFFFh) and then loaded with the pointer to the next driver in the list when installed. The last driver in the list is marked with -1. The next two bytes are called the driver attribute word. Its function is to specify thedrivercharacteristics.Asshownin Table 1, the attribute word has bi ts for character device, block devices, standard input and output (CON devices), NUL, CLOCK, and finally IOCTL (for program I/O device control calls). The offsets for the strategy and interrupt routinesareused by the system to find the appropriate procedures when it wants to use the driver. And finally, thedevicenameisan B-character, left-justified, blank-filled device name. If you are implementing a new device, make sure this name does not conflict with any of the old ones. If it is the same as an old one, your new driver will replace the existing device.

```
;        interrupt section: state save

interrupt:
         cld
         push  es
         push  ds
         push  ax
         push  bx
         push  cx
         push  dx
         push  si
         push  di
         push  bp

;        retrieve the rh pointer

         mov    dx,cs:rh_ES
         mov    es,dx
         mov    bx,cs:rh_BX

         mov    al,es:[bx]+2    ;find the command code,
                                ; offset+2bytes
         xor    ah,ah
         cmp    ax,MAXCMD       ;is it within our range
         jle    ok
         mov    ax,UNKNOWN      ;outside known command table
         jmp    finish


ok:
         shl    ax,1
         mov    bx,ax
         jmp    word ptr [bx t d_tbl];jump to the
                                ; appropriate  function
;

Once the task is completed the status flag must be set and all
the registers restored.

finish:
         mov    dx,cs:rh_ES
         mov    es,dx
         mov    bx,cs:rh_BX
;
         or     ax,DONE
         mov    es:[bx]+3,ax

         pop    bp
         pop    di
         pop    si
         pop    dx
         pop    cx
         pop    bx
         pop    ax
         pop    ds
         pop    es
         ret                    ;back to DOS

;
; driver commands, eliminate unused commands thru an error exit

s_mchk:
s_bpb:
s_ird:
s_read:
s_nrd:
s_inst:
s_infl:
s_vwrite:
s_ostat:
s_oflush:
s_iwrt:
s_open:
s_close:
s_media:
s_busy:
         xor    ax, ax
         jmp    finish
```

listing 1-continued

```
;
;       DRIVER  initialization
;
ident:
        db      CR,LF
        db      'Circuit Cellar Test Driver Shell V'
        db      '0.0'
        db      CR,LF,LF,'$'

keyboard dw     ?
        dw      ?

s_init:
        mov     ah,9            ;display string
        mov     dx, offset ident
        int     21h             ;universal DOS function
        cli                     ;clear interrupt flag
;READ Keyboard interrupt vector
        mov     bx,cs
        mov     ds,bx
        mov     al,16h          ;keyborard I/O vector
        mov     ah,35h          ;get interrupt vector
        int     21h             ;universal DOS function
        mov     keyboard[0],bx  ;offset
        mov     keyboard[2],es  ;segment

;Store the keyboard interrupt vector in INTERRUPT 7F

        mov     ds,keyboard[2]
        mov     dx,bx
        mov     al,7fh          ;vector, not used
        mov     ah,25h          ;set interrupt vector
        int     21h             ;write to vector 7fh, [1fc+3]

;GET the CLOCK INTERRUPT VECTOR

        mov     bx,cs
        mov     ds,bx
        mov     al,01ah
        mov     ah,35h
        int     21h
        mov     keyboard[0],bx  ;offset
        mov     keyboard[2],es  ;segment

;SET the KEYBOARD interrupt vector TO the CLOCK
; interrupt vector

        mov     ds,keyboard[2]
        mov     dx,bx
        mov     al,16h
        mov     ah,25h
        int     21h             ;keyboard io vector to clock
                                ;now retrieve the RH pointer
        mov     dx,cs:rh_ES
        mov     es,dx
        mov     bx,cs:rh_BX
;
        lea     ax,end_driver   ;get the end of driver address
        mov     es:[bx]+14,ax
        mov     es:[bx]+16,cs

        xor     ax,ax           ;zero the ax register
        jmp     finish
;
;
; write data

s_write:
        xor     ax,ax
        jmp     finish

; End of Driver
;
end_driver:
DRVR endp
cseg ends
        end
```

Listing **2**—*Driver initialization is important, since it is the only section that can* **legally `call`** *DOS functions, and is called once then 'thrown away.'*

If the device is a block type, the first byte in the device name field becomes the number of devices associated with the driver and all other bytes are ignored. For documentation purposes you can stick the name of the driver in these locations anyway. It helps you recognize the device driver when looking through memory with the debugger.

STRATEGY ROUTINE

The second part of a device driver is the Strategy Routine. Why it's called that is beyond me. It has little to do with what is usually considered "strategy." But it is short and sweet, about five lines long. Its purpose is to "remember" where in memory the operating system has assigned the location of the device's request header (RH). This RH has two basic functions. The first is to provide an area for DOS's internal operations and the second is to provide a communication area in which DOS commands to the driver and the driver response are passed. For example, whenever a driver is requested to output data, the data address is passed through the RH. The driver performs the output task and the sets a status flag within the RH to inform DOS that it has completed that task.

When installed, the RH is built into a reserved area of memory and its address is passed to the strategy routine in ES:BX. It is of variable length but always has a fixed 13-byte header called the "static portion." The structure of the request header is shown in Table 2.

INTERRUPT ROUTINE

And finally we come to the "true heart and soul" of the driver: the interrupt routine, the work horse of the device driver. But in all honesty, I'm confused again by "whoever named these utilities." It surely doesn't act like an interrupt, nor does it end with IRET. It ends with a RET. Maybe his brain was in an interrupt mode? Anyway, onward and upward!

The interrupt routine may contain as many as 20 different functions required by DOS to process all of the driver requests. These functions are specified by examining the byte at offset 0002h of the RH. A reference table of pointers to each of the driver's functions is then used to process the request. This table (command branching table) is easy to lay out using assembly language since ASM keeps track of the functions and automatically inserts correct offsets into the table. It provides a branch point for the driver so that the appropriate function can be activated.

THE BASIC PROCESS

The device driver starts a process by saving the "current" machine state in a stack, grabbing the pointer to the request header, determining its "current" command by examining the offset 02h within the RH, finding the appropriate branch pointer, and then jumping to the location where the function code resides. When it completes a function, the driver retrieves the pointer to the request header and sets the "done" bit within the status word. The registers saved at the beginning of the driver are then restored and control is returned to the DOS kernel. Simple! Right?

AN EXAMPLE DEVICE-DRIVER SHELL

Well, we'd all like to think so. Unfortunately I've glossed over some important details, so I'll try to fill in the gaps. I'd like to discuss a simple device driver shell which you can use as a basis for your own creations. It is a program that I modified from a listing given in the book, "DOS Programmer's Reference" by Terry R. Dettmann, Que Corporation, Carmel, Indiana. Hopefully you can create a functional version of your driver by working with this code also. Be warned, this driver doesn't do much—I didn't want it to. Its purpose was to provide me with a window into the DOS boot sequence so I could mask out the keyboard (trap all CRTL-BREAKs, etc.). That mask interrupt handling sequence is included within the example. For **more details on** interrupt handling, I refer you to my article "Software at the Hardware Level: Programming TSRs for Interrupt Handling" in CIRCUIT CELLAR INK #21.

Listing 1 shows the sampledriver. I will attempt to fill in as much detail and documentation as possible at various points within the listing.

The first part of the actual driver coding is made up of instructions to the assembler. It contains several definitions which makes overall programming easier. I used Microsoft MASM for my code development.

The driver initialization function must be present. It is used to set the address of the end of the driver into byte offset 0Eh of the request header. It is also used to check for the presence and operability of the interfaced device. It is the first routine called by DOS and typically the last routine within the driver code. This is because DOS calls this routine only once and you can throw it away after that call completes.

```
          TITLE kbres
cseg      segment
          assume cs:cseg,ds:cseg
          org    100h
start:    jmp    begin
keyboard  dw     ?
          dw     ?
begin:
          mov    bx,cs
          mov    ds,bx
          mov    al, 7fh
          mov    ah,35h
          int    21h
          mov    keyboard [0], bx     ; offset
          mov    keyboard[2],es       ; segment
          mov    dx,bx

          mov    ds,keyboard[2]
          mov    dx,bx
          mov    al,16h
          mov    ah,25h
          int    21h        ; reset keyboard_io vector 16h

          mov    ax,4C00h             ; end
          int    21h
;
cseg      ENDS
          END    start
```

Listing 3—*An ASM listing of KBRES used to reset the keyboard interrupt vector in the AUTOEXEC.BAT procedure.*

I used the initialization process to insert my interrupt handling procedure into the DOS boot-sequence. Since it is the only section that can call DOS functions legally, it's ideal for this purpose. In my application, I had the **init** procedure do the following:

1. output a device ID line to the video monitor
2. read the keyboard interrupt vector using the Int 21h instruction
3. write that vector to location 7Fh within the vector table (hopefully an unused location)
4. misdirect the keyboard interrupt to a clock request interrupt by setting the keyboard interrupt vector to the clock vector. Each time someone hits a key while the keyboard is rerouted, the clock buffers are reset. But who cares?
5. exit the **init** part of the driver, and basically the driver itself. The keyboard will now ignore any keystrokes until it is revectored by my code running under AUTOEXEC.BAT.

ASSEMBLING THE DRIVER

To make the code from the listings work, you need to run your macro assembler, linker, and then the EXE2BIN program to create the driver as a binary image. I wrote a simple .BAT-type file to handle this for me and called it MK.BAT. For MASM this sequence takes on the form:

```
echo off
masm %1;
link %1;
exe2bin %1 %1.sys
```

where the sequence was invoked by

```
MK driver
```

which creates a DRIVER.OBJ, DRIVER.EXE, and then the desired DRIVER.SYS file.

To test the above driver you need to install it in your CONFIG.SYS file by entering the command,

```
DEVICE=C:\DRIVER.SYS
```

which will load the driver into the device chain during the boot phase. But since this will mask out your keyboard, you need to run KBRES.COM from AUTOEXEC.BAT so the keyboard vector is restored (see Listing 3).

Once you've added the device driver to the CONFIG.SYS file and set KBRES.COM to run from your AUTOEXEC.BAT procedure, you can tempt fate and do a reboot of your system so the new DRVR will be initialized. If all goes well, you'll see no change in your boot sequence except an added display line. If your code fails, then your keyboard will more than likely be locked out. I therefore suggest that you keep a system disk handy.

If you're successful, I suggest you go through the following sequence to examine the driver chain so you can see some of the things we talked about previously.

INSPECTING THE DRIVER CHAIN

A simple procedure for mapping out your system's device chain is possible using DOS's DEBUG utility. Just start DEBUG and type,

```
A
MOV  AH,52
INT  21
RET
```

then press Enter once (by itself) to terminate the assembly process. Next, type

```
G 0104
```

from the debug prompt. Now record the values of the ES and BX registers. Then type

```
D  ES:BX
```

substituting the actual segment value ES and offset address BX in the above directive. At this point you should see the characters "NUL"displayed somewhere on the right side of your screen. Now count back 10 bytes from the location of "N" in "NUL" and you will find the beginning of the NUL device header and the address of the next driver in memory. Now dump the bytes at these addresses. Doing this successively will enable you to map the location and extent of each driver in the chain and see exactly how DOS stacks them.

ONWARD AND UPWARD

I hope I've been able to convince you that the devicedriver isan important and powerful feature of DOS. Although the above description was too brief and designed only for a basic overview, I hope you have come to realize that driversallow manufacturers or software designers to incorporate their systems into the DOS environment in a uniform manner. It's obvious that the driver has a wide degree of flexibility in application. I believe DOS users would be well served if both hardware and software types paid more attention to the possibilities inherent in loadable device drivers.

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

IRS

401 Very Useful
402 Moderately Useful
403 Not Useful

FEATURE ARTICLE Part 1

Steven E. Strauss & P.K. Govind

ISDN (S/T) Interface

General Review of Functional Concepts

The telecommunications network is migrating from an analog to a digital network, transporting digitized voice and data on subscriber loops that connect the switching network to customer endpoints Digital telephones, facsimile machines, and integrated voice/data workstations are examples of customer endpoints, commonly known as Terminal Endpoints (TEs), in the architectural model of an Integrated Services Digital Network (ISDN). An ISDN has four elements: information services, channels, interfaces, and message sets, conforming to international standards for information exchange.

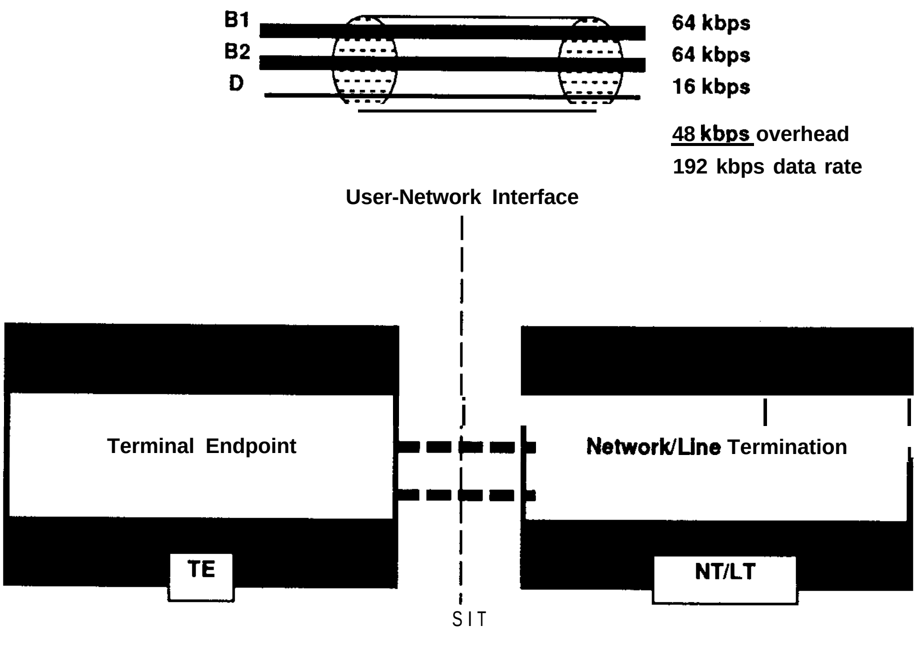


Figure 1 —*The 'basic" service in ISDN is provided via two 64-kbps* B channels *and a 16-kbps D channel*

The 'basic" service in ISDN is provided via two B channels and a D channel. The bit rate on each B channel is 64 kbps. The bit rate on the D channel for this type of service is 16 kbps. This "2B+D" service, shown in Figure 1, gives the user a 144-kbps digital pipe to transport information through the network. In this article, we will first review the functional principles of the basic access ISDN interface standard that allows information transfer on separate transmit and receive paths. Later, we'll describe a design example of a plug-in interface board that operates in a PC.

REFERENCE MODEL AND STANDARDS

ISDN standards have been defined and documented by the CCITT (Consultative Committee on International Telegraph and Telephone) and the American National Standards Institute (ANSI). CCITT I-series Blue Book documents define several reference pointsand functional groups (Figure 2). The "S" reference point is a subscriber-side demarcation point for basic access at 192 kbps.

The network end of a subscriber service loop is called an NT (Network Termination). It may be partitioned into two parts-NT1 and NT2—to connect the subscriber to the network. NT2 provides the subscriber-side connection. NT1 provides access to the network. NT2 can provide switching functions (e.g., a PBX, a terminal controller, etc.), whereas an NT1 cannot. NT1 provides physical link (bit level, Layer 1) multiplexing only. The "T" reference point separates the NT2 and

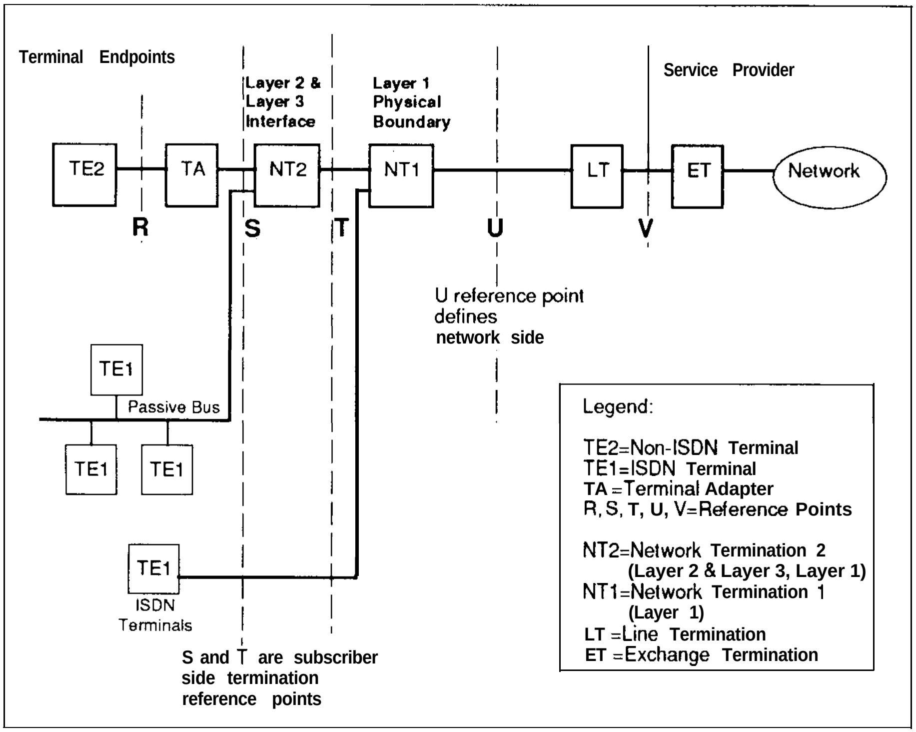


Figure 2—*CCITT* **I-series** *Blue Book documents define several reference points and functional groups.*

NT1 functional groups. Phones and integrated voice/data terminals can connect to the network at the S or the T reference point. Point-to-point and point-to-multipoint (passive bus) connectionsarepossible.Since theelectrical interface is identical at the S or T reference point, the symbol "S/T" is frequently used in ISDN literature. The "U" reference point identifies the network side transmission interface. The line termination, LT, and exchange termination, ET, are usually located in the phone company switching office.

Thecommunicationprotocol standards used in ISDN apply to the lower three layers of the 7-layer Open Systems Interconnection (OSI) model, defined by the International Standards Organization (ISO). The connection control protocol standards for the three layers pertinent to our discussion are shown in Figure 3:

Layer 7: I.430/ANSI T1.605—applies to (2B+D) transport; a hardware function

Layer 2: I.441 (Q.921 &-applies to the D channel; a software/hardware function

Layer 3: 1.451 (Q.931)-applies to the D channel; a software function

The purpose of these standards is to provide a set of integrated voice and data "bearer" (Bchannel) services via thecall control proceduresof the D channel. The D channel enables the user to request services through the set of three layered protocols. Signaling over the D channel can provide fast call setup, look ahead for busy, incoming call screening with automatic number identification, and so on.

LAYERED ARCHITECTURE

Layer 1 defines the physical link characteristics. It includes bit transport timing and electrical characteristics of the 2B+D access interface. It also provides contention resolution, which allows multiple terminals connected to the same NT to send messages on the D channel.

Layer 2, the Link Access Procedure for the D-channel (LAPD) has three functions. First is message frame processing, that is, converting messages between a serially transmitted HDLC (High-level Data Link Control) format and a computer memory data structure. Second is procedural error control and flow control of message traffic in the D channel. Third is the terminal identifier (TEI) and serviceaccesspointidentifier(SAPI)management, which provides the capability to distinguish between message traffic from different terminals.

Layer 3 defines the content of messages in the D channel and provides the capability for negotiating services with the network. This protocol includes functions like B channel call control, as well as data-oriented

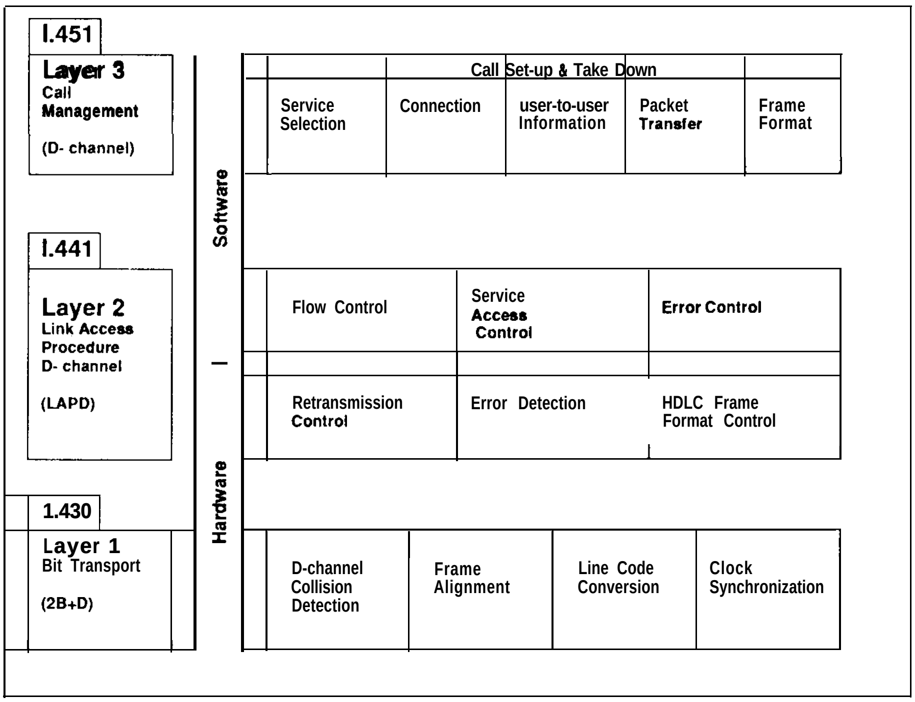


Figure 3a—*The protocol layers of the ISDN basic rate interface (BRI).*

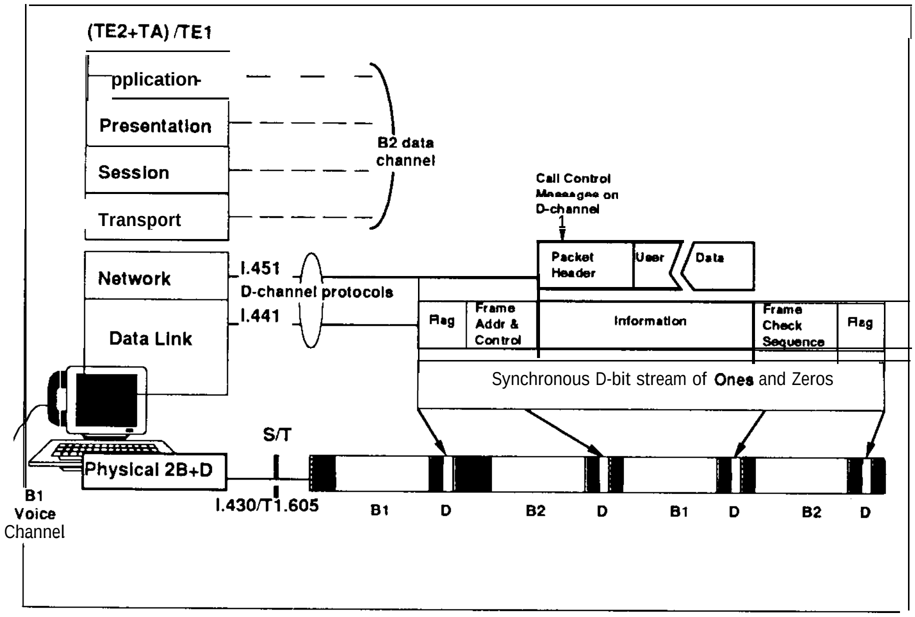


Figure 3b—*The frame composition of the basic rate interface.*

services like packet switching via the B or the D channel.

FRAME STRUCTURES

The CCITT 1.430 and the ANSI T1.605 standards describe the physical link (Layer 1) characteristics of the Basic Rate Interface (BRI) at the S or T reference point between TEs and NTs. Specifications include voltage levels, impedance templates, bit timing, and coding. The frame structure defined by 1.430 has the following characteristics (see Figure 4):

•192-kbps transmission on separatereceiveand transmit twisted pairs, with 144 kbps for user 2B+D channels and a 48-kbps overhead for framing control, link maintenance, and synchronization

- 48-bit frames in 250 µs (i.e., 4000 frames per second)

*Alternate space inversion (pseudo ternary) line code, 750-mV peak signal

*Echo back to the TE of the D channel received by the NT

- l'assive bus for point-to-multipoint operation

The NT-to-TE and TE-to-NT frames have different formats, since the responsibilities of the TE and NT are different, especially when using a multiframemaintenanceprocedureor a passive bus configuration.

The line encoding uses electrical signals of alternating polarity to main-

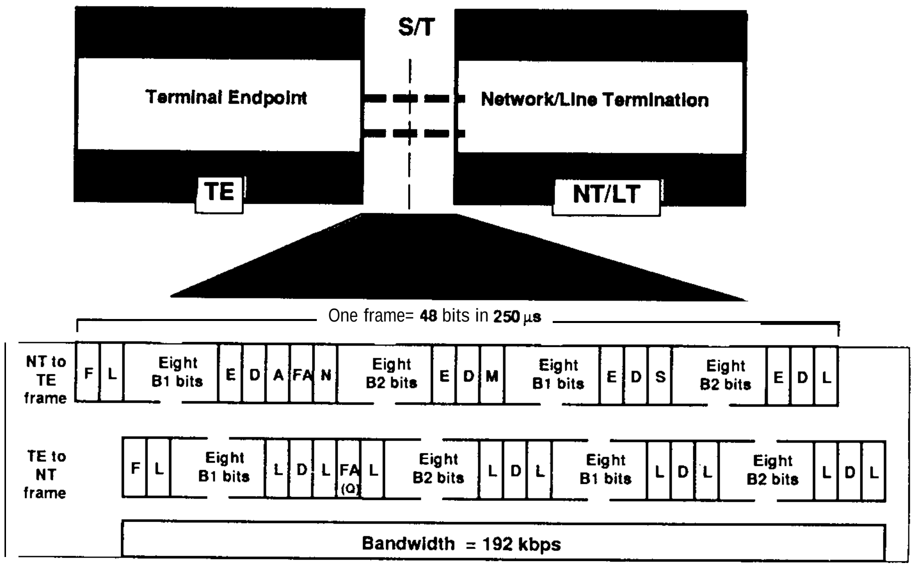


Figure **4**—*The frame structure of a 2B+D basic rate interface at the S/T reference point.*

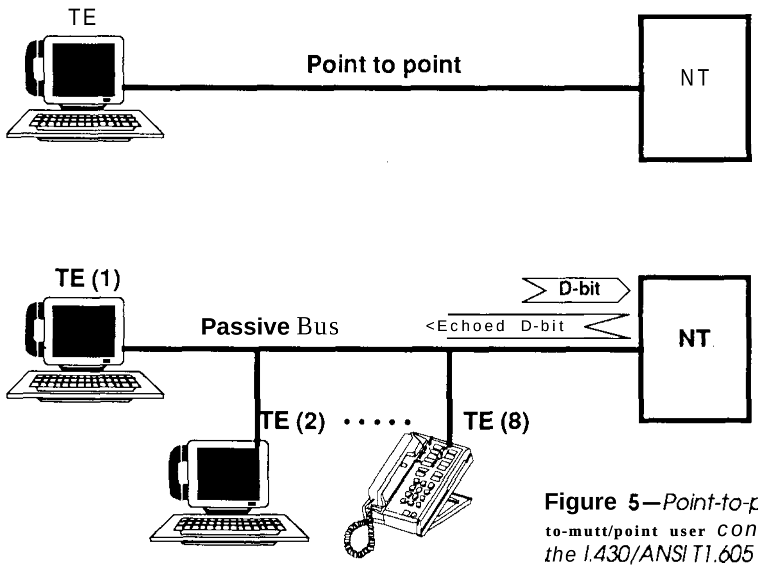


Figure 5—*Point-to-point* **and** point-to-mutt/point user *configurations of the I.430/ANSI T1.605 standard.*

tain a DC **balance.** In NT-to-TE frames, the L bits are used to electrically balance the entire frame. This prevents DC level wander. In the TE-to-NT frames, L bits balance each octet of B channel data and each individual D bit. This is done to avoid line code violations. However, deliberate code violations at specific bit positions are introduced to mark frame boundaries. Line code violations are used to establish frame synchronization.

WIRING ARRANGEMENTS

Two types of wiring configurations are possible: point-to-point and point-to-multipoint (passive bus), as shown in Figure 5. In a passive bus configuration, up to eight terminals could be connected to a BRI. All terminals share the two B channels and the single D channel.

On a passive bus, all the TEs contend for the same D channel while monitoring the D bits being echoed back from the NT. When a given TE sees that the echoed D bits coming back from the NT are different from the D bits it had sent to the NT, it knows that a collision has taken place and it must try again. An access mechanism is built into each TE that prioritizes the use of the D channel in such a way that each TE has equal access to the D channel. The priority mechanism is described in detail in the standards document and is beyond the scope of this article. Semiconductor devices implementing the 1.430 standard take care of the details of the priority algorithms, relieving the user of such burden. An example of a device with a built-in split reservation system for D channel access is the AT&T T7250B, tailored for TE applications in ISDN. We will highlight the use of this device in Part 2.

LAYER 1 MAINTENANCE

The ANSI T1.605 standard for the U.S. environment calls for the use of extra channels to provide Layer 1 maintenance on a BRI loop. Maintenance messages are provided for the TE-to-NT direction (Q-bits) and for the NT-to-TE direction (S bits). The Q

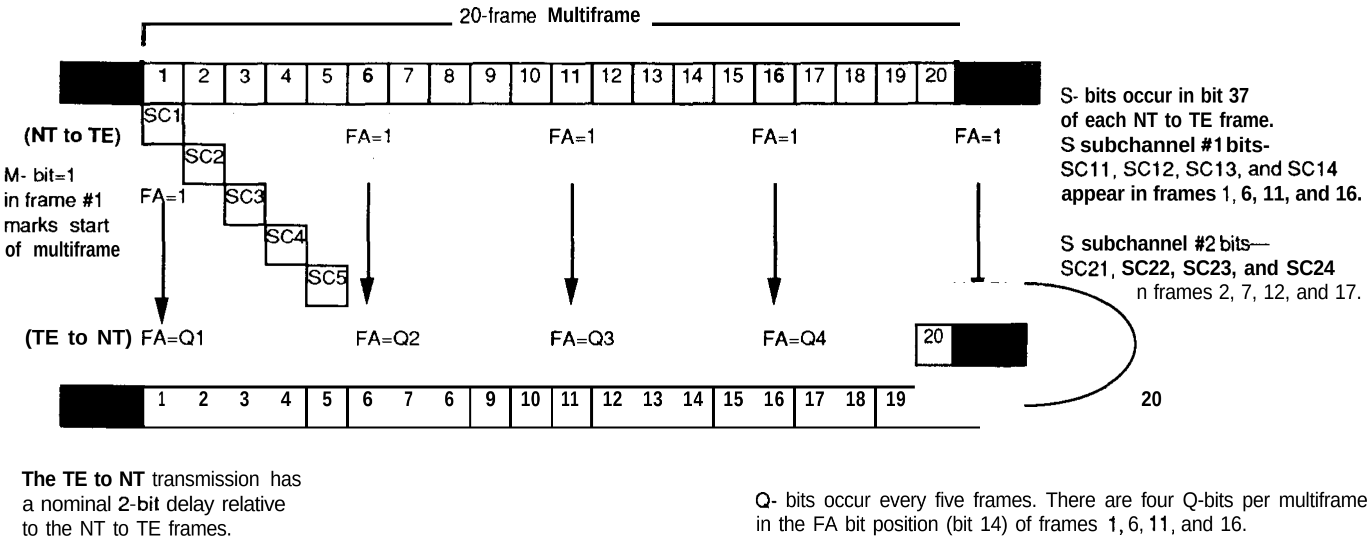


Figure 6—*Multiframing and the S-bit and Q-bit channels for Layer 1 maintenance.*

channel message is conveyed to the NT in 4-bit codes every 20 frames, which make up a multiframe (shown in Figure 6).

Multiframing is established by setting the Mbit, which is the 26th bit in the NT-to-TE frame (see Figures 4 and 6). Messages are sent from the NT in the S-bit channel and from the TE in the Q-bit channel. Five S-bit subchannels are defined in the T1.605 standard: SC1, SC2, SC3, SC4, and SC5. Each subchannel uses four bits to convey a message per multiframe. Since only one bit is used per basic 48-bit frame, it takes 5 ms (20 x 250 µs) to collect a 4-bit code.

The T1.605 standard covers the use of S-bit subchannel SC1 only. Unused S-bit subchannels (SC2-SC5) are set to binary zeros. Recently, the use of S-bit subchannel SC2 has been defined for adoption by the standards bodies to convey the U interface messages to the TE in an NT1 application

The Q-bit channel in the TE-to-NT frame (see Figures4 and 6) uses the FA bit position of every fifth frame to provide a Q-channel message from the TE to the NT as a 4-bit code. The coding and the use of the 4-bit messages is tabulated in the T1.605 document.

COMMUNICATION BETWEEN THE TE AND NT

A TE and an NT establish a line connection between them in three stages: Activation, Communication, and Deactivation (see Figure 7). The activation stage allows the two ends to get synchronized, allowing them to communicate. The communication stage sets the two ends in a fully operating mode. The deactivation stage disconnects the call and allows the two ends to idle in a low-power mode of operation or cease all operations.

Layer 1 signals with specified meaning and coding are called INFO signals. INFO 0, INFO 2, and INFO 4 are sent by the NT and received by the TE. INFO 0, INFO 1, and INFO 3 are sent by the TE and received by the NT. An inactive link is characterized by INFO 0 signals at both ends. A TE has the option to activate the link by sending INFO 1. An NT can activate the link by sending INFO 2. INFO 1 and INFO 2 have distinct bit patterns. Once communication is established between

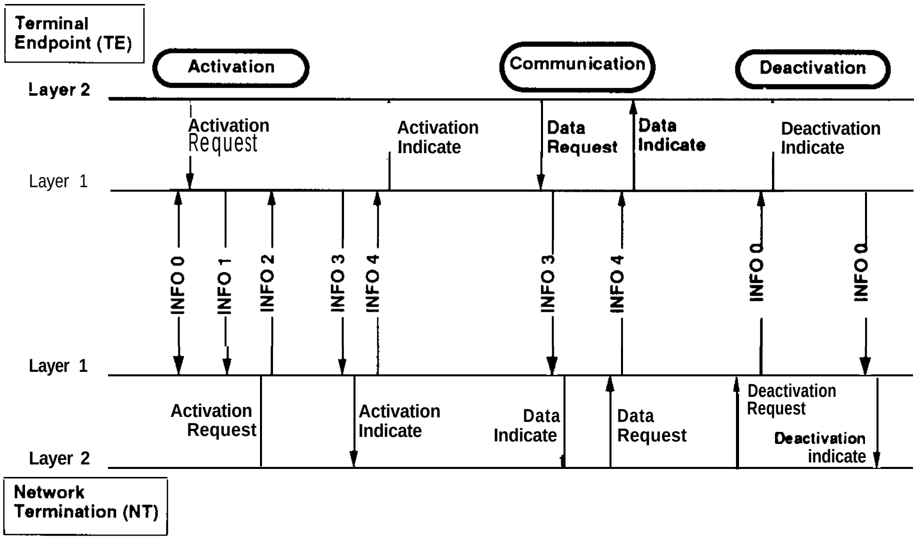


Figure 7—*The handshake sequence between the TE and the NT link.*

the NT and the TE, INFO 3 and INFO 4 signals carry frames with user data on the B and D channels.

In Part 2, we will describe the circuit design of an ISDN BRI interface, using newly introduced semiconductor devices, optimized for the TE application. ✣

Steven Strauss *is licensed Professional Engineer and a member of the technical staff at AT&T Bell Laboratories in Allentown, Pa., specializing in communication devices. He holds a B.S.E.E. from Pennsylvania State University and an M.S.E.E. from Rensselaer Polytechnic Institute in Troy, N.Y.*

P.K. Govind is a distinguished member of the technical staff at AT&T Bell Laboratories in Allentown, Pa. He is an application consultant for communication devices. He has extensive experience in product planning, system integration, and product development. Govind received an M.S. and Ph.D. in physics from the University of Colorado in Boulder, Colo.

REFERENCES

1. CCITT Blue Book, Volume III, Fascicle 111.8, Integrated Services Digital Network (ISDN), Overall Network Aspects and Functions, ISDN User-Network Interfaces, Geneva 1989, International Telecommunication Union.
2. ANSI T1.605-1989 ISDN basic access interface for S and T reference points. American National Standards Institute, NY.

IRS

404 Very Useful
405 Moderately Useful
406 Not Useful

FEATURE ARTICLE Part 2

S-ART: Building the Network Software

Dun **Burke &**
John Dybowski

Part one of this two-part article described S-ARTnet, a binary control network. This second part will describe S-ARTboy, a PC program that can be used to manage and view S-ARTnet. S-ARTboy is written in Borland C++. Communication with S-ARTnet is implemented using the Greenleaf communications library. Screen management is handled by Trans Atlantic Software's Screenman package. One interesting part of this article is a discussion of using C++ to hide the complexities of such third party C language packages by developing an object-oriented interface to the package. I'll discuss briefly the functionality of S-ARTboy and then examine the communications module to illustrate how simple communications become using the object-oriented approach. I'll also touch on techniques for using idle time waiting for keystrokes for performing system functions.

FUNCTIONALITY

The main S-ARTboy screen, the management screen (Figure 1), allows the user to assign names to points on the network, both input and output. The default name for each point consists of an ID of the form mm-n where *mm* is the number of the S-ART module (O-291, and *n* identifies the specific point (0 and 1 for outputs, 2 and 3 for inputs). ID 00-2 identifies input 1 on module 0, and might be named TEMP1 for a temperature sensor. ID 00-0 identifies output 1 on module 0 and might be assigned the name BIG-FAN. The user can specify for each point whether it is active or inactive and can toggle the display of inactive points on and off. The current state, on or off, of each active point is displayed in real time, along with the length of time each point has been in its current state. Finally, the user can specify whether history on the state of each point should be recorded and if so whether state transitions or periodic samples should be recorded.

Output points can be controlled from the management screen. By highlighting a point and pressing Alt-0, the state of the point is toggled. The user can also bring up a history screen for a point by highlighting the point and pressing Alt-I' (Figure 2). The last 16 transitions or samples are displayed for the point, along with durations.

From the management screen, Alt-C brings up the Create Script screen (Figure31 which is used to automate the control of output points in the network. A script consists of several conditions, a target output point, the desired state of the output, and a link to another script. Conditions that can be entered include:

● Current date and time is equal to specified date and time.
*Current date and time is greater than specified date and time.
● specified input point is in an on state.
*Specified input goes from off to on.
@Specified input goes from on to off.
*Specified input is off.
•Specified input is on.

If the specified conditions are met, a command is sent to S-ARTnet to control the specified output point. Blanks for date, time, or source point evaluate to true. If the conditions are true, the script specified in the "next" field (if any) is executed after the point has been set. Most of these fields are filled in by selecting from a pop-up list of options. This ensures that only valid data is entered.

Using this mini script language the user can enter scripts such as

```
At 13:OO if Templ
   is on turn big fan on
After 21:30 if Alarm1
   goes on turn floodlight on
      if switch1 goes on turn lights1 on
      if switch1 goes off turn lights1 off
```

```
S-ARTboy                                    Wed Mar 13 21:31:02 1991

                               MANAGEMENT

                                           FREQUENCY          DURATION
ID          NAME               ACT  HST    HH:MM:SS   STATE   HH:MM:SS
------
00-0     Big Fan                Y    T     00:00:00    OFF    48:36:25
00-1     Heater                 Y    F     00:00:00    ON     48:36:14
00-2     Templ                  Y    F     00:00:00    OFF    47:45:26
00-3     Switch A               Y    F     00:00:00    ON     50:10:39
01-0     Switch B               Y    T     00:00:00    ON     20:19:04
01-1     Light                  Y    T     00:00:00    ON     47:45:26
01-2     Dan's widgit           Y    T     00:00:00    OFF    01:10:09
01-3     None                   Y    T     00:00:00    OFF    00:00:00
02-0     None                   Y    T     00:00:00    OFF    00:00:00
02-1     None                   Y    T     00:00:00    OFF    00:00:00
02-2     None                   Y    T     00:00:00    OFF    00:00:00
02-3     None                   Y    T     00:00:00    OFF    00:00:00
03-0     None                   Y    T     00:00:00    OFF    00:00:00
03-1     None                   Y    T     00:00:00    OFF    00:00:00
03-2     None                   Y    T     00:00:00    OFF    00:00:00
03-3     None                   Y    T     00:00:00    OFF    00:00:00
04-0     None                   Y    T     00:00:00    OFF    00:00:00

Alt-K for Keystroke Help; Alt-F for Field Help; ESC to Exit
```

Figure 1—*The S-ARTboy* management screen.

```
S-ARTboy                                    Wed Mar 13 21:31:18 1991

                     HISTORY FOR BIG_FAN

             DATE        TIME      STATE   DURATION
           mm dd yy    hh mm ss            hh mm ss
           02/10/91    20:54:37    OFF     48:36:41
           02/10/91    19:19:30    ON      01:35:07
           02/10/91    19:19:21    OFF     00:00:09
           02/10/91    19:19:20    ON      00:00:01
           02/10/91    19:19:18    OFF     00:05:02
           02/10/91    18:44:31    ON      00:34:47
           02/10/91    18:42:30    OFF     00:02:01
           02/10/91    18:42:28    ON      00:00:02
           02/10/91    18:42:16    OFF     00:00:12
           02/10/91    18:41:58    ON      00:00:18
           02/10/91    18:41:56    OFF     00:00:02
           02/10/91    18:41:53    ON      00:00:03
           02/10/91    18:41:51    OFF     00:00:02
           02/10/91    18:21:41    ON      00:20:10
           02/10/91    18:21:34    OFF     00:00:07
           02/10/91    18:21:31    ON      00:00:03

Press Alt-H for Help or ESCAPE to Return to Management Screen
```

gure 2—*The history screen shows an accounting of all past activity.*

COMMUNICATIONS

S-ARTboy continually polls the network for the state of all input points and compares each point with its last known state. Whenever a change of state is detected, the new state and current time are stored to be used to update the management screen and the history screen.

Polling the network is implemented using the Greenleaf communications package. The interface to Greenleaf has been greatly simplified by creating a C++ port object which provides only those services required by S-ARTboy. In C++ it is possible to create classes and objects which are instances of those classes. A class is analogous to a data type in that it includes data and a set of operations, called member functions, that may be performed on the data. The magic word is encapsulation: the data and the access to the data are balled up together (encapsulated) in the class. An integer, for example, is a standard data type consisting of an implementation-defined number of bytes that can hold an integer value. The declaration "`int i`" creates an instance of data type integer. You can do a bunch of things to a variable `i`, like get its value, set its value, add it to another integer, and so forth. In like manner you could define a new integer-like class, new_int, with member functions to get the value, assign a value, and increment. The declaration "`new_int new_i`" would create an object of the `new_int` class. An attempt to perform an operation on `new_i` such as decrementing it or adding it to another new_int would be illegal in C++. Only access via member functions is permitted.

S-ARTboy defines class `port` (Listing 1) to handle the serial port and communication with S-ARTnet. Several objects of class port can be created. S-ARTboy creates one for communication with S-ARTnet. Num is the serial port in use and status holds the current status of the port. The keyword private indicates that this data is private to the port object and cannot be seen or manipulated except through the member functions. Member functions port, gets, puts, and get_status are declared public, indicating that they can be invoked from outside the port object to operate on the port. Puts sends a request for network status to the S-ARTnet controller. `Gets` gets the requested status of network points. Function port is special: the constructor. Every class has a constructor which has the same name as the class and is automatically invoked whenever an object is created. At run time, S-ARTboy creates the port 1 object with the statement

```
port port1 (commport, 500, 50,
    19200, P_EVEN);
```

which invokes the constructor (see Listing 2). This initializes the port with a call to the Greenleaf function asiopen, installing transmit and receive interrupt service routines and reserving transmit and receive buffers. The port number is stored `in num` for use by the gets and puts meth-

```
S-ARTboy                                          Wed Mar 13 20:47:51 1991

                              CREATE  SCRIPTS

    at      -- date & time --         if     is    on    turn       on
   after    yy mm dd hh mm ss   if source   goes off   target     off next
01 At         /  /    13:00:C0  TEMP1       Is    On   Big  Fan   On
02 After      /  /    21:30:00  Alarm:      Goes  On   Floodlite  On
03            /  /      :  :    Switch1     Goes  On    Back Lite On
04            /  /      :  :    Switch2     Goes  Off  Back Lite  Off
05            /  /      :  :
06            /  /      :  :
07            /  /      :  :
08            /  /      :  :
09            /  /      :  :
10            /  /      :  :
11            /  /      :  :
12            /  /      :  :
13            /  /      :  :
14            /  /      :  :
15            /  /      :  :
16            /  /      :  :

   Alt-K for Keystroke Help; Alt-F for Field Help; ESC to Exit
```

Figure 3—*The* **create** *script* **screen** *is used to automate the control of output points in the network.*

PLUMBING THE SOFTWARE DEPTHS

Having designed the hardware and firmware needed to perform the low-level S-ARTnet network maintenance, we will now turn our attention to putting these elements to some useful work. One of the simplest control devices that we all have in our homes is the three-way switch. So why not use the S-ARTnet to emulate a three-way switch?

THE S-ARTNET COP

The S-ARTnet rendition of the three-way switch uses the controller as a “traffic cop” that scans the S-ARTnet, maintains satellite status, and dispatches (the possibly updated) control point status to the network satellites. Updating is performed on the order of about six to seven times a second so even if a satellite gets glitched and loses its output state it will be recovered quickly. The S-ARTnet COP application makes use of the support functions described last time, spans about one page of assembly code. and runs from EPROM in a stand-alone fashion.

Thenetworkisdivided intothreegroupsconsistingoftensatelliteseach: 0 through 9.10 through 19, and 20 through 29. Our scheme logically links the groups; these can be thought of as circuits. On power up, the entire network is scanned, the satellites’ sense point status is recorded, and all satellite control points are turned off. Now the controller continually scans the network looking for a change in any sense point. If a change is detected, the corresponding control point is toggled and is written to the corresponding satellite in each group. For example, a change In the sense input 0 of satellite 0 would cause satellite 0 control point 0, satellite 10 control point 0, and satellite 20 control point 0 to toggle. Likewise, a change in satellite 10 sense point 0 would result in the same effect as would a change in satellite 20 sense point 0. The parity checking scheme inhibits any nonexistent or malfunctioning satellite from influencing any of the control points.

The above arrangement can be viewed from several different perspectives. Electrically, the satellites are simply connected to the single wire pair. From an implementation standpoint, there are ten sets of three linked satellites. When considered from an installation point of view, the merit of this approach is most evident. The flexibility of having all communications occur over a single wire pair can be put to use in setting up a **master** control panel that can control and monitor the entire network from a central location. The linked satellites can be equipped with any combination of indicators or load driving devices as well as control switches. Even with our arbitrary design constraints imposed, thisconfiguration permits the control or monitoring of up to twenty unique circuits dispersed over a considerable distance.

The 8031 is well suited to the task at hand. After all, it is at its best when used as a boolean processor, The S-ARTnet is a binary network. Rather than expending the verbiage required to describe the "traffic cop" application, I'll let the code speak for itself. The main body of the program appears thus:

```
; THIS CODE IMPLEMENTS THE S-ART TRAFFIC COP

COP     PROC
L?M0:
        MOV     R0,#SART_BLOCK      ;FOR STATUS CHECKING
        MOV     R6,#SART_BLOCK      ;FOR STATUS UPDATING
        MOV     R7,#0               ;FOR S-ART ACCESS
L?M1:
        MOV     A,R7                ;GET ADDRESS
        CALL    READ_SART           ;GO READ S-ART
        JNB     P,L?M2              ;JUMP IF NO ERROR
        MOV     A,@R0               ;GET PRIOR STATUS
        SETB    ACC.2               ;INDICATE PROBLEM
        MOV     @R0,A               ;UPDATE STATUS BLOCK
        SJMP    L?M5                ;PROBLEM STATUS, LEAVE
L?M2:
        MOV     B,A                 ;SAVE CURRENT STATUS
        MOV     A,@R0               ;GET PRIOR STATUS
        PUSH    ACC                 ;SAVE STATUS
        MOV     C,B.1
        MOV     ACC.1,C             ;UPDATE S1
        MOV     C,B.0
        MOV     ACC.0,C             ;UPDATE S0
        MOV     C,ACC.2             ;RETAIN PROBLEM STATUS
        CLR     ACC.2               ;INDICATE NO PROBLEM
        MOV     @R0,A               ;UPDATE STATUS BLOCK
        POP     ACC                 ;RETRIEVE STATUS
        JC      L?M5                ;LEAVING PROBLEM STATUS
        MOV     C,B.1               ;S1 STATUS
        XORC    ACC.1               ;CHECK FOR CHANGE
        JNC     L?M3                ;JUMP IF NO S1 CHANGE
        CPL     ACC.5               ;TOGGLE C1
L?M3:
        MOV     C,B.0               ;S0 STATUS
        XORC    ACC.0               ;CHECK FOR CHANGE
        JNC     L?M4                ;JUMP IF NO S0 CHANGE
        CPL     ACC.4               ;TOGGLE C0
L?M4:
        MOV     B,A                 ;SAVE NEW STATUS
        MOV     A,R6                ;GET UPDATE POINTER
        CALL    INSERT_BITS         ;INSERT BIT FIELDS
        MOV     A,R6                ;GET UPDATE POINTER
        ADD     A,#10               ;NEXT GROUP
        CALL    INSERT_BITS         ;INSERT BIT FIELDS
        MOV     A,R6                ;GET UPDATE POINTER
        ADD     A,#20               ;NEXT GROUP
        CALL    INSERT_BITS         ;INSERT BIT FIELDS
        MOV     A,B                 ;RETRIEVE NEW STATUS
        MOV     B,R7                ;GET ADDRESS
        SWAP    A                   ;POSITION CONTROL BITS
        CPL     A                   ;SETUP FOR UPDATE
        XCH     A,B                 ;SETUP FOR WRITE TO S-ART
        CALL    WRITE_SART          ;GO UPDATE
L?M5:
        INC     R0                  ;POINT TO NEXT STATUS
        INC     R6
        CJLE    R6,#SART_BLOCK+9, L?M6
        MOV     R6,#SART_BLOCK      ;10 GROUPS!
L?M6:
        INC     R7                  ;POINT TO NEXT S-ART
        CJLE    R7,#29,L?M7         ;EXECUTE INNER LOOP
        JMP     L?M0                ;EXECUTE OUTER LOOP
L?M1:
        JMP     L?M1                ;JUMP HELPER
        ENDPROC

; INSERT CONTROL POINT BITS FIELDS
; INPUT: ACC CONTAINS STATUS BLOCK ADDRESS
;     B CONTAINS BIT FIELDS
```

```
INSERT_BITS  PROC
        MOV     R1,A
        MOV     A,@R1           ;PICK UP STATUS
        MOV     C,B.5
        MOV     ACC.5,C         ;UPDATE Cl
        MOV     C,R.4
        MOV     ACC.4,C         ;UPDATE CO
        MOV     @R1,A           ;UPDATE STATUS BLOCK
        RET
        ENDPROC
        END
```

CLOSING THE LOOP

Just a few words about closing the control loop. Depending on the application, there are different degrees of confidence required In knowing that the control points are in fact in the desired state. When the controller is running the S-ARTnet gateway, the control point state the controller has received is available via the read command. The controller returns both the sense point states and the control point states. Further, if the report indicates that the satellite is in problem status, it can be assumed that the control point Is inoperative. The way the S-ART satellites are configured allows looping the control point back into the sense point, thus the read command will now convey the satellite's actual control point state.

The above technique can be extended using a device that can sense the actual AC or DC load state. This is typically implemented using a relay that closes the sense point input when energized. As you read the S-ARTboy functional description, it will be seen that the scripting capabilities of the S-ARTboy package can be put to good use to take corrective actions if the expected control function fails.

ods. The result of `asiopen` is stored in `status` which is checked by S-ARTboy to make sure the port was initialized successfully. A buffer used to build the string returned from S-ARTnet is created using the C++ new operator.

S-ARTboy continually polls the S-ARTnet controller for the status of all input points by calling `get-points` out of the keyboard loop, as discussed below. If a read from the S-ARTnet controller is not pending, `get_points` calls `port.puts`, passing the S-ARTnet `read_a 11` command. `Port.puts` will initialize the S-ARTnet controller to computer interface mode if necessary and then issue the `read-all` command to the controller. `Port.puts` returns immediately, leaving the actual transmission details to the Greenleaf `ISR`. `Get-points` then calls `port.gets` which continues to take characters from the Greenleaf receive ISR buffer and store them in an intermediate buffer. `Gets` returns immediately if no more characters are available from the ISR, returning 0 until the S-ARTnet trailing envelope character, CR, has been detected. When a CR is detected, **gets** returns 1 indicating to the caller that a S-ARTnet packet of the form

<LF><node **address**> <Out1>
<Out0> <In1> <In0> <CR>

is now available in the caller's buffer. S-ARTnet sends one such packet for each satellite in response to a read all command. When data for the last satellite on the network has been received, gets records that the read from the controller is complete to prepare for another poll.

I found that once the port class had been defined, communication with S-ARTnet was very straightforward. Greenleaf offers a wealth of features and functions, most of which are not needed for the simple communications S-ARTboy requires. Developing the port class hides this complexity from the main body of the application and makes sending **commands** to and receiving data from S-ARTnet simple. For example, **when** the user toggles an output point from the management screen, S-ARTboy builds the S-ARTnet command `cmd_str` to set the selected point to on or off and transmits it to the S-ARTnet controller with the statement "`port1.puts(cmd_str)`". Being able to tailor the port to perform exactly the functions required, and prohibiting any other access, leads to cleaner, more robust code.

```
class port
{
  private:                  //visible only to member functions
    char *p;                //points to buf to build S-net frame
    char *buf;              //buffer used to store S-net frame
    char *bufend;           //points to end of buf
    int num;                //store number of objects comm port
    int status;             //status of comm port
    int leadchar;           //character that starts S-net frame
    int trailchar;          //character that terminates frame
    int net_initialized;    //record if port initialized

  public:                   //visible from outside object
    port(int pnum, unsigned int rx_len, unsigned int tx_len, \
         int baud, int parity);
    int gets(char *dest, unsigned int len);
    unsigned int puts(char *source);
    int get_status(void);
    ~port();
};
```

Listing 1—*The port class handles the serial port and communication with S-ARTnet.*

COORDINATING S-ARTBOY'S TASKS

S-ARTboy has several tasks: accept and process user keystrokes, check input points for state changes, toggle output points, update the management and history screens in real time, and execute scripts when conditions become true. The approach adopted for S-ARTboy was to use idle time spent waiting for a keystroke to undertake other tasks. After a screen

```
//Implementation of member functions of class port

#define  NOTERM -1
#define BUF_LEN 25
#define INITSTR "\012P0\015"

//Constructor; init port, allocate buf
port::port(int pnum,  unsigned int rx_len, unsigned int tx_len, \
              int  baud,  int parity)
{
   num = pnum;
   leadchar = LF;
   trailchar = CR;
   net_initialized = FALSE;

   buf = new char[BUF_LEN];
   bufend = buf + BUF_LEN;
   p = buf;

   status = asiopen(pnum, ASINOUT | BINARY | NORMALRX, rx_len, \
              tx_len, baud, parity, 1, 7, 1, 1);


//Destructor;  release memory, release port
port::~port()

   asiquit(num);
   delete buf;


//Member function to get a string from S-ARTnet control_er
int port::gets(char *dest, unsigned int len)

   while (1)

      *p = (char) asigetc(num);

      if (*p < 0)                //if no more characters in ISR
      return(0);                 //receive buffer, return 0

      if (*p == leadchar)        //skip start of frame char
         continue;

      if (*p == trailchar)       //we have a S-net frame
      {
         *p = (char)NULL;
         if  (strlen(buf) > len)
            return  0;
         strcpy(dest, buf);      //copy to callers buffer
         p = buf;                //ready to build next frame
         return(i);

      p++;
   }


// Member function to send a command to S-ARTnet controller
unsigned int port::puts(char *source)

   if (!net initialized)
   {
      asiputs(num, INITSTR, NOTERM);
      net_initialized = TRUE;

   status = asiputs(num, source, NOTERM); //send command to ISR

   return(status);
}

//Member function to check status of comm port
int  port::get_status(void)

   return status;
```

Listing **2**—*The* implementation *portion of the port class*

such as the management screen is displayed, t **as-get-key** iscallcd to fetch a keystroke (Listing 3). Tas_get_key loops while waiting for a keystroke, calling routines get points, check scripts, and update screen.Get points updatesinput point information as described above. Check_ scriptsrunsthroughthelist of user-created scripts to see if any should be executed, and if so sends the appropriate commands to the S-

```
int tas_get_key()
{
   while (1)
   {
      get_points();
      check_scripts();
      update_screen();

      if (keyready())
         return(kb_get());
   }
}
```

listing 3—*After a screen is displayed, this function is called to fetch a ketstroke.*

ARTnet controller. Update-screen updates the changed fields on the current screen.

Coupled with theinterrupt-driven communications, thisapproach works well. Even with continuous activity on the network at 19,200 bps, there is no delay in response to user keystrokes. Scripts execute with no apparent delay. For example, if input 00-2 is connected via a script to output 06-0, flipping a switch connected to the input off results in the light connected to the output immediately going off.

S-ARTBOY GROWS UP

We named this first version of the S-ARTnet management software S-ARTboy because we expect it to grow and mature with time. Planned enhancements include support for several S-ARTnets so the control network could be expanded greatly. Also to be added is the capability to allow a user to dial in via modem to run S-ARTman from a remote PC and to dump history files to that PC. The script language will be expanded to support additional features such as dialing out to activate an alarm system. And of course there will be a Windows version. We envision S-ARTman to be a flexible tool, easily modified to meet the needs of its users.+

SOURCES

Online Devices
P.O. Box 218
Stafford, CT 06075-0218

S-ARTnet Network Adapter
S-ART Satellite Node
Programmed S-ARTmaster EPROM

Write for specifications, pricing, and availability.

Trans Atlantic Software
2 Cole Rd.
Haydenville, MA 0 1039
(413) 268-3077

S-ARTman-PC-based S-ARTnet management program
ScreenMan-The screen management package used to create S-ARTboy and S-ARTman

Cottage Resources Corp.
1405 Stevenson Dr., Ste. 3-672
Springfield, IL 62703
(217) 529-7679

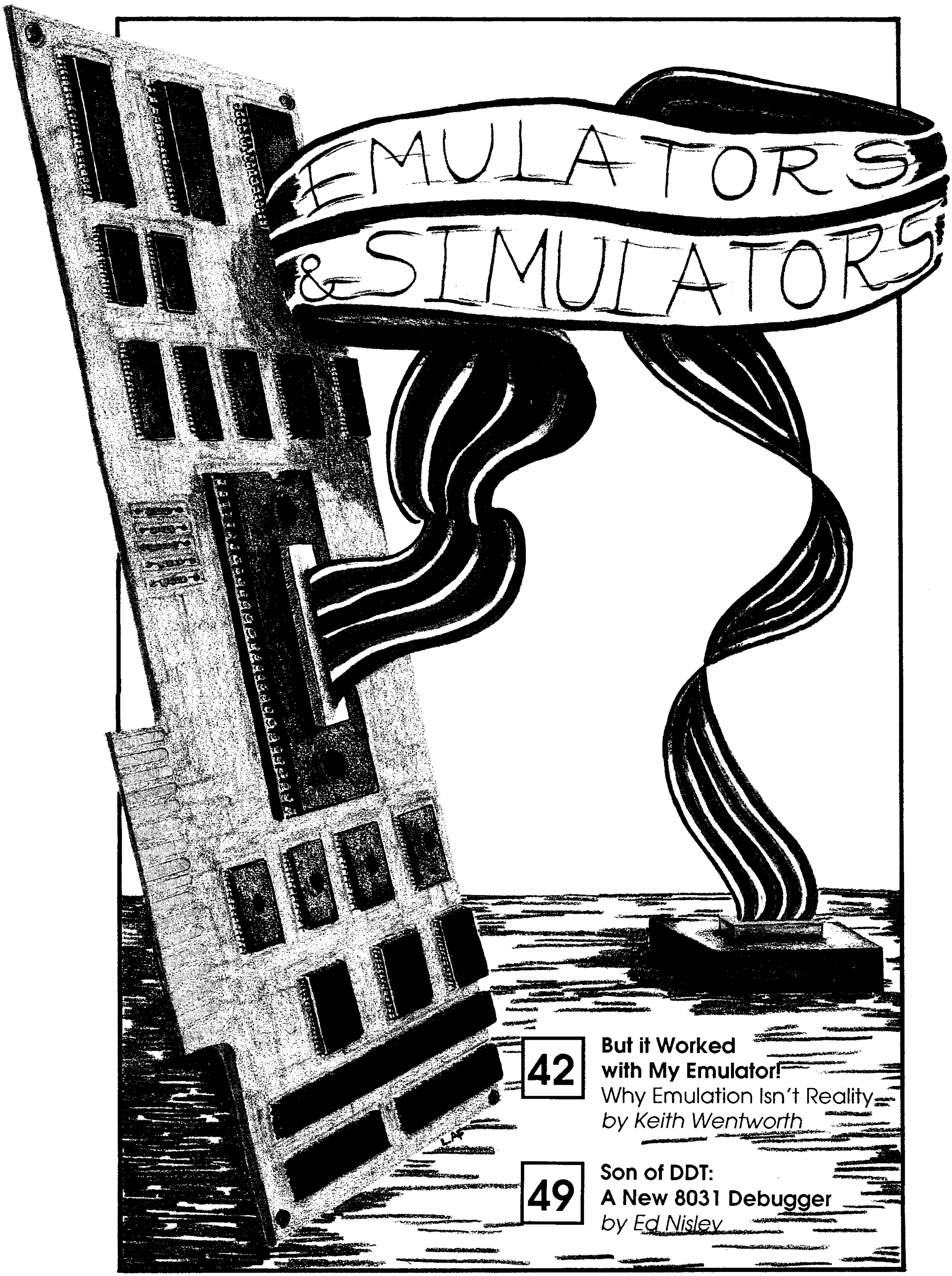
Control-R I 803 1 single-board computer

Dan Burke has an M.S. in computer science from the University of Massachusetts and 10 years of experience in software development. He is the owner of Trans Atlantic Software, specializing in network control software and systems for the video conferencing industry.

John Dybowski has been involved in thedesign and manufactureof hardwareand software for industrialdatacollectionandcommunications equipment.

IRS

407 Very Useful
408 Moderately Useful
409 Not Useful



42 **But it Worked with My Emulator!**
Why Emulation Isn't Reality
by Keith Wentworth

49 **Son of DDT: A New 8031 Debugger**
by Ed Nisley

SPECIAL SECTION

But it Worked with My Emulator!

Keith Wentworth

Why Emulation Isn't Reality

When designing a new product, the choice and use of new tools are often crucial to meeting deadlines. An appropriate tool used in the proper way can save many hours of frustration. On the other hand, a tool used incorrectly can waste your valuable time.

An In-Circuit Emulator, ICE for short, is a powerful debugging tool used by programmers. The ICE is a device that can take the place of some integrated circuit chip. It imitates that chip's software and electrical functions. There are many types of in-circuit emulators, but our focus will be on microcomputer ICEs. This article will specifically cover many common "gotchas" that emulators cause, so these pitfalls can be avoided. Most of them will seem obvious, but in the rush to complete a project, we sometimes don't notice an obvious problem until it has wasted much of our time.

Why would someone use an expensive piece of hardware for debugging purposes when inexpensive software simulators exist? There are two main reasons. First, as its name implies, an ICE can be connected to a real circuit. This allows real signal inputs and outputs to be used during testing. Second, it executes instructions instead of simulating the execution of instructions. Simulation is usually much slower.

An ICE can also be used in a production environment to automate testing of target system hardware (your product's PC board). Several different test programs may be executed to do functional testing of the various circuits on the target system. They could help diagnose such hardware problems as shorted lines, bad memory, and so on.

There you are with a deadline looming above your head. Your project has been debugged using an in-circuit emulator. The final test is to bum a chip and watch it work. Instead, a loud cry can be heard throughout the building, "But it worked with my emulator!"

Or-you may be revising an existing product that works fine. All you want to do is add some features. You buy an ICE to help. After downloading the program into the ICE, you plug it into your product. Nothing happens. A perfectly good program no longer works.

How could this happen? Isn't an in-circuit emulator supposed to behave exactly like the chip it is emulating? No. In order for an in-circuit emulator to provide the user with all these great debugging features, it must

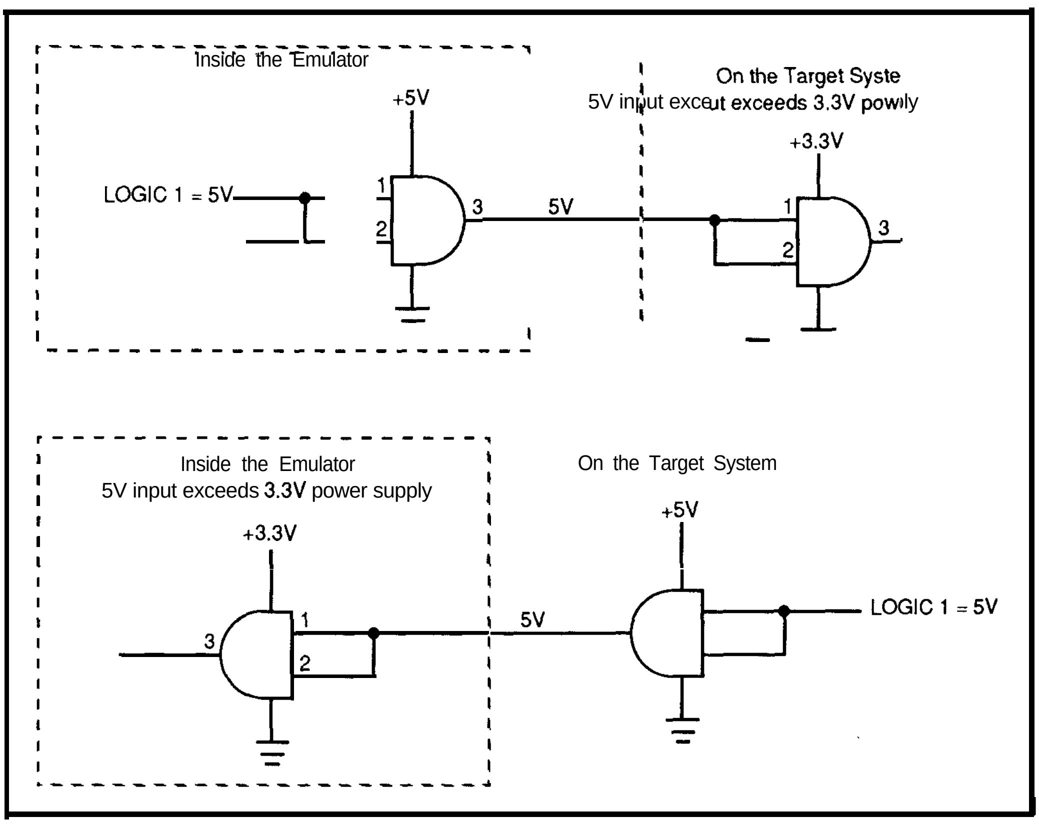


Figure 1 —*In* CMOS devices, *power supply differences are a real problem. It* would be *disastrous to attach an ICE with a 5-V power supply to a target system with a 3.3-V supply. or vice versa.*

be different from the chip. Usually these differences are minor and easy to live with if you know and keep within the limitations they impose.

You have decided to use an ICE as a debugging tool for a project. This decision requires that you design your target system and write your programs to function correctly when using the ICE and also when the real chip is used. This sounds easy but provisions for physical, electrical, and operational differences must be made.

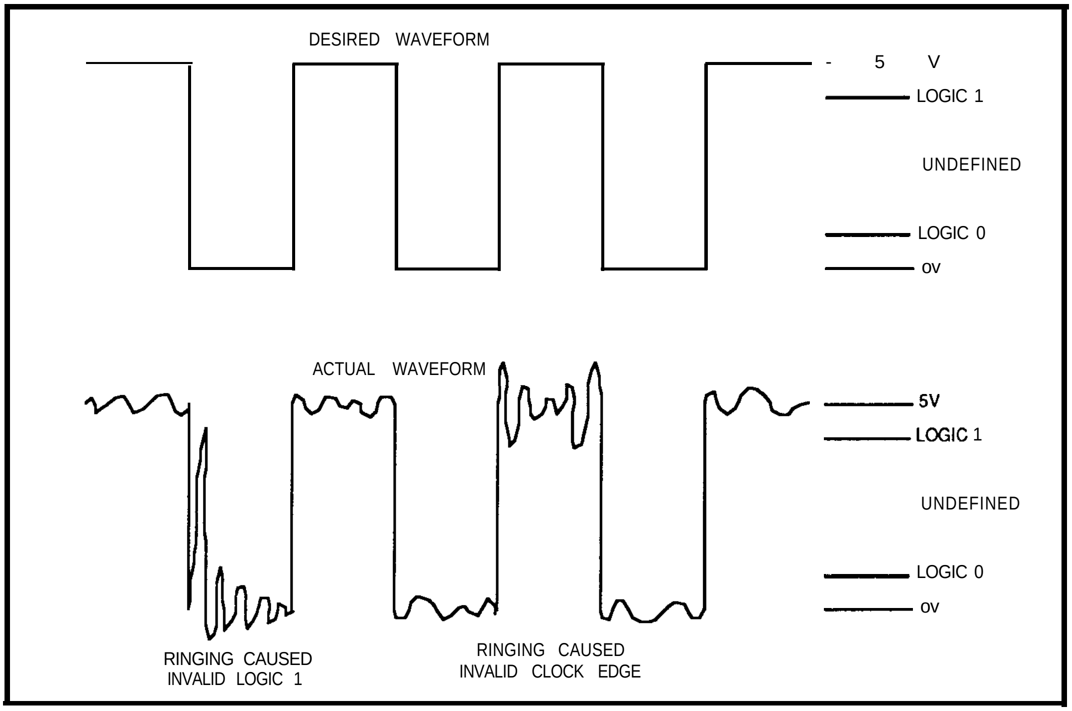


Figure *Z-Ringing in ribbon cables can sometimes cause false logic /eve/s to be detected.*

PHYSICAL

The emulator needs to have some way to plug itself into the target system so it can be "in circuit." This is often done with some sort of plug on the end of a ribbon cable. The plug goes into the socket where the real chip would normally go and the other end of the cable connects to the emulator. Sometimes there is a small PC board (often with IC chips of its own) between the cable and the plug. The target system design needs to provide enough room for whatever connection is required by this plug and PC board.

ELECTRICAL

As with all electronic equipment, an emulator will need to have power supplied to it. The emulator can have its own power supply or receive its power from the target system.

An emulator typically consumes more power than the chip it is emulating because of the extra circuitry required. Obviously, if the **emulator** uses the target system's power supply, that power supply must be able to take on this extra load, and the emulator must always be connected to the target system to work. Excuse the statement of the obvious, but do you think that ever gets forgotten in the heat of the deadline frenzy? You bet it does.

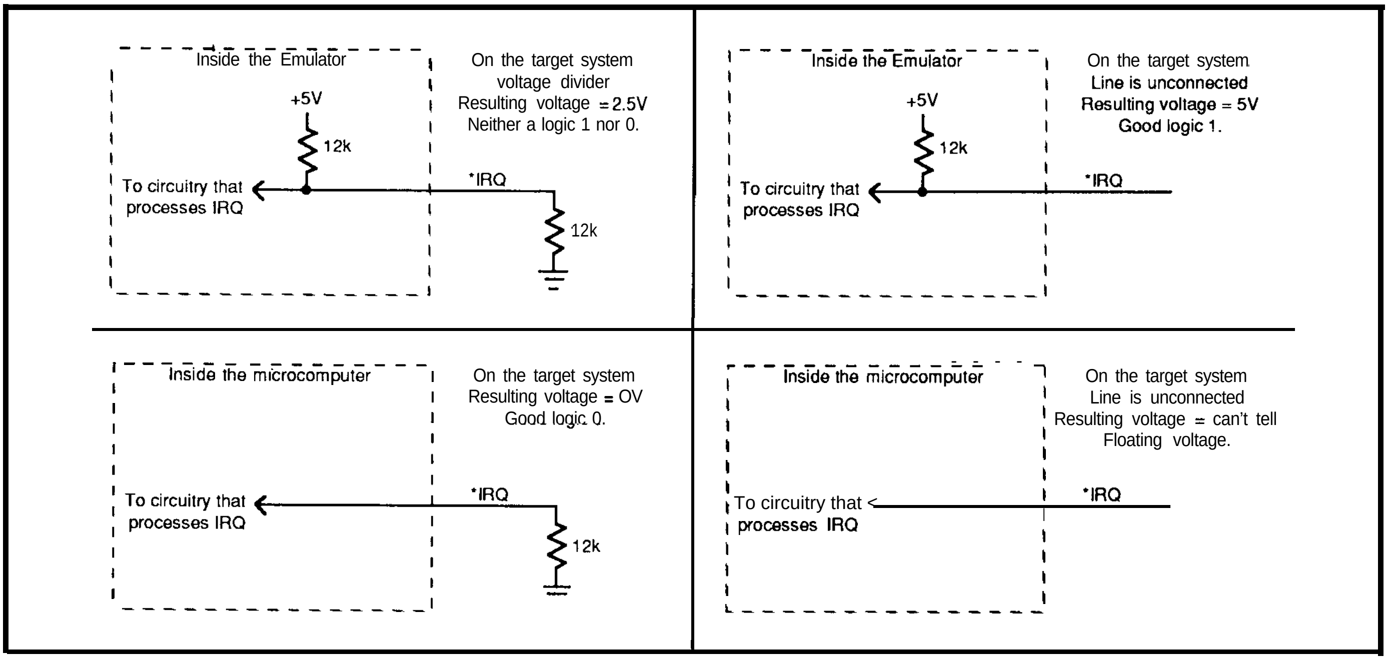


Figure 3—*Pull-up* **resistors** *internal to the emulator can cause unexpected results when a real chip is plugged into the circuit.*

If the emulator supplies its own power, care must be taken to ensure that it and the target system use the same power voltage. For starters, different voltages can cause the two devices to disagree on logic levels. So one device could output a one and the other may consider it a zero.

In CMOS devices, power supply differences are a real problem. CMOS devices can typically operate from 3.3 V to above 6 V. It would be disastrous to attach an ICE with a 5-V power supply to a target system with a 3.3-V supply, or vice versa. What would happenistheonewiththelowerpower supply voltage would have signalson its inputs that are greater than its supply as shown in Figure 1. This would cause the device to go into a CMOS "latchup" condition—essentially making the impedance from power to ground nearly zero. This can cause various unpredictable results--all of them bad, some disastrous. CMOS latchup could also occur if the two power supplies are powered up or down at different times.

SEND A CABLE

Ribbon cable, often the connection between emulator and target system, causes considerable changes in theemulator'selectricalbehavior.The length of the cablecausesa delay in all the signals traveling over it. Remember a typical signal delay is about 2 ns per foot. Make sure that the timing requirements of your target system can accept this extra delay. This is not muchofaproblemexceptforthefaster microprocessors.

Ribbon cable has a different intrinsic impedance from the traces on a PC board, which is often a considerable impedance discontinuity. This will cause all higher frequency signals to have some ringing such as that shown in Figure 2. If the ringing on a signal becomes too great, it can go over logic level thresholds. If the signal in question isoneprovidinga clock for a flip-flop, the ringing may be mistaken for a valid clock pulse.

In the event that ringing is a problem, there are four simple solutions: if you can,emulate at a lower frequency; shorten the cable; add some termination resistors to the end of the cable; or route the offending line through a Schmitt trigger buffer.

The cable provides some crosstalk between its wires. Signals from one wire can induce noise into the wires next to it. The above solutions can also help with crosstalk.

Usually, theribboncableprovides the only common reference voltage (ground) between the emulator and the target system. When large current spikes travel along this wire, it causes a ground shift, which can cause the same sort of problems previously described for different logic levels. If ground shift causes a problem, just add another ground connection between the emulator and the target.

Emulators for faster processors use more elaboratecables to minimize the above problems. Some cables have every other wire serve as a ground wire to minimize crosstalk and to control the impedance.

ON TARGET

A target system that uses a crystal as the clock for the processor may not oscillate when using an emulator. Again the ribbon cable is part of the problem. Its long length and other properties can make the crystal oscillate unreliably, if at all. Some emulators have their own internal clock that may be used. The clock in the emulator must match the frequency of the clock source designed into the target system.

An emulator that can function without a target system will have a method-usually a resistor-to provide a stable voltage for such critical inputs as an IRQ (Interrupt Request) line. When such an emulator is connected to a target system and that line is not connected, the system will still function properly. However, when the chip is used instead, the line is able to float and cause sporadic interrupts. If you pointed all your vectors to valid interrupt handlers, the only problem you will have is a good deal of wasted time servicing the bogus interrupts. If you have any unassigned vectors, the first unexpected interrupt will be vec-

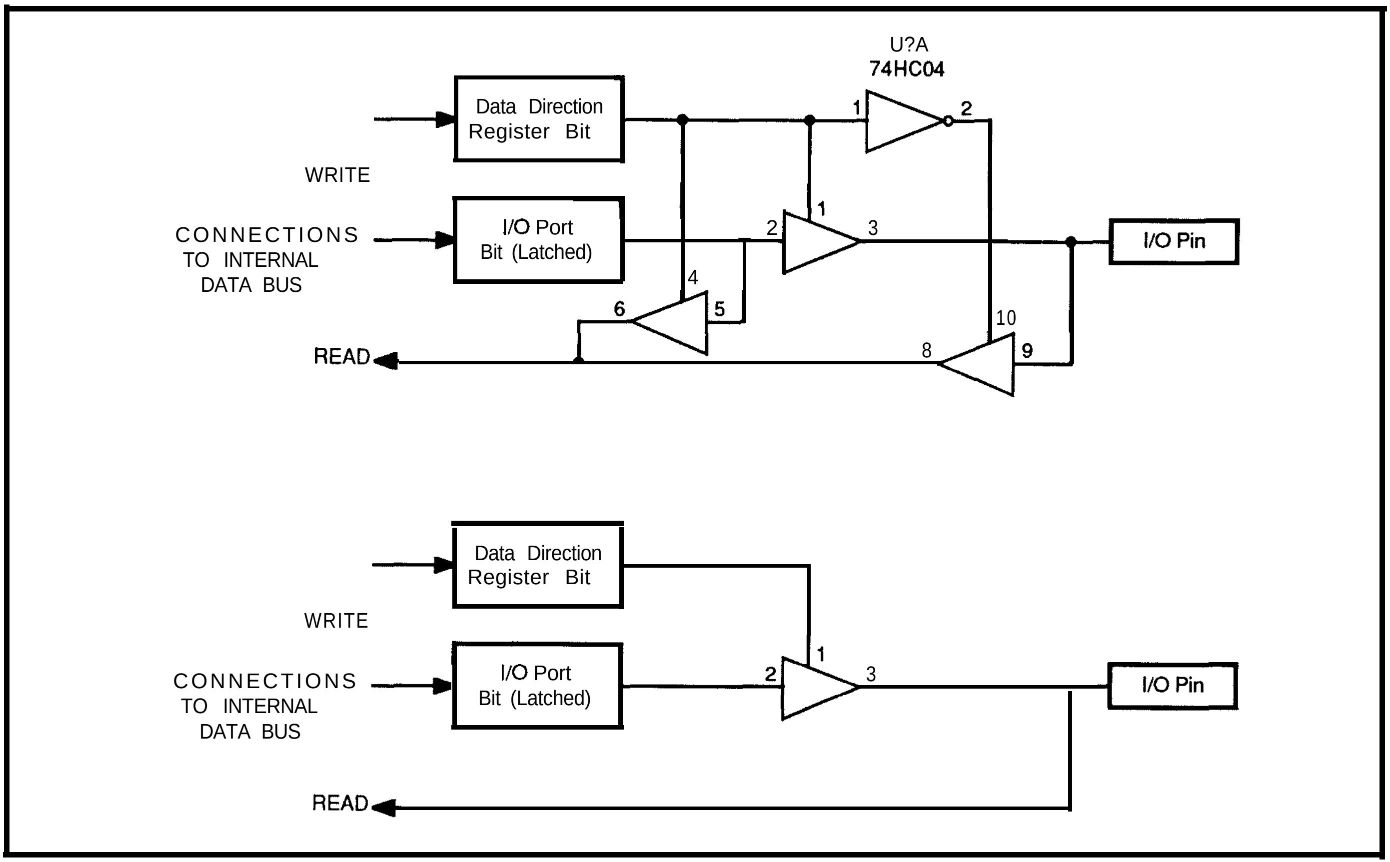


Figure 4—*The use of an emulator can also mask potential signal loading problems.*

tored to who knows where and probably result in the program crashing.

Another problem could arise if the emulator has a pull-up resistor and the target system pulls a line low through a resistor as seen in Figure 3. When the emulator is connected, the resistors provide a voltage divider so the line may not be read as low. When the chip is used, the pull-down resistor can then pull the line down to zero.

The input/output ports of an emulator may have slightly different input levels and output drive capacities than the real chip. Some microcontroller chips accept as valid a wider range of input levels than do the standard CMOS gates which emulators commonly use for their ports. Any signals from the target system that have near marginal levels could become marginal when the emulator is in use. An emulator may be able to source and sink more current then the real chip. Therefore, the emulator may be able to drive many TTL loads directly where the chip **may** need a buffer to get the same results. In a similar manner, an output port may, when read, give the results of its latch, where the emulator reads the actual output pin value. Therefore, the former will show the correct logic value even if the port is heavily loaded, but the latter will depend on the pin load. See Figure 4 for more details.

OPERATIONAL

Many emulators are designed to function for a whole family of similar chips. These chips may have some significant differences between them. For example, the memory maps may be different between chips. Emulators often use RAM as pseudo-ROM. This RAM can have a program loaded into it, then behave like ROM when the program is running. Since the emulator may work for different chips, it may have more RAM or (pseudo) ROM than the real chip. If your program uses this invalid memory, it will not function correctly with the real chip.

In order for an emulator to be useful as a debugging tool, it needs to have a method of running and stopping a user's program and showing the user the current state of everything. To do this an emulator has a monitor program and some hardware to stop running the user's program and start running the monitor program. When a user's program is stopped, it is called a breakpoint.

As you may have guessed, the monitor program and breakpoints make the emulator different than the real chip in ways we must take into consideration. The monitor program must be located in the memory map. This can be done two ways. One way is to reserve some of the emulator's memory map for use of the monitor program. Now you must write programs that won't use the memory needed by the monitor program. This method is only used in very simple, low-cost emulators. The other way is to have two **memory maps,** one for the user and one for the monitor, and switch between them. This method allows the monitor program to function "invisibly." It doesn't need any of the user's memory map. This second method is used in most emulators.

One method used to implement breakpoints in testing is to cause a special interrupt to occur. A commonly used interrupt for this purpose is the Software Interrupt (e.g., SWI). Of course, if you use the software interrupt for testing, this makes that instruction unavailable for the user program. Thus, if your program uses the software interrupt, the result will be a breakpoint instead of the intended effect.

When a breakpoint occurs, its associated interrupt may leave informationon the user stack. Theonly significant problem this poses is that there must be room on the stack for this additional information. If no room is provided, this information will write over the previous important stack information.

Breakpoints are useful and necessary, but regardless of their implementation, they bring with them a few problems. All these problems occur because the real world doesn't stop when the emulator stops the user's program at a breakpoint.

While the emulator is running the monitor program, there can still be interruptsgenerated by the target system. These interrupts may be ignored and still be pending when the user's program is continued, or they may be serviced by themonitor program with a dummy service routine. On 68xx families, the built-in, free-running timer cannot be stopped, so it keeps ticking while the monitor program is running. What often occurs after a breakpoint is the timer immediately timesout and other interrupts want to be serviced.

If you are emulating on a real target system, it is possible to have conditions where a program must not be stopped. This could include the situation where one signal must follow another after a short period of time. If the program is stopped after the first signal but before the second, real damage could occur on a target system. Determining and avoiding such dangerous breakpoints is the responsibility of the user.

What happensif you want to emulate a low clock frequency? This is a real issue with CMOS chips since they can operate down to a DC clock. The emulator will work just fine while it is running the user's program, but it is running too slow for the monitor program to communicate quickly with the user. This can make a simple debugging session drag on and on. Some emulatorsget around this problem by switching to a fast clock while running the monitor program regardless of the speed of the clock used for the user program.

FINAL NOTES

This article has covered many of the special considerations involved when using in-circuit emulators. Before buying an ICE, be sure to find out about its limitations. Armed with this knowledge, you can choose an emulator that will work for your system, or design your system to work with that emulator. ✣

Keith Wentworth is a project engineer with The Engineers Collaborative Inc. (TECI). He helped design and provides customer support for the company's MC68HC05 and MC68HC11 in-circuit emulators.

IRS

410 Very Useful
411 Moderately Useful
412 Not Useful

Son Of DDT: A New 8031 Debugger

FEATURE ARTICLE

Ed Nisley

Back in the early days, computers were electromechanical, programs were tight, and debugging used harsh chemicals. One such program produceddifferentresultsoneachrun; something was wrong.

The engineers-in-attendance found, after a diligent search, a moth trapped in a relay's contacts. It was extricated and taped into the logbook with the notation that the system had been "debugged."

Thus is language created.. .

While you will never find a moth between thecontactsofyour8031 CPU, your programs still need some debugging. But with no relays or vacuum tube filaments to examine, the task has become far more difficult. Fortunately, complex computers now help get bugs out of complex programs.

EMBEDDED DEBUGGING

The DDT-51 project (described in volume 7 of the "Ciarcia's Circuit Cellar" books) struck a responsive chord. Many programmers (and ex-nonprogrammers!) evidently needed a low-cost 8031 hardware debugger. Although the DDT-51 project was not a full ICE (In-Circuit Emulator), the price wasn't that chilling, either!

The heart of DDT51 was a board holding a remarkably small amount of logic and a 2K—byte static RAM chip. The RAM was accessible from both an IBM PC and the 8031 target system; the debugging kernel routines copied 8031 CPU registers into the Debug RAM where the PC could read and display them. The kernel could also copy changed registers back into theCPU. The kernel was small enough to run entirely from Debug RAM, so only a very few vital instructions intruded into the normal program space.

In order to keep the board's cost down, the PC controlled the DDT-51 hardwarethroughabidirectionalparallel port. A standard PC parallel printer port card would work after a one-wire change, but more recent LSI and ASIC printer cards could not be modified. Most laptops have a bidirectional printer port built in, but each activates input mode in a different way. Worse than that, some standard PC I/O buses wereincompatible with a modified printer port card that worked correctly on other standard systems. So much for standards.

While the basic concept was sound, an RS-232 serial interface would be compatible with more PC systems. An improvement in the "user interface" part of the program running on the PC would be handy, too. And, while we're at it, why not add a few features, like symbolic debugging, support for CPU bits, and.. .

The result was the Son of DDT, an 8031 debugger for the '90s!

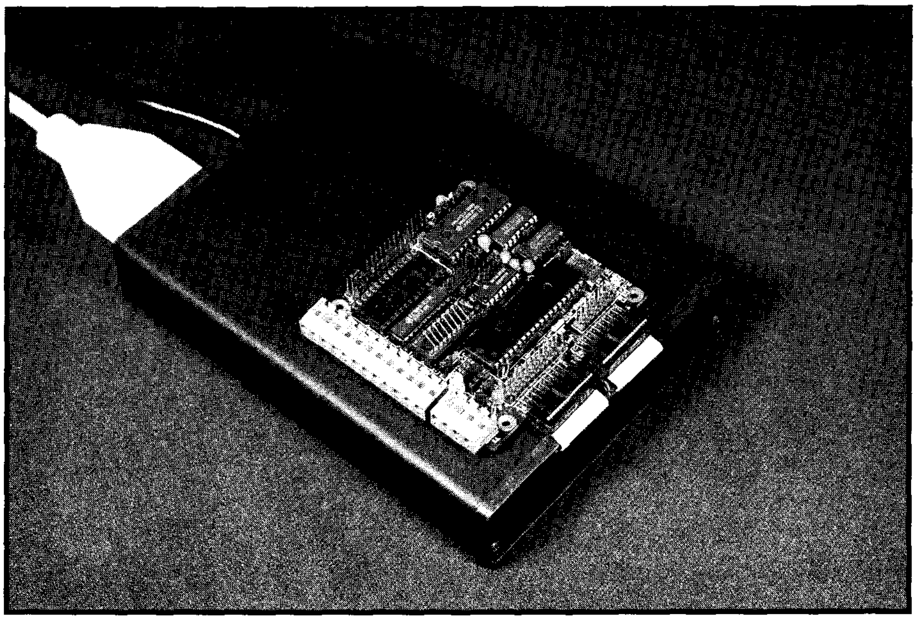


Photo 1 —*The finalpackagedSon of DDT system has the targetprocessormountedon top.*

This article is a guided tour into Son of DDT's structure rather than a user's guide. When you know how Son of DDT works its magic, you can use it more effectively in your 8031 systems. And, of course, if you find yourself writing a debugger for your own systems, these tips and traps can save you a lot of trouble.

SOMETHINGOLD, SOMETHING NEW

Figure 1 details the interconnections for a complete Son of DDT system. The DDT51 Controller board is an RTC31 board with 8K RAM and 32K EPROM which, among other functions, converts the DDT-51's parallel interface into a standard RS-232 link at 9600 bps. *[Editor's Note: See "Creating a Network-based Embedded Controller" in issue #8 of CIRCUIT CELLAR INK for design information about the RTC31.]* The original DDT-51 board serves as the Target System hard ware interface. The Target System is an RTC31 system with 8K of static RAM and no EPROM, as the program is downloaded from the PC through the Controller's serial port.

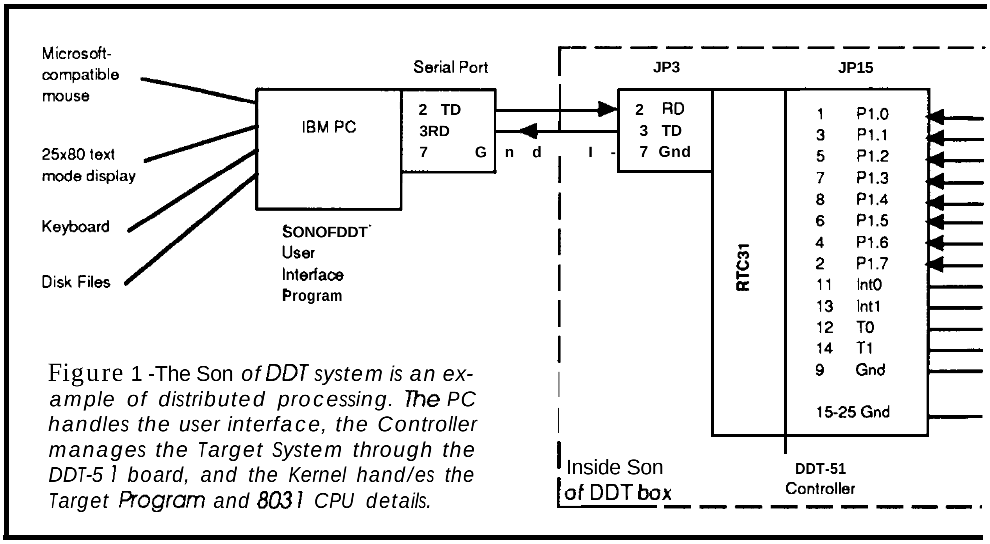


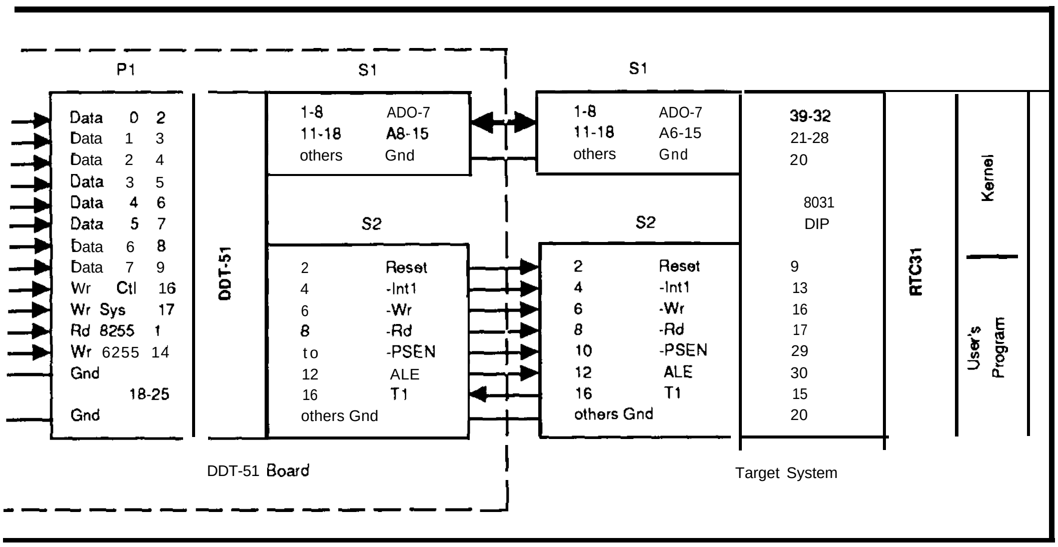
Figure 1 -The Son *of DDT system is an example of distributed processing. The PC handles the user interface, the Controller manages the Target System through the DDT-5 1 board, and the Kernel hand/es the Target Program and 8031 CPU details.*

To show how far things have come, one "component" in the schematic is a whole RTC31 board. It has two input lines (the serial port) and twelve output lines (the parallel port). The fact that an RTC31 is a complete computer doesn't show up at this level of detail. Now that's integration!

Photos 1 and 2 show the Son of DDT Development System, inside and out. The packaging takes advantage of the RTC31 expansion connectors; you can build a complex project atop the Development System case by stacking I/O boards atop the RTC31 CPU board. The PC's serial cable plugs directly into the back-panel DB-25 connector, requiring only three wires.

By combining the Controller, DDT-51 board, and Target System into a single package, the Son of DDT system eliminates the hardware compatibility problems some users had with the original system. All of the critical cables are preassembled and that parallel port is now entirely internal.

As in the original DDT51 system, an IBM PC program fetches data from the Target System and formats it for the PC's screen. As you can see from Photo 3, though, there have been some changes! Son of DDT displays the 8031 CPU status in a full-screen format,



disassembles the Target System program using symbolic names, and dumps both Internal and External Target RAM in a pair of windows. Along the top of the screen is a pull-down menu bar giving access to all of the program's features.

One particularly endearing feature is that CPU bits are individually displayed, so decoding hex registers is a thing of the past. For example, the myriad serial port status bits are collected in the upper-right corner of the display. To change a bit, move the cursor to that field (a mouse click will do) and press the spacebar to toggle it. Son of DDT merges the bit into the proper CPU register automatically.

KERNEL COMMUNICATIONS

Because Son of DDT's hardware is so straightforward, the key to its capabilities must lie in the debugging kernel. This code is responsible for starting, stopping, and stepping the user's Target Program, copying internal CPU and RAM data to and from the Debug RAM, and handling communications with the DDT-51 Controller. All this function is packed into 850 bytes of 8031 code; obviously not a job for a high-level language!

The Debug RAM holds nearly all the Kernel code, but several essential jumps and loops must be located in the Target System RAM. For example, the Reset and INT1 interrupt vectors (at 0000 and 0013, respectively) must branch into Kernel code.

This introduces an obvious chicken-and-egg situation: how does the Kernel code get into the Target System and Debug RAM in the first place, without using any Target System code? The answer to that illustrates how the whole Son of DDT system works, so a digression is in order before I describe the Kernel functions.

The 8031 is a microcontroller, not a "real" computer, and is therefore missing several key features we take for granted in other systems. Conspicuous by their absence are pins to control memory wait states or bus access by other CPUs or peripherals. An 8031 program is the only way to read or set the on-chip registers and RAM, because external hardware cannot control the bus or affect the CPU!

Incidentally, ICE hardware solves this problem by replacing the **8031 CPU** with discrete logic that provides access to those internal registers. Old ICE systems used tons of MSI TTL (and fans!), but current systems use LSI to reduce cost and size. When an ICE is stopped at a breakpoint, the internal state is available without executing any 8031 instructions because the wires (or at least logical connections) are there to extract the information.

Figure 2 shows the Target System memory map. The Debug RAM on the DDT51 board has data, address, and control connections to both the Target System and the DDT-51 Controller, so that either side can access it independently. That logic (discussed in the original DDT51 articles) was simplified by a software agreement that only one "side" of the system would attempt to access the Debug RAM; the logic need not worry about arbitrating simultaneous access. The remaining addresses are part of the Target System, so they are normally accessible only from the 8031 CPU.

Son of DDT takes advantage of one special case: when the CPU reset

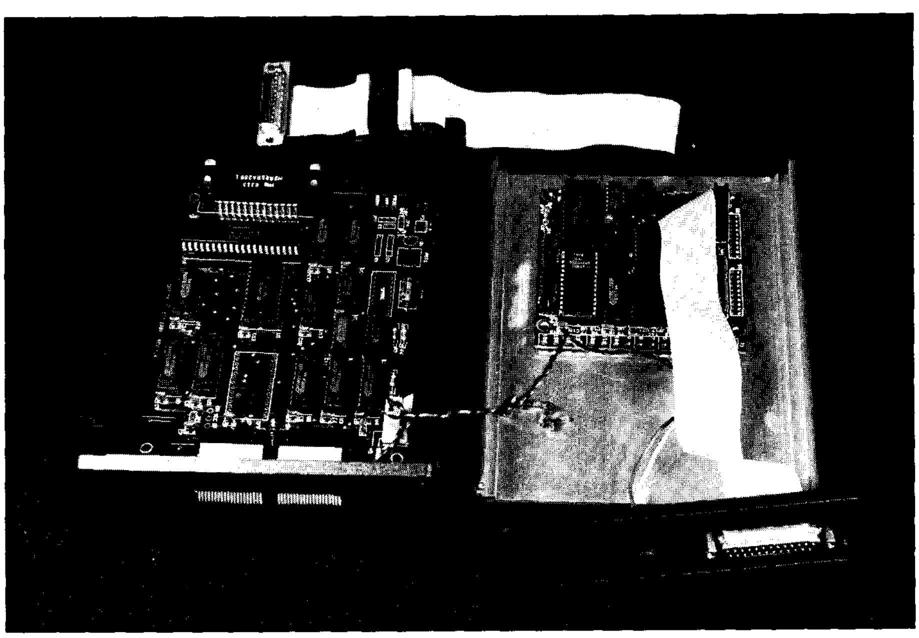


Photo P—*Inside, the RTC31 on the right handles* **all host communications for the** *DDT-51* **on the left.**

line is active, the 8031 puts all its bus outputs and controls in a high-impedance state. Thisgivcs the DDT51 Controller full Target System access without conflicts. In effect, the Controller wiggles the WR, RD, and PSEN lines just as the 8031 CPU would, and the peripheral devices cannot tell the difference (apart from the speed!).

Of course, resetting the CPU halts the 8031 program, so it is not a step to be taken while debugging a program. Although the original IBM AT used a dastardly trick involving hardware resets and magic bytes in the keyboard controller to switch the 80286 CPU from protected to real mode, I didn't think this was a trick worth emulating in Son of DDT.

To load the Kernel code, the DDT-51 Controller activates the Target System Reset line and transfers the Kernel code directly into the Debug and Target RAM chips. When the Controller releases Reset, the CPU vectors through the branch at address 0000 in Target RAM to the Kernel code in Debug RAM. After that, Kernel must follow the software **agreement** to avoid collisions with the DDT51 controller.

The 8031 can address 64K bytes of program memory and 64K bytes of data memory, but the Son of DDT system combines these into a single 64K program / data memory. The reason should bc obvious from the preceding discussion: the DDT51 Controller and Kernel code must bcable to read and write the user's Target Programinstructionsas"data"ratherthan as"program" values. Because the 8031 has no way to write program instructions (they are usually stored in EPROM!), the only practical method is to treat them as data.

DELICATE HANDSHAKES

Any time you plan communication between two computers (or two people!), you must first determine a "protocol" for the conversations. The DDT51 Controller and the Kernel code use just two bits to negotiate a conversation through the Debug RAM, ensuring that only one side accesses the RAM at a time.

The DDT-51 Controller can cause (or at least request) an 8031 interrupt by activating the INT1 line; the bit is called IRQBIT by the Kernel code. When the Kernel responds to the interrupt,itactivatestheTOoutput(a.k.a. HALTBIT) to indicate that the Target Program is no longer running. That exchange marks the start of a conversation between the two.

Figure 3a shows what happens **when** IRQBIT **goes** active when the Target Program is running. The Kernel interrupt handler copies the 8031 CPU's internal state into Debug RAM, updates some status variables, and asserts HALTBIT. The Controller waits for HALTBIT to become active, then extracts the values from Debug RAM.

Just as the Controller cannot access Debug RAM data while the Kernel is active, Kernel code must stay out of the Controller's way. Because the hardware overlaps Program and External Data spaces, the Kernel cannot even execute instructions from Debug RAM! The need for the "Reserved for Kernel" block in Figure 2 should now be apparent: the Kernel code spins in a tight loop while waiting for the Controller to finish accessing the Debug RAM.

The whole point of stopping the Target Program is to get the CPU's state to the PC via the DDT-51 Controller. The user (you!) can then update the values shown on the PC's screen and restart the Target Program. The SONOFDDT program sends the new values to the Controller, which writes them into Debug RAM and sets a control value indicating that Kernel should resume running the Target Program.

Figure 3b shows this program start sequence. When IRQBIT **goes** inactive Kernel emerges from the Target RAM spin loop, restores the CPU state from the Debug RAM, clears HALTBIT, and passes control to the Target Program by executing a normal "Return From Interrupt" instruction.

As described above, the DDT51 Controller cannot read or write the Target RAM (addresses 0000–7FFF), nor I/O devices (addresses E000–FFFF), because the 8031 controls the bus lines. Therefore the Kernel must copy data from the those sections of the 8031's address space into Debug RAM and copy changed values back,

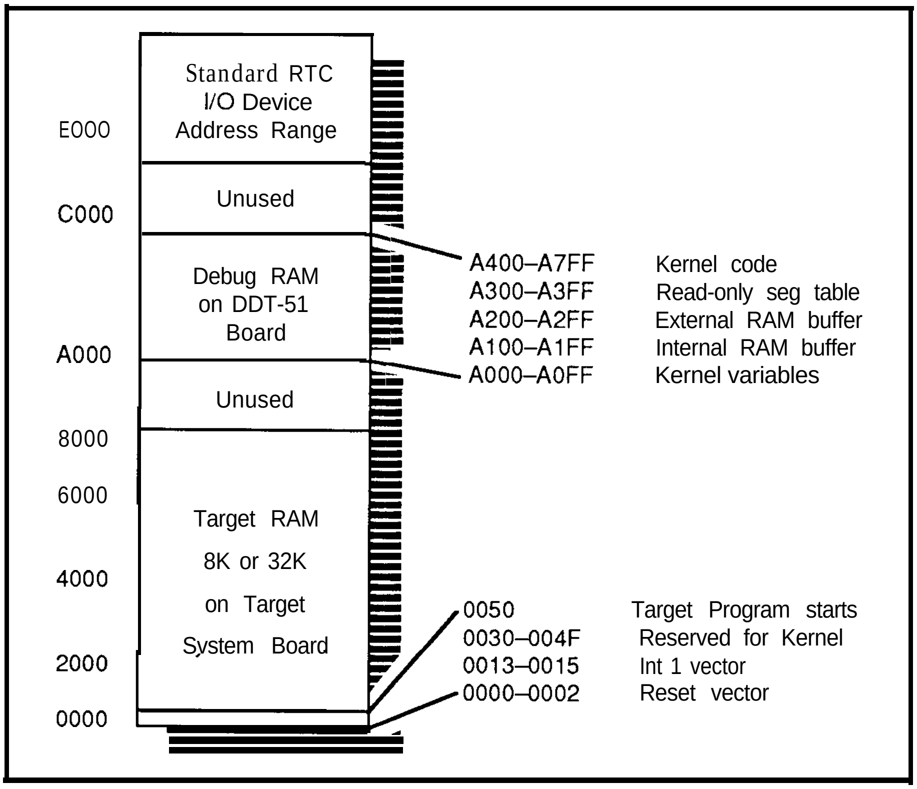


Figure 2—*The Target System CPU see the Target RAM and Debug RAM as a normal part of its addressspace. However, logic on the DDT-5 1 board allows the DDT-5 1 Controller to read and write the Debug RAM.*

much as it does for the internal CPU state. A shortcut protocol avoids the overhead of saving and restoring the Target Program state, which can't change during this process.

Figure 3c shows what happens during such a Kernel function. The Controller sets up the Debug RAM and passes control to Kernel as before, but Kernel clears HALTBIT and executes the function, which may involve copying up to 256 bytes of data in either direction. Meanwhile, the Controller polls HALTBIT, which Kernel will assert when the function is complete. Once again, each side can access the Debug RAM only during the specified parts of the exchange.

A special case arises when running the Target Program to a breakpoint, because several seconds (or hours!) may elapse before it halts; if it never hits any of the breakpoints, the Target Program will never stop. Figure 3d shows this situation. In essence, the Controllermust know when breakpoints are active (it should, because it tells Kernel to set them!) and poll HALTBIT until Kernel sets it after a breakpoint. For obvious reasons, there can't be a timeout on this delay.

To regain control before hitting a breakpoint, the Controller executes the same Halt handshake shown in Figure 3a. Kernel removes all the breakpoints and responds normally.

Starting Kernel after a program load also involves a special handshake, as shown in Figure 3e, because the 8031 hardware comes out of reset with HALTBIT active. The DDT51 controller releases the reset line and waits for HALTBIT to go low, which indicates that Kernel has started up successfully. The Controller then activates IRQBIT much as it would normally and Kernel responds by activating HALTBIT even though it has not actually gone through the normal interrupt handler entry. From this point on both programsuse normal handshake.

The DDT51 Controller uses a timeout whenever it waits for the Kernel code to respond to an interrupt request, because the Kernel can be damaged by anerrant Target Program instruction. Except while running to a breakpoint, the normal Kernel re-

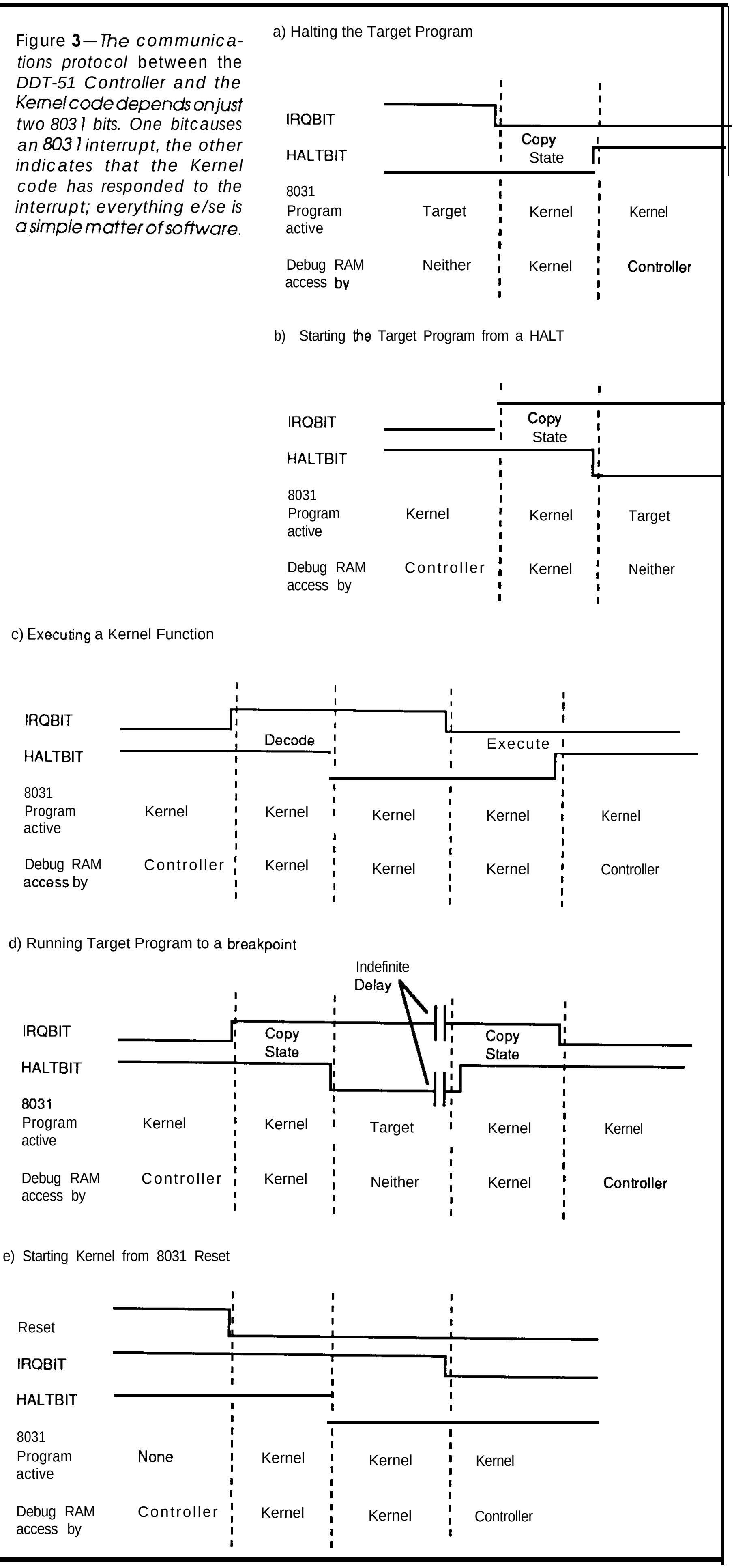


Figure 3—*The communications protocol between the DDT-51 Controller and the Kernel code depends on just two 8031 bits. One bit causes an 8031 interrupt, the other indicates that the Kernel code has responded to the interrupt; everything else is a simple matter of software.*

sponse time does not exceed a few hundred milliseconds and is generally a few tens of microseconds.

MODELING CPU REALITY

Up to this point, I've described the DDT51 Controller as telling the Kernel what to do. While that's true, the action really begins at the PC where you are expecting new values to appear on the screen right after halting the Target System.

Although the Kernel code copies the CPU's state into Debug RAM, it is not practical to refer to those values every time they're needed, so the DDT-51 Controller copies them from the Debug RAM into its own RAM. Those values are the ones sent to the SONOFDDT PC program each time the Target Program halts.

You can change many registers and RAM locations while the Target System is halted, so SONOFDDT also maintains a copy of the CPU state in PC RAM. Those values are sent to the DDT51 Controller, which forwards them to the Debug RAM where the Kernel code copies them back to the CPU. All told, the 8051 CPU state can be summed up in about 150 bytes and SONOFDDT sends only changed bytes to reduce the transmission time.

But SONOFDDT must also display sections of the External RAM: the disassembled program and storage dumps. It is not practical to transfer all 64K bytes whenever the Kernel code gets control!

The solution is an External RAM cache in the PC's memory. Every time a field needs an External RAM location, SONOFDDT checks its cache for that address. If one of the cache blocks matches, the RAM byte is immediately available. Otherwise, SONOFDDT asks the Controller for a new block, the Controller issues a Kernel function to copy the block from the 8031's address space, and the Kernel goes through the whole dance described above. The desired value returns to the PC embedded in a 64-byte block, so a request for a nearby location is likely to be a cache hit.

When you change a byte in External RAM, SONOFDDT sets a "changed" flag for the cache block holding that byte. When you tell SONOFDDT to run another Target Program instruction, it sends that block to the DDT51 Controller as part of the CPU state. If you make no changes, SONOFDDT need not send any blocks at all.

DOING BUSINESS AS...

The 8031 architecture poses a debugging challenge that simply doesn't appear in most other CPUs: aliases. A single Internal RAM location can be a working register, a byte on the hardware stack, a program variable, and pointer to another Internal RAM location-all at the same time.

Snap quiz: which locations meet this description and how many of them are there? Extra credit essay: would you ever deliberately set SP to the value needed to push a byte into one of those locations? Why or why not?

Also unlike most CPUs, the 8031 has an extensive set of bit manipulation instructions, in addition to the more familiar byte-wide Boolean instructions. Many of the SFRs (Special Function Registers), ALU registers, and a whole section of Internal RAM aremadeupofbits that maybeflipped at will. The bit names are separate from the register names, but affect the same hardware: aliases at work!

The PSW (shown in Photo 3, near the middle of the screen) is an excellent example of this situation. It is displayed in three ways: a hex byte, an ASCII character, and six individual bits. The two remaining PSW bits appear in the Working Register block, where they identify the active bank.

The SONOFDDT program reads this file to assign symbolic names to storage addresses.

```
AS DemoD2 D:31                 Internal RAM variable
SY DemoB1 B:0                  Internal Bit variable
AS DeLay C:100                 Subroutine
AS BLINKBIT B:b4
AS inner2 C:IOb                Branch target
AS inner1 C:105
AS TimeOn D:30
AS Blinker C:5b                Subroutine
AS DecLow C:7d
AS TimeOff X:1000
AS P1OK C:65
AS DemoX2 X:1001
SG BIT B:0 0 RW                Read-write bit segment
SG ANON0000 D:30 32 RW         Read-write data segment
SG ANON0001 X:10000 1002 RW    Read-write external segment
SG ANON0002 C:50 113 R         Read-only code segment
```

The first two characters identify the type of symbol (Absolute Symbol, SYmbol, or SeGment). The next field is the symbol name, in mixed case. The single character before the colon identifies the address space (Bit, Code, Data, external), while the hex number after the colon is the starting address. Segment lines include the ending address as well as a Read/Write or Read-only marker.

Figure 4—*The Avocet* SYM *file* format. The SONOFDDT program reads *this file* fo assign *symbolic names to storage addresses.*

Not only must the SONOFDDT program distribute the bits, but it must also update everything when you change any one. While this can occur withotherdebuggers, the8031 takesit to extremes. Consider changing the Bank Select bits in the PSW: the PSW changes, but so do all eight Working Registers.

The C-scape library used to build SONOFDDT'suser interface defines the screen in terms of fields, where each item on the screen is a single field: each bit, register value, and ASCII character is a separate field with a predetermined set of characteristics. Whenever a keystroke occurs within a field, SONOFDDT updates the value shown on the screen, then checks to see if any other fields must change. Even in the worst case, the code must update only a few dozen characters, so there is no problem keeping up with a fast typist or repeating keystrokes.

SYMBOLICALLY SPEAKING

The ultimate source of the program in the Target System is a PC disk file produced by an assembler and linker. While that raw hex information is what actually runs the 8031 CPU, we humans tend to think about the program using the names we gave to subroutines, branch targets, variables, and so forth. The linker summarizes that information in the SYM file, so SONOFDDT has **ready** access to all of your program's symbols.

Two areas of the screen shown in Photo 3 display sections of Internal and External RAM. SONOFDDT allows you to select the dump addresses either directly as hex numbers or by picking a variable name from a list taken from the SYM file. You can also dump RAM based on RO, R1, DPTR, and other addressing combinations used by the CPU.

The left half of the screen shows disassembled instructions near the current Program Counter value. The SYM file provides names for subroutines, statement labels, and internal, extemal,and bit variables. SONOFDDT decodes the instructions to insert the proper symbol in the operand field, as well as an arrow pointing to branch targets.

Because the DDT-51 Controller and the Kernel have such limited storage, all the symbolic manipulation takes place in the PC under SONOFDDT's control. For example, you can set a breakpoint picking the label from a list. SONOFDDT converts that name into the corresponding numeric value and tells the Controller to set a breakpoint. The Controller adds the breakpoint address to a list in Debug RAM. The Kernel uses that list to insert and remove breakpoints when it starts and stops the Target Program.

The Avocet assembler and linker also provide storage segments. Despite the rather dismal reputation segments have achieved in the PC world due to the Intel 80x86's 64K-byte size limit, they remain a valuable way to organize your program's storage. SONOFDDT takes advantage of these segments to detect improper storage accesses.

When you define an Avocet code segment, the linker enters the segment boundaries in the SYM file and adds a "read-only" flag to the line. SONOFDDT passes those entries to the Controller, which builds a table of read-only segment boundaries in Debug RAM.

Just before Kernel passes control to the Target Program, it computes a checksum for each read-only segment. Whenever it regains control, it checksums the segments again. The Controller compares the two values and reports an error for any mismatches. In principle, a running program should never modify its code, so any changes indicate a bad pointer or runaway code.

This is particularly valuable in single-step mode, where you learn of the damage right after offending instruction. While Son of DDT can't prevent code damage, immediate detection tracks down program bugs that would otherwise remain invisible.

If your linker doesn't produce a symbol table file in the Avocet format, you can write a filter program (using your favorite PC language) to convert your SYM files. If that isn't practical, you can use a time-honored trick: put all your variables at fixed locations and create a SYM file describing them. Unless you can force your subroutines to fixed locations, you won't be able to refer to them by name, but it's surely better than nothing. Figure 4 shows a sample SYM file.

DECLARE WAR ON BUGS

Although Son of DDT has many more features than the original parallel DDT51, the best way to find out

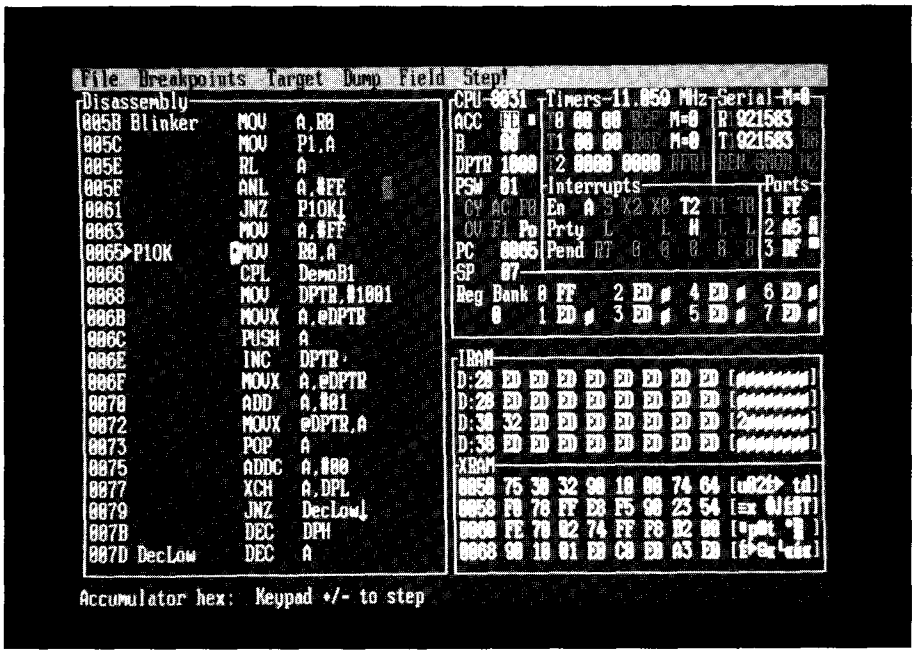


Photo 3—*The Son of DDT software displays everything about the target processor on one easy-to-read screen.*

what it can do for you is to download the SONOFDDT demo. The demo version of the program provides all the "look and feel" of the real program but doesn't require the Son of DDT hardware. While you can't debug a program, you can get a good feel for how the user interface works on your system. *[Editor's Note: Software for this article is available on the Circuit Cellar BBS or on Software On Disk #22. See page 106 for downloading and ordering information.]*

A parallel port version of the new Son of DDT software, compatible with the original parallel port connected DDT51, will be available in September as an upgrade.

Finally, in the spirit of Steve's original DDT-51 design and other Ciarcia's Circuit Cellar projects, Circuit Cellar will provide full-function software to anyone who builds Son of DDT hardware from scratch. There will be a nominal charge for materials and shipping. Realize of course, that "from scratch" means just that: Connecting purchased RTC31 and DDT-51 printed circuit boards does not qualify. Instead, you'll have to build the original DDT51 as described by Steve in volume 7 of "Ciarcia's Circuit Cellar" and the RTC31 in CIRCUIT CELLAR INK issue #8. Send pictures of your finished assembly to Steve at the address below and join our war on bugs. ✣

Ed Nisley is a Registered Professional Engineer and a member of the Circuit Cellar INK engineering staff. He enjoys finding innovativesolutions todetnandingand unusual technical problems

IRS

4 13 Very Useful
414 Moderately Useful
415 Not Useful

SOURCE

Circuit Cellar Kits
4 Park St., Suite 12
Vernon, CT 06066
(203) 875-2751
Fax: (203) 872-2204

1. Son of DDT assembled and tested in a 5.5" x 7" x 1.5" metal enclosure with power supply and target RTC31 controller mounted on top surface. Comes with user's manual and software on disk. System functions on any PC or AT clone with 640K, serial port, and any standard display. $499

2. Son of DDT software upgrade for use on original parallel port connected DDT51. Available in September '91. Comes with user's manual and diskette $150

Items above do not include shipping and handling. Please call for appropriate costs.

FEATURE ARTICLE

Eduardo Pérez
& Dapang Chen

Numerical Applications Using DSP

Using a DSP Chip for High-Speed Numeric Processing

Historically, discrete-time domain or DSP algorithms have closely tracked the development of digital computer technology. Before the 1960s, the technology for signal processing was mostly analog. Electronic circuits and even mechanical devices processed signals in the continuous-time domain, The introduction of digital computers in the 1950s brought signal processing into the digital domain. The limited computational capabilities of the early digital computers restricted their use to numerical computations and some simulation of analog systems. Computers were treated as speedy calculators. Hardware aside, lack of DSP theory also made the use of digital computers for DSP operations difficult at best.

The discovery of the fast Fourier transform (FFT) by Cooley and Tukey in 1965 toimplement thediscrete-time Fourier transform (DFT) was a quantum leap in DSP technology. It reduced the computation time of the DFT from $O(N^2)$ to $O(N \times Log_2(N))$, making DFT spectrum analysis computationally feasible. The new FFT algorithm also promoted development of special hardware dedicated for the FFI'. Most importantly, the new FFT inspired development of a complete theory of discrete-time mathematics. The FFT is not a simple approximation or simulation to an analog system but is exact in the discrete-time domain. The FFT has become a key component of spectral analysis, and FFT-related algorithms have been studied and used extensively.

In addition to discrete-time domain spectral analysis, many other applications are in the discrete domain by nature. For instance, many differentialequationsand integrations are not analytically solvable and must be solved by numerical approximations. Finite element analysis is another largebranch of discrete-timedomain applications that is extremely important to the study of aerodynamics. These types of applications usually involve a set of grid points on which the evaluations are made. The grid points must often be dense enough to avoid irregular behaviors between them. A denser set of grid points means more points to be evaluated, which slowsdown thecomputation. Often computational complexity is proportional to the number of points: $O(N)$, $O(N^2)$, or even $O(N^N)$. Control applications often require matrix operations to solve a system equation and to determine the stability of the control system. DSP and numerical applications often require floating-point processing for accuracy, precision, and dynamic range. In the past, software performed floating-point number operations, a very slow process in computationally intense algorithms. Dedicated floating-point processing hardware was expensive and not readily available.

Hardware technology, especially VLSI technology of recent years, has made the floating-point processor affordable and available. A floating-point processor is often called a math coprocessor. The instruction set of the coprocessor includesmany frequently used floating-point instructions, arithmetic operations, transcendental operations, and logarithmic operations. Usually, the math coprocessor is designed as an integral part of a computer system. The instruction set often becomes part of the computer's instruction set and can be directly controlled and programmed from the CPU. The interface and communications between the CPU and a math coprocessor are very simple to a programmer. High-level language users need not be concerned about the existence of the math coprocessor. By using the math coprocessor, however, floating-point computational time is reduced by orders of magnitude.

DSP hardware is quite different from the math coprocessor. The chip is independent of the host computer and can be built as a stand-alone system. DSP hardware, such as the Texas Instruments (TI) TMS320Cx0 series, is designed to maximize certain math operations (such as parallel multiply and add) **commonly** used in spectral analysis and numerical applications. Most DSP chips can complete any instruction in one instruction cycle, including the floating-point multiplication and addition made possible by special floating-point hardware. To keep everything in one cycle, however, DSP chip instructions do not contain many floating-point instructions. For instance, the floating-point division instruction does not exist in the instruction set of most DSP chips. The instruction is implemented via software. This strategy is true for all transcendental and logarithmic operations and is very similar to the reduced instruction set computer (RISC) where a smaller core set of instructions is optimized at the expense of other less frequently used instructions. In addition, DSP chips are capable of some special addressing modes unique to FFT applications such as bit-reverse addressing and circular addressing. DSP chips can also handle DMA data transfer and interrupt handling. Therefore, DSP hardware is not just a simple improvement of the math coprocessor; it is a unique hardware architecture suitable for spectral analysis and other numerical applications. In the following sections, the hardware and software features of DSP chips are briefly introduced and application examples are discussed.

FEATURES OF DSP HARDWARE AND SOFTWARE

Dedicated Floating-Point Hardware and Parallel Instructions

The dedicated hardware multiplier provides the bulk of a DSP's processing power. A floating-point multiplication is executed in a single instruction cycle. In addition to fast floating-point multiplication, increased performance is achieved by parallel instructions. Typically, a multiplication and addition can be executed in parallel in one instruction cycle. For example, the following 96002 instruction can execute three floating operations in one instruction cycle:

```
FMPY X0,X1,D1 FADDSUB.X D3,D2
```

The instruction multiplies the contents of X0 and X1 and stores the result in D1, adds D3 and D2 and stores the result in D2, and subtracts D2 from D3 and stores the result in D3 in parallel. This instruction is very efficient for FFT butterflies, as shown in Figure 1.

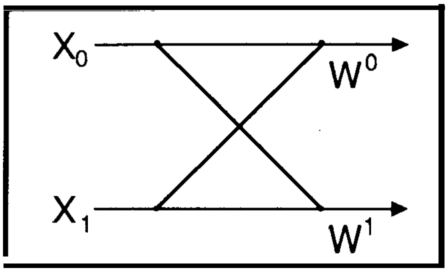


Figure 1—*FFT butterfly signal diagram.*

Pipelining

Pipelining is common to all DSP chips. A pipeline architecture breaks a computation into separate stages for instruction fetch, decode, operand fetch, and execution. While one instruction is being executed, the remaining instructions can be fetched and decoded. Pipelining is automatic whenever possible. When standard branch instructions are encountered, the prefetched instructions in the pipeline are no longer valid, and the pipeline is flushed. Some DSP chips normally have delayed branch instructions that do not require flushing the pipeline.

Special Addressing and Zero-Overhead Loop

DSP chips also have many special instructions that are useful to carry out the FFT and digital convolution operations. For instance, the following TI TMS320C30 instructions implement a 100-tap dot product. One tap is the accumulation of one multiplication:

$$y = \sum_{k=0} h_k x_k \qquad (1)$$

```
LDI    @A,AR0   ; addr. of h
LDI    @X,AR1   ; addr. of x
LDF    0.0,R0
                ; first product
MPYF3 *AR0++(1),*AR1++(1),R1
RPTS  98        ; rep. 99 times
MPYF3 *AR0++(1),*AR1++(1),R1
|| ADDF3 R1,R0,R0 ; accum. result
ADDF3 R1,R0,R0 ; final accum.
```

There are two features in these instructions: zero-overhead loop, and parallel multiplication and accumulation. Zero-overhead loop means that

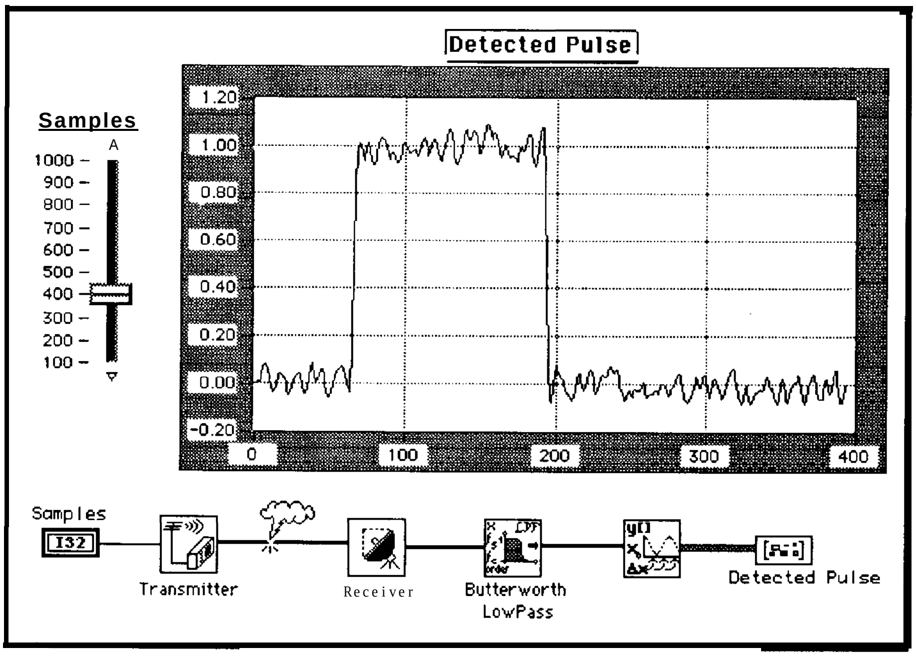


Figure 2—*This graphical pulse demo example was developed with LabVIEW 2 from National Instruments.*

there is no penalty for RPTS when it is set up (three cycles for setup). The parallel instruction "||" means that both MPYF3 and ADDF3 can be executed in the same instruction cycle.

Fast and Parallel Memory and Peripheral Access

DSP chips such as the TI TMS320C30 chip have separate program buses, data buses, and DMA buses for parallel program fetches, data reads and writes, and DMA operations. Many DSI' instructions can access two memory operands at the same time, providing an excellent platform for complex operations such as the complex FFT. An on-chip DMA controller provides parallel data movement between the DSP chip and the external peripherals, which are usually much slower than the DSP chip. Many add-on DSP boards, such as the NB-DSP2300 from National Instruments shown in Photo 1, also provide an additional on-board DMA controller for convenient data transfer.

DSP SOFTWARE DEVELOPMENT

The hardware features of DSP chips make them quite attractive for a variety of applications. They are fast, flexible, and economical. But, hardware without software is hardly useful, and is especially true for DSP systems. To fully use the computational power of a DSP chip, software must be written to take advantage of the architecture and addressing modes of the DSP. Naturally, this places a higher requirement on the DSP programmer.

To simplify software development, most DSP chip manufacturers have DSP software development tools: a high-level language, assembler, linker, debugger, and simulator. The high-level language is a convenient, quick, and efficient way to begin development. In this case, much of the existing software can be easily ported onto a DSP platform, saving development time. With the assembler, the user can have complete control over the DSP hardware for the best possible performance, which is necessary for many timing and interrupt-related tasks. The simulator provides a good testing environment. Before running a program on the hardware, the user can test it using the simulator to make sure that all the requirements are met.

The current trend is to provide a graphical user interface not only to develop numerical analysis and DSP-based applications quickly and efficiently but also to be able to directly interact with the system itself as is the case of LabVIEW, a graphical DSP software development system. An example appears in Figure 2 where the user interaction is through front-panel controls and the actual program is a block diagram.

NUMERICAL ANALYSIS AND DIGITAL SIGNAL PROCESSING

Numerical analysis is the branch of mathematics concerned with the

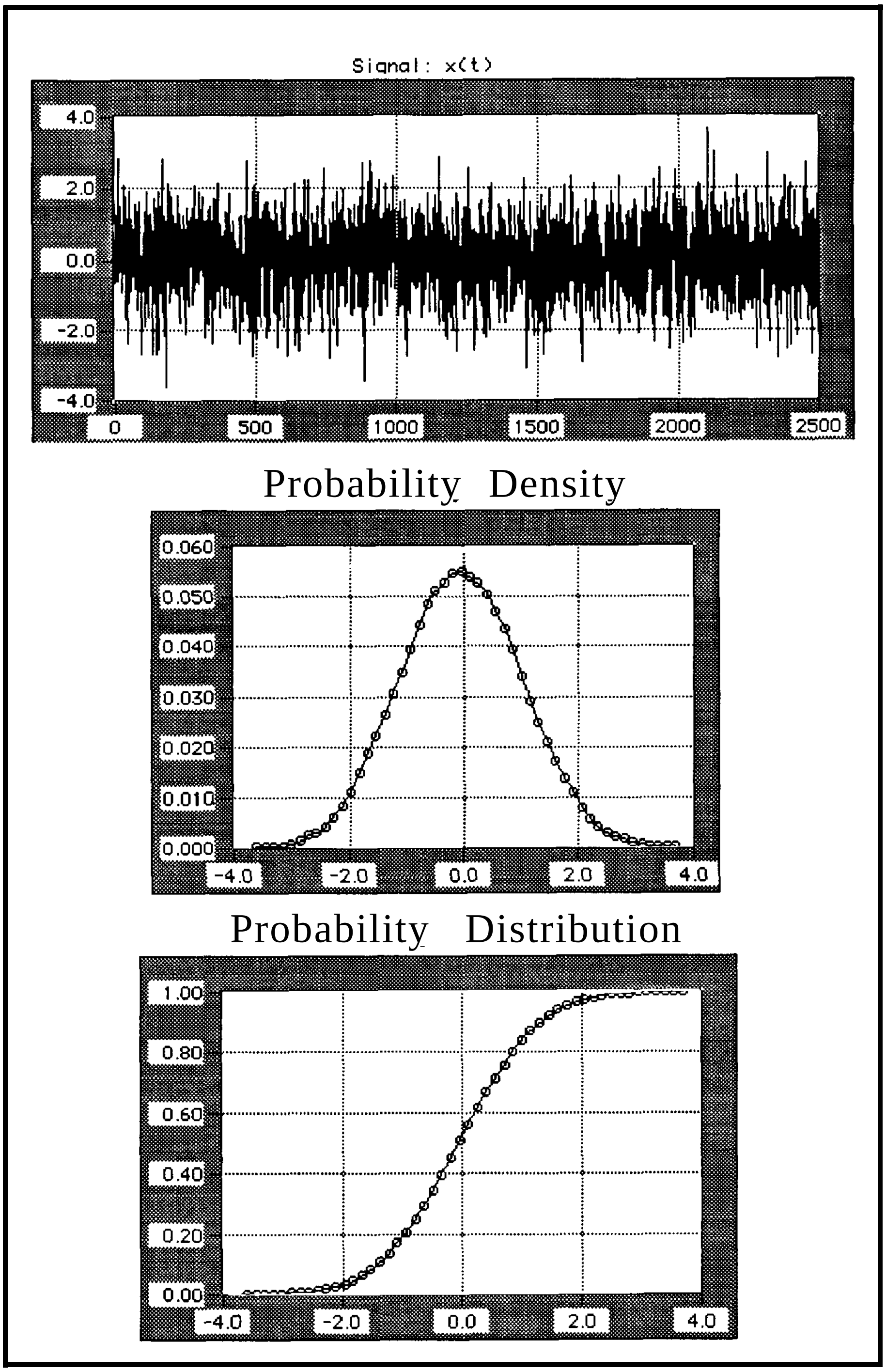


Figure 3—*Computed Probability Density and Distribution functions of a Gaussian distributed noise signal.*

development of algorithms and efficient implementations to obtain approximate solutions of mathematical expressions. An algorithm is a sequence of logical steps or operations that, when completed, produces a result with specified accuracy. Although many numerical analysis techniques have been known for centuries, the advent of floating-point coprocessors and DSPs in mainframes, workstations, and personal computers has caused renewed interest among scientists, researchers, and engineers in the area of numerical analysis.

Floating-point DSP architectures provide a mechanism by which high-accuracy and high-efficiency numerical algorithms can be implemented. Since floating-point DSP architectures address the accuracy and efficiency issue, the remainder of this discussion focuses on basic numerical analysis techniques that can easily be implemented in DSP systems.

Numerical analysis and DSP software applications are largely based on two software structures: The sum of products structure

$$y_n = \sum_{k=0} h_k x_k \qquad (2)$$

and the iterative structure

$$y_{k+1} = f(y_k, y_{k-1}, \dots, y_0, x_k) \qquad (3)$$

By using the sum of products and iterative structures, sophisticated libraries of mathematical functions, digital filtering functions, and DSP functions can be built. Implementing accurate and efficient algorithms on any digital computer, especially often-used mathematical functions, is essential to the development of advanced mathematical libraries, digital filtering libraries, and DSP libraries. Examples in numerical methods, digital filtering, descriptive statistics, and Fourier analysis are presented.

NUMERICAL ANALYSIS

Although often taken for granted in most high-level languages, many mathematical functions, such as cos(x) and sqrt(x), have been implemented using the two structures discussed previously. There are three common methods to evaluate these functions: series expansion, polynomial approximation, and iterative approximation.

SERIES EXPANSION

Series expansion involves expressing a function f(x) as a sum of products of the form

$$f(x) = \sum_{k=0} a_k x^k \qquad (4)$$

such that

$$\lim_{k \to \infty} a_k x^k = 0 \qquad (5)$$

where a_k is the set of series expansion coefficients. Notice that this form of equation (4) is the same as that of equation (2) where $x_k = x^k$. Consider the function f(x) = cos(x) which has the infinite series expansion

$$\cos(x) = \sum_{k=0} \frac{x^{2k}}{(2k)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots \qquad (6)$$

The pseudocode below shows a possible implementation on a digital signal processor with a maximum absolute error of 10^{-7}.

```
max_error = 1.0E-7
cos x = 1.0
num = x_squared = x * x
den = 2.0;
y = num / den

k = 2
sign = -1.0
while (y > max_error)
    cos_x = sign * y + cos_x
    num = num * x_squared
    k = k + 2
    den = den * k * (k-l)
    sign = -1.0 * sign
    y = num / den
end while
return  cos x
```

POLYNOMIAL APPROXIMATION

The polynomial approximation is a finite representation of a series expansion representation because higher order terms generally have little bear-

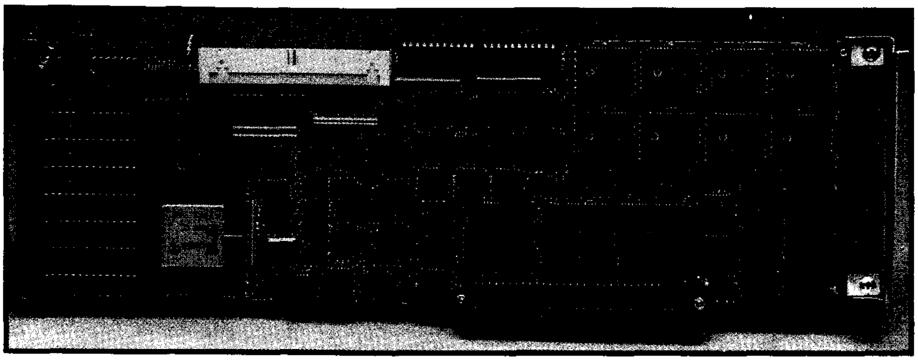


Photo 1 -The *NB-DSP2300 delivers 33.33 MFLOPS of computationalpower, and contains a high-speed DMA controller which supportsNuBus blockmode transfers of33.7 Mbytes/sec. It can be programmed with several software options.*

ing on the final result when implemented on a digital computer or processor. For example, the cosine function can be approximated by the following polynomial:

$$\cos(x) = 1 + a_1x^2 + a_2x^4 + a_3x^6 + a_4x^8 + a_5x^{10} \quad (7)$$

where

a, = -0.4999999963
a, = 0.0416666418
a_3 = -0.0013888397
a_4 = 0.0000247609
a, = -0.0000002605

and produce results with a maximum absolute error of 2×10^{-9} for $0 \le x \le 0.5\pi$. Equation (7) can be rewritten as

$$\cos(x) = 1 + (a, + (a, + (a, + (a, + a_5x^2)\,x^2)\,x^2)\,x^2)\,x^2 \quad (8)$$

and the following pseudocode can be used to evaluate the cosine function:

```
x_squared = x * x
Y = a[5]
i = 4

while (i > 0)
    Y'Y   * x_squared + a[i]
    i = i - 1
end while
cos_x = Y * x_squared t 1.0
return  cos x
```

ITERATIVE APPROXIMATIONS

A function cannot always be evaluated in terms of series expansion or polynomial approximation. A function can alternatively be evaluated by successivelyapproachingtheresultin an iterative manner. Before this technique is described, consider the Taylor series expansion of a function f(x):

$$f(x) = \sum_{k=0}^{\infty} \frac{(x-a)^k}{k!} f^{(n)}(a) \quad (9)$$

where

$$f'''(a) = \left.\frac{d^n f(x)}{dx^n}\right|_{x=a} \quad (10)$$

Using a truncated Taylor series to the first-order term, f(x) can be rewritten:

$$f(x) \approx f(a) + (x - a)\, f(a) \quad (11)$$

To illustrate how functions can be evaluated using iterative approximations consider the square root function

$$y_k = f(x_k) = \sqrt{x_k} \quad (12)$$

By substituting equation (12) into equation (11) and rearranging the terms, an iterative expression is obtained to evaluate the squareroot function as in (equation 13)

$$y_{k+1} = \frac{y_k}{2} + \frac{x}{2y_k} \quad (13)$$

To initiate the whole process, let $y_0 = 1$. The following table shows the iteration number and the evaluation of the square root of 9 to seven significant figures at the specified iteration:

k	y_k
0	1 .0000OOo
1	5.0000000
2	3.4000000
3	3.0235294
4	3.0000916
5	3.0000000

The following pseudocode shows a possible implementation of the square root function:

```
max_error = 1.0E-7
y = 1.0
a = 0.5 * x
sqrt_x = a t 0.5

while (|sqrt_x - y| > max error)
    y = sqrt_x
    b = a / y
    sqrt_x = 0.5 * y + b
end while
return  sqrt_x
```

NUMERICAL ANALYSIS APPLICATIONS

Numerical analysis techniquesare applicable to a wide variety of fields and can be greatly enhanced by DSP hardware and software architectures because of the recurrence of the sum of products and iterative structures in these fields. The ability of DSP chips to multiply and add quickly make them suitable for an overwhelming number of numerically intensive analysis applications such as statistical analysis, graphics, linearalgebra, digital filters, Fourier and spectral analysis, and image processing.

The following section will briefly discuss some of the above applicationsand givesa summary of themost important equations in the respective fields. All the equations presented have a sum of products or iterative form, or both.

STATISTICAL ANALYSIS

Statistical analysis methods are often applied in human and biological sciences such as sociology, psychology, meteorology, economy, and business and administration. The methodology consists of obtaining experimental data under controlled environments to later establish a statistical relationshipbetween thegathered data and the controlled parameters.

A good example is provided by computing the root-mean-squared (RMS) value of the set $X = \{x_0, x_1, x_2, \ldots, x_{n-1}\}$. The RMS value of X, Y_x, is evaluated as:

$$Y = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i^2} \qquad (14)$$

and the pseudocode implementation is

```
i = 0
sum = 0.0
while   (i <  n)
    sum = x[i]  * x[i]+   sum
    i = i + 1
end  while
rms =  sqrt  (sum /  n )
return  rms
```

With careful implementation, the contents of the while loop can execute in one clock cycle because it is a sum of products structure. Furthermore, the final computation of the RMS value requires the square root value which, as discussed previously, can be evaluated using iterative techniques.

Similar to the RMS example, advanced statistical analysis methods such as probability densities and distributions (see Figure 3), tests of hypotheses, analysis of variance, curve fitting, and nonparametric tests rely on the sum of products and iterative structures.

VECTOR AND MATRIX OPERATIONS

Statistical analysis, three-dimensional graphics, and process control are only a few of the fields that require manipulation of large data sets in the form of vectors and matrices. The sum of products and iterative structures are important to these fields, but zero-overhead loops also play an important role because, in many cases, matrix operations can be extremely repetitive, and the number of computations can be of the order of n^3, where n is the size of a square matrix (see Figure 4).

The prime example for this section is the computation of the dot product of two vectors x and y, equation (15), because this computation is only the sum of products.

$$x \bullet y = \sum_{i=0}^{n-1} x_i y_i \qquad (15)$$

where

n is the number of elements in x and y

x_i is the i^{th} element of x

y_i is the i^{th} element of y

Consider the matrix multiplication operation. For illustrative purposes, let **A** and **B** be two n x n matrices. The elements of the resulting matrix C = A x B are obtained using:

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j} \qquad (16)$$

which is the dot product of the i^{th} row of A and the j^{th} column of B. This operation required n multiplications and n additions and must be carried out for each element of C, which contains a total of n^2 elements. Thus, the total number of operations required to compute this matrix multiplication is n^3 multiplications and n^3 additions.

DIGITAL FILTERS

Digital filters (Figure 5) are being used to replace analog filters because

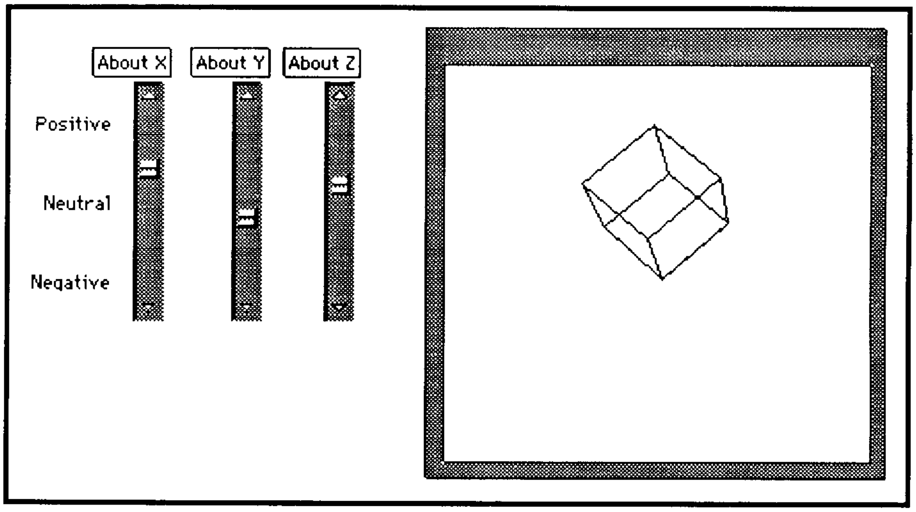


Figure 4—*Rotation and translation of a 3D object is easily accomplished with the aid of vector and matrix operations.*

of high-speed, high-efficiency, fixed-point, and floating-point DSP chips. Digital filters have had a great impact on instrumentation, automatic test and measurement, and speech synthesis and analysis—only a few of the possible applications.

Two common techniques for designing digital filters are the Finite Impulse Response (FIR) and the Infinite Impulse Response (IIR) filter design techniques. The difference between FIR and IIR filters is that FIR filters use the sum of products structure, while the IIR filters use the iterative structure. The sum of products process is a finite process (FIR), and the iterative process, at least in theory, is an infinite process (IIR).

FIR FILTERS

Design and implementation of FIR filters involve: obtaining the filter coefficients, and filtering the incoming data. The filter coefficients can be derived from the frequency domain specifications and are the subject of many introductory DSP and digital filtering textbooks.[6,7]

For example, the low-pass FIR filter coefficients $h = \{h_{-m}, h_{-m+1}, \ldots, h_{-1}, h_0, h_1, \ldots, h_{m-1}, h_m\}$ can be obtained using

$$h_i = 2f\,\Delta t\ \mathrm{sinc}\,(2i\,f_c\,\Delta t) \qquad (17)$$

where

$$\mathrm{sinc}(f) = \frac{\sin(\pi f)}{\pi f} \qquad (18)$$

Δt is the sampling interval
f_c is the cut-off frequency
$2m + 1$ is the total number of coefficients

If the input to the filter is the sequence of values $x = \{x_0, x_1, x_2, x_3, \ldots\}$ and the output filtered sequence is $y = \{y_0, y_1, y_2, y_3, \ldots\}$, then the k[th] element of the output sequence y is obtained using the following formula:

$$y_k = \sum_{i=-m}^{m} h_i x_{k-i} \qquad (19)$$

The filter design results from equation (17) generally show the need to evaluate special math functions to derive the filter coefficients. The actual filtering function is carried out by implementing equation (19). Equation (19) is the discrete implementation of the convolution integral and is discussed in a separate section later.

IIR FILTERS

The canonical form of analog filter designs in the complex frequency domain (the s-plane) is shown in equation (20):

$$H(s) = \frac{\prod_{i=0}^{m-1} (s - s_i)}{\prod_{i=0}^{n-1} (s - p_i)} \qquad (20)$$

where
s_i is the location of the i[th] zero in the s-plane
p_i is the location of the i[th] pole in the s-plane
m is the number of zeros
n is the number of poles

Most frequently, the Bilinear Transformation is used to map the canonical filter form representation H(s) into a suitable set of IIR filter coefficients. The Bilinear Transformation consists of evaluating H(s) at

$$s = \frac{1 - z^{-1}}{1 + z^{-1}} \qquad (21)$$

Substituting equation (21) into equation (20), expanding, rearranging, and normalizing the factors, a Z transform H(z) is represented as follows:

$$H(z) = \frac{\sum_{i=0}^{m-1} a_i z^{-i}}{1 + \sum_{j=1}^{n-1} b_j z^{-j}} \qquad (22)$$

The representation obtained corresponds to a discrete-time, recursive system represented by the following difference equation:

$$y_k = \sum_{i=0}^{n-1} a_i x_{k-i} - \sum_{j=1}^{m-1} b_j y_{k-j} \qquad (23)$$

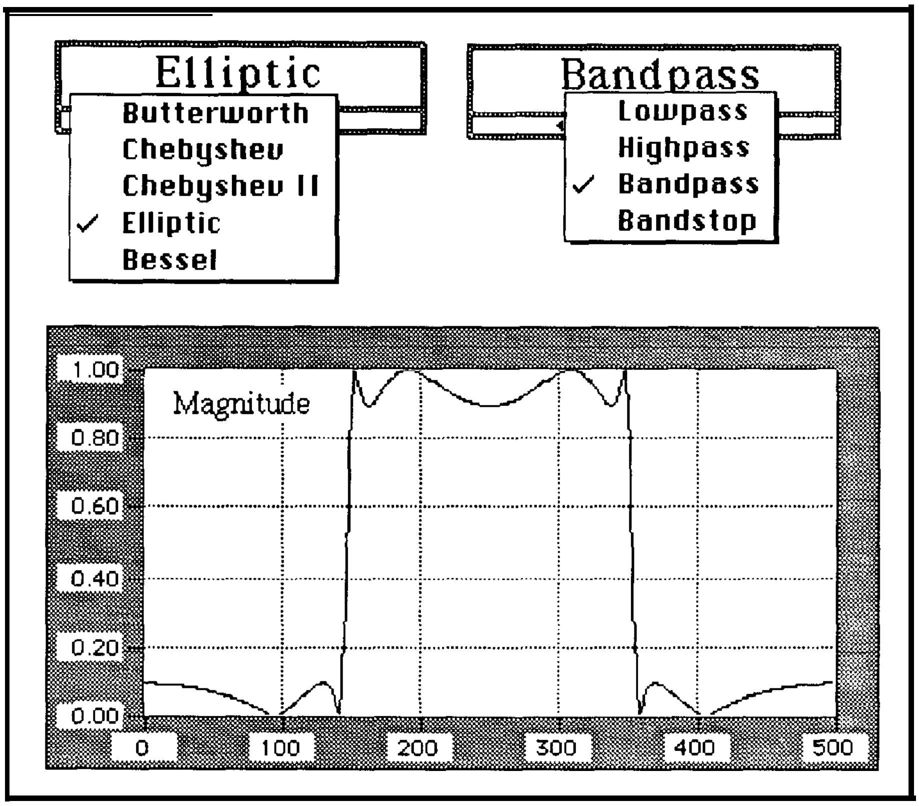


Figure 5— Designing *and implementing digital filters requires the use of the sum of products and iterative techniques.*

where

- a_i is the i^{th} the forward IIR filter coefficient
- b_i is the i^{th} the feedback IIR filter coefficient
- n is the total number of forward coefficients
- m is the total number of feedback coefficients

Similar to FIR filer design and implementation, the computation of the forward and feedback IIR coefficients require the evaluation of special math functions. The recursive filter form in equation (23) is the difference of two sum of product terms.

Digital filter design, whether it be FIR or IIR, requires efficient implementationofadvancedmathfunctions such as trigonometric, hyperbolic, Bessel, and elliptic integrals. Real-time filterimplementationsrequirefastand efficient computation of the sum of products. Consequently, DSP systems provide a highly sophisticated environment in which to implement digital filters.

DSP

Many factors have contributed to the rising interest in the development of DSP-based systems: inexpensive DSP chips, consumer electronics, DSP market share, and so on. Because DSP systems gained popularity over the course of a few years, some believe that DSP development is somewhat of a black art. In fact, DSP is the application of basic numerical techniques to high-level mathematicalconcepts. The primary operations in DSP are the convolution and the Fourier Transform.

CONVOLUTION

Linear, time-invariant systems can be modeled using the convolution operation. Convolution simply mixes two signals, one that represents the input signal and one that represents the system's impulse response, to produce an output signal. In the discrete-time implementation, let x be the input sequence, h be the discrete impulse response, and y be the resulting output sequence. The elements of y are obtained using

$$y_k = \sum_{i=0}^{n-1} h_i x_{k-i} \qquad (24)$$

Equation (24) is known as the discrete implementation of the convolution integral and is the basis of many physical systems and models. Equation (24) is optimally implemented in DSP-based systems. Furthermore, the digital filter models presented previously are based on the convolution operation, where the filter coefficients have been precomputed to obtain a desired effect on the input signal.

Determining the system'simpulse response in order to be able to model and predict the behavior of the system under different conditions and signal is a more realistic problem. When this is accomplished, a digital system can be implemented in the form of a convolution to enhance or compensate for deficiencies in the system because the convolution operation is a linear operator.

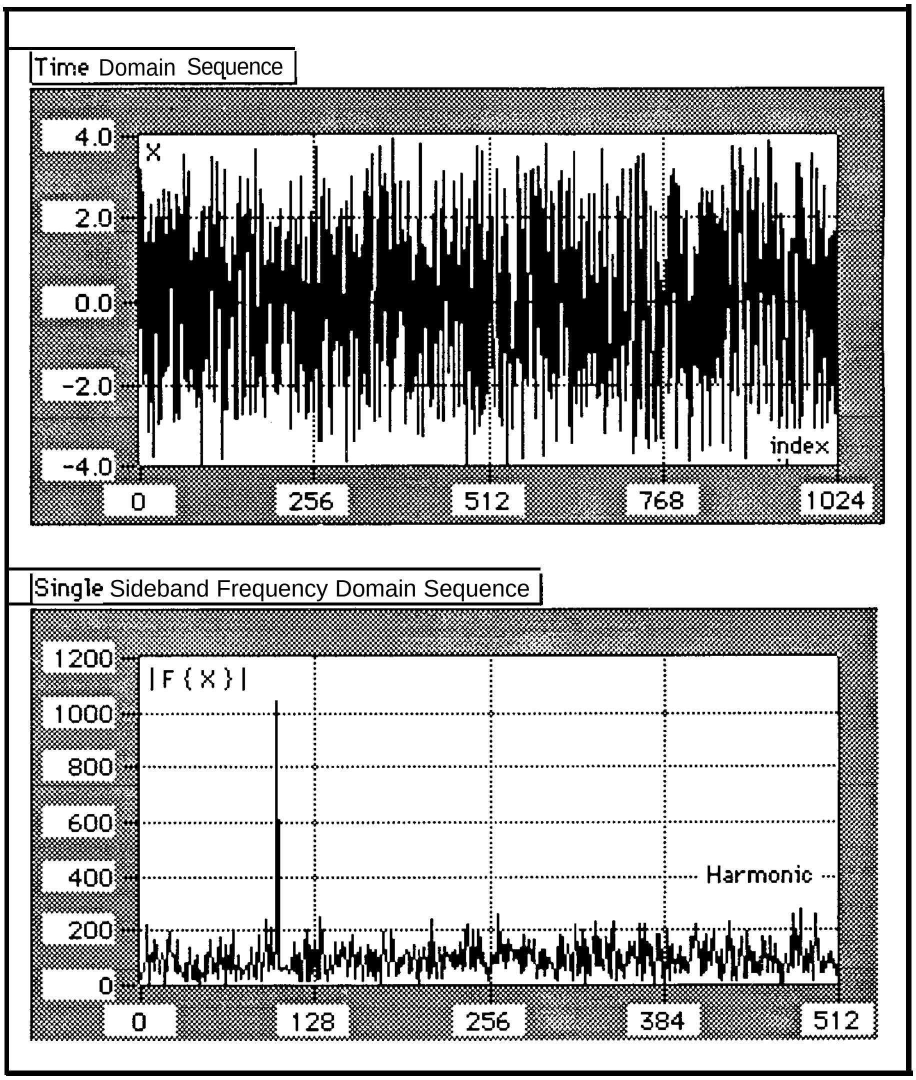


Figure 6-A *time-domain signal and its Fourier transform*

FOURIER TRANSFORM

Fourier Transform is a powerful analysis tool, applicable to fields such as spectral analysis, telecommunications, seismography, instrumentation, vibration analysis, medical imaging, optics, and acoustics.

The Fourier Transform determines the harmonic components of a time domain signal. The discrete implementation of the Fourier integral is known as DFT and is summarized by equation (25).

$$Y_k = \sum_{i=0}^{n-1} X_i W^{ik} \quad \text{for } k = 0, 1, 2, \ldots, n-1 \qquad (25)$$

where

$$W = e^{-2j\pi/n} = \cos\left(\frac{2\pi}{n}\right) + j\sin\left(\frac{2\pi}{n}\right) \quad (26)$$

$$j = \sqrt{-1}$$

X maybe a real or complex valued array
Y is a complex valued array
n is the total number of discrete samples

Figure 6 shows the graphical result of performing an FFT on an acquired time-domain sequence.

Efficient DFT implementations are known as FFTs and are the subject of many introductory DSP textbooks.[8,9] From the numerical analysis point of view, the FFT is a series expansion using complex terms. The performance is related to the implemented algorithm as well as to the efficiency of support library routines.

The Fourier transform and its digital implementa tionare important analytical tools and DSP chips and their architecturesaredesigned tooptimize the computation of the FFT. The basis of this design is the ability to multiply and accumulate in one clock cycle.

MORE THAN FOURIER

Digital signal processing technology is an excellent environment for implementing and developing numerically intensive analysis applications. The heart of a DSP system is a simple but elegant architecture consisting of a multiplier and an accumulator. With this architecture, series expansions, polynomial evaluations, and iterative approximations can be implemented in a very efficient manner. These basic principles are also the basis of other applications and can be easily extended to include applications such as statistical analysis, digital filter design, and Fourier analysis. The common denominator in all these applications and numerical analysis techniques is the implementation of the sum of products and/or iterative structures. The series expansions, polynomial evaluations, and iterative approximations are basic numerical analysis techniques that can be used to evaluate and solve mathematical expressions and functions. Implementation of these techniques in digital signal processors greatly enhances the performance of numerically intensive analysis applications.+

The authors wish to acknowledge members of the DSP group at National Instruments—Warren Dixon, Mike Cerna, and Jibran Jahshan—for their input to this article and Sandy Garza for her help in getting this article published.

Eduardo Pérez received a BSEE in 1981, a MSEE in 1983, and a Ph.D. EE in 1989, all from the University of Texas at Austin. Mr. Pérez is a Design Engineer for National Instruments Corporation.

Dapang Chen received a BS from the University of Science and Technology of China in 1982, and an MS in biomedical engineering and a Ph.D. EE both from the University of Texas at Austin. Mr. Chen is a Design Engineer for National Instruments Corporation and leads the DSP software development group.

IRS

416 Very Useful
417 Moderately Useful
418 Not Useful

REFERENCES

1. M. Abramowitz and I.A. Stegun: *Handbook of Mathematical Functions.* Dover Publications, Inc.: New York, 1975.
2. R.L. Burden and J.D. Faires: *Numerical Analysis.* PWS Publishers: Boston. 1985.
3. W.H. Press, et al: *Numerical Recipes in C.* Cambridge University Press: Cambridge, 1988.
4. C. Phillips and B. Cornelius: *Computational Numerical Methods.* Ellis Horwood Limited: Chichester. England, 1986.
5. I. Miller and J.E. Freund: *Probability and Statistics for Engineers.* Prentice-Hall. Inc.: Englewood Cliffs, N.J., 1985.
6. L.B. Jackson: *Digital Filters and Signal Processing.* Kluwer Academic Publishers: Boston, 1986.
7. T.W. Parks and C.S. Burrus: *Digital Filter Design.* John Wiley & Sons, Inc.: New York, 1987.
8. A.V. Oppenheim and R.W. Schafer: *Discrete-Time Signal Processing.* Prentice Hall: Englewood Cliffs., New Jersey, 1989.
9. E.O. Brigham: The Fast Fourier *Transform and its Applications.* Prentice Hall: Englewood Cliffs, New Jersey. 1988.

DEPARTMENTS

page 74
Firmware Furnace

page 81
From the Bench

page 88
Silicon Update

MODULE: Prac
page 95
Practical Algorithms

page 104
ConnecTime

Toolmaker's Toolworks

We humans are tool users. While other animals may fiddle with sticks or rocks, we alone regard a muffler repair and oil change as normal weekend activities.

However, brute force rules the day in the engineering workplace, as many beasts (uh, engineers) continue to do their PC tasks manually. Their excuse is that it takes too much effort to find, write, or build the right tool for the job. While the PC itself is a tool, there are ways to simplify common tasks that don't require a lot of effort.

What prompted this column is a conversation 1 had with an engineer at an IEEE meeting. We were talking about bulletin board systems and I mentioned the "robot typist" scripts that automate my BBS message handling. He was intrigued that such a thing could be done.. .

Although I will concentrate on scripts for serial communication, you can apply similar tricks to nearly any repetitive task. In effect, you create a robot to do your bidding without having to bend any metal.

YOUR FIRST SCRIPT...

The 8052 is a good starting point as many of you have struggled with the BASIC-52 interpreter. Although BASIC-52 includes a rudimentary editor, retyping an entire line to replace one character gets old quickly. What you really want is to edit the program using your favorite editor, then download the whole smash. Show of hands: how many people do just that?

While it is easy enough to put the program into an ASCII text file, automatically sending that file to the 8052 is oddly difficult. BASIC-52 parses each line into an internal format after you press Enter, then displays the familiar ">" prompt when it is ready for the next line. The delay can range

*Many engineers **continue to** do their PC tasks manually. Their excuse is that it takes too much effort to find, **write,** or build the right tool for the job.*

from milliseconds to seconds, but, because there is no input buffer, you cannot send the next line until the interpreter is ready.

The obvious solution is to wait for the prompt character. Show of hands again: how many of you have fired up Procomm, changed the Pace Character to ASCII 62 (">"), sent the file using ASCII transfer, and found it didn't work? Thought so. Unfortunately, ASCII 62 can appear within the program line as the "greater than" operator, as part of the "not equal to" token, and in REM comments. Because BASIC-52 echoes each character as you type, Procomm (and most other terminal emulators) react to any ">" character within the line without waiting for the real prompt on the next line.

The next best solution is to delay for a fixed interval after each carriage

FIRMWARE FURNACE

Ed Nisley

return character, which has the predictable molasses-in-January effect. Worse, if the delay isn't quite long enough for just one line, there is a cascade of errors as BASIC-52 reacts to themush.Theerrorsscrolloff-screen quickly and may leave you with a damaged program that works almost correctly but gives you no indication of the missing parts.

Although the prompt character is not unique, it is preceded by a linefeed only at the start of a line. The trick to a successful BASIC download is writing a program to watch for the prompt sequence and send lines only when the interpreter is ready for them. Sounds easy enough, right?

Writing the "first program" seems to be the stumbling block for most engineers; once you've written one program, the next dozen or so come naturally. The trick is to get some early success quickly so you have enough confidence to continueonward. Rather than starting from scratch with C, the serial port hardware, and Campbell's "C Programmers Guide to Serial Communications," It Would Be Really Nice If you could avoid all the grunt work.

It turns out that your serial communicationsprogramcanhandlemost of the job already, so all you need is a tweak to handle this special case. Most heavy-duty shrink-wrapped programs now include a programming language; the language may be called a "script" or "macro" or some such, but the result is really a program.

Listing 1 shows a bare-bones Procomm Plus 2.0 script to download a BASIC-52 file. There are two major sections: display a prompt for the file name and open it, then read each line

```
proc main
string  FileSpec                          ; file name to download
integer FileEOF                           ; file at EOF
string  FileText                          ; text line from file

;--- fetch the file name & open it if possible

   clear
   box 8 8 12 70 7
   atsay 10 10 7 "File name:"
   atget 10 21 15 48 FileSpec

   strcat FileSpec ".BAS"
   isfile FileSpec
   if not success
       errormsq "Can't find file!"
       exit
   endif

   fopen 0 FileSpec "RT"
   if not success
      errormsg "Can't open file!"
      exit
   endif

;--- get to BASIC-52 command prompt

   clear
   call sendwait with "`r" "`r`n>" 1

;--- fetch file lines & ram 'em into the 8052

   FileEOF = 0
   set fqets_crlf off

   while not FileEOF
      fgets 0 FileText                    ; fetch the next line
      eof 0 FileEOF                       ; hit EOF yet?
      transmit FileText                   ; send data to 8052
      call sendwait with "`r" "`n>" 3 ; wait for prompt at end
   endwhile
   exit
endproc

;-------------------------------------------------------
; Send a string and wait for a specific response

proc sendwait
   strparm Cmd,Resp
   intparm WaitTime

   transmit Cmd
   waitfor Resp WaitTime
   if not waitfor
      SO = "Timeout after sending "
      strcat SO Cmd
      errormsg SO
      exit
   endif
endproc
```

listing 1 *-Procomm Plus 2.0 includes a revised and extended ASPECT script programming language which* **makes** *this example possible. Older versions are not compatible.*

from the file and send it to the interpreter. As usual, most of the code is in the setup and error handling sections! ***[Editor's Note:*** *Software for this article is available from the Circuit Cellar BBS and on Software On Disk #22. See page 106 for downloading and ordering information.]*

The core of the script comprises just four lines: read a line from the BASIC file, send it to the serial port, wait for the real prompt, and repeat until the file is finished. The rest is just window dressing that you add after you dope out the core.

Onceyougetit working, of course, there is an overwhelming urge to improve the situation. Listing 2 shows a revised (read "more complicated") inner loop that discards blank lines and checks for ascending line numbers (ugh!). The `showerror` procedure displays an error message and terminates; the full source codeisavailable on the BBS.

Now, that wasn't so bad, was it? A few dozen lines of code and you get a perfect BASIC-52 download every time, at very nearly the maximum speed the chip can handle. No more lengthy delays, no damaged programs, no muss, no fuss. *That's* why you use tools!

```
proc main

string  FileSpec                          ; file name to download
integer FileEOF                           ; file at EOF
string  FileText                          ; text line from file
long    LineNum                           ; current line number
long    LastLineNum                       ; previous line number

<<< file setup same as Listing 1 >>>

;--- fetch file lines & ram 'em into the 8052

  FileEOF = 0
  set fgets_crlf off
  LastLineNum = OL

  while not FileEOF

    fqets 0 FileText                      ; fetch the next line
    eof 0 FileEOF                         ; hit EOF yet?

    strlen FileText NO                    ; skip empty lines
    if zero NO
      loopwhile
    endif
    strpeek FileText 0 NO    ; or ones that start with a blank
    if eq ' ' NO
      loopwhile
    endif

    find FileText " " NO     ; extract what should be line number
    if not found
      call showerror with"*** Can't find line number in "FileText
    endif
    substr SO FileText 0 NO
    strupr so
    strcmp SO "REM"              ; is it "REM" or not?
    if success
      loopwhile                  ; skip REMs...
    endif
    atol SO LineNum              ; convert to number (0 if fails)
    if eq LineNum OL             ; skip invalid line numbers...
      loopwhile
    endif

    if le LineNum LastLineNum ; check for ascending line numbers
      ltoa LineNum S1
      call showerror with "*** Line sequence error at " S1
    endif

    LastLineNum = LineNum               ; remember this line
    transmit FileText                   ; send data to 8052
    call sendwait with "`r" "`n>" 3 ; wait for prompt at end
  endwhile
  exit
endproc
```

listing 2-Checking for sequential *line numbers with ProComm requires a little more code, but finds obvious goofs in the* BASIC-52 *program.*

THE REXX CONNECTION

As long as the program you use has a script language, you can create special-purpose tools. The catch (as you might expect) is that each and every program uses a different script language: talk about the Tower of Babel! Most of the languages are allegedly "similar" to C or Pascal or something, but they are always different enough that you must reach for the manual when you write a script. There is no standardization.

Well, almost no standardization. Heaving into view over the horizon is IBM's Grand Plan For The Future Of Computing known as SAA (Systems Application Architecture). Among other things, SAA includes CUA (Common User Access) which defines how programs should lookand feel. If you've seen Windows, OS/2 PM, or similarenvironments (love that term), you already know what SAA and CUA are all about: the implementations are not spot on CUA, but you get the idea.

The batch language IBM has been using on mainframes for years is now an official part of SAA and is shipping with OS/2 for PCs. Contrary to what you might think, REXXisnot a stodgy, bloated, half-baked attempt at a language...because it started as an unofficial, underground replacement for the existing mainframe batch language (which was stodgy, etc.). Eventually, The Powers That Be realized that nobody was using the official batch language any more, and the rest is history.

Oddly enough, IBM REXX does not include a standard CUA interface or an easy way to create one. IBM moves in fits and starts, but we've gotten a decent PC batch language at long last.

I can hear it now: "So who cares about OS/2 anyway?"

It turns out that Mansfield Software Group has had (for years!) an excellent REXX implementation for PCs running plain old DOS as well as OS/2. In fact, Mansfield's REXX is betterbyfar thanIBM'sOS/2REXX,if only because it includes most of the auxiliary functions you need to actually write a useful PC program.

Mansfield's KEDIT text editor includesa scaled-down version of REXX called KEXX (which sounds like a breakfast cereal, doesn't it?) as its macro language. Although KEXX is usable on its own, if you have REXX installed you can write macros in the full REXX language with all the bells and whistles.

Quercus Systems offers a serial communications program called REXXTERM which uses (no surprise!) REXX as its script language and provides a host of functions to simplify your serial scripts. More on this in a moment.. .

These products run under both DOS and OS/2, so the same scripts, macros, and batch files are usable on both operating systems. There are slight differences for some functions that reflect the different natures of DOS and OS/2, but, on the whole, portability problem are No Big Deal.

While REXX, KEDIT, and REXXTERM have not gathered the publicity of the full-page-ad class products you read about in PC Magazine, they are all solid, reliable programs that don't lack for power or features. Highly recommended.

SCRIPTS GONE WILD

Because REXX started out as a full programming language, it does not suffer from the limitations of most script or macro languages. To quote from "The REXX Language" by Mike Cowlishaw: "The primary design goal has been that [REXX] should be genuinely easy to use both by computer professionals and by 'casual' general users." Unlike most such claims, this one is true!

Because REXX was designed as a batch language, it includes a simple way to perform operating system commands: REXX evaluates an expression and, if the result is not a REXX language element, it goes to the host environment for execution. Both KEDIT and REXXTERM take advantage of this, so REXX provides a program framework for the script commands.

Listing 3 shows the file transfer loop from my REXXTERM BASIC-52

```
do while lines(fspec) \= 0
  pnum = pnum + 1

  ln = linein(fspec)

  if datatype(word(ln,1),'N')      /* skip unnumbered lines */
    then do
      parse upper var ln linenum keyword .
      if linenum <= lastline
        then do
          'emsg *** Sequence error at line ' linenum
          signal halt
        end
      lastline = linenum

      if keyword = ""          /* skip empty numbered lines */
        then iterate

      'send' ln || '\r'
      'matchx "\1>"  "ERROR:"  "SYNTAX:"'

      select
        when rc = 1 then do
          iterate
        end
        when rc > 1 then do
          'emsg *** Bad program!'
          signal halt
        end
        otherwise do
          'emsg *** Bad response from BASIC-52!'
          signal halt
        end
      end
    end
end
```

Listing 3—*This REXXTERM script ensures that BASIC-52 lines are in ascending order, skips unnumbered lines, and checks for unexpected responses from the interpreter. Because the transfer terminates when an error occurs, there is less chance of running an incorrect program.*

```
address dos 'globalv get curdisk curdir' /* get working dir */

dirspec  = curdisk|| curdir || '\*.bas'

fname. = 'none'
address dos 'listfile' dirspec '(sorta name ftype stem fname.

if  (fname.0  = 0)| \Datatype(fname.0,'N')
  then do
    'emsg No matches for ' dirspec
    exit 1
  end

address dos 'globalv get lastbas' /* get previous program */

fnwide = 1
item = 1
do fnum = 1 to fname.0
  fname.fnum = overlay('.',fname.fnum,9)
  fnwide = max(fnwide,length(fname.fnum))
  if lastbas = fname.fnum
    then do
      item = fnum                 /* mark previous file */
    end
end

fnum = fname.0

/* useful constants and suchlike */

esc = '1b'x
up = '0048'x
down = '0050'x
enter = '0d'x

wnorm = x2d(30)
whiqh = x2d(3F)

UnloadRXWIN = 0
SIGNAL ON HALT
if \fcnpkg('rxwindow')
then do
   address dos 'rxwindow /q'
   if fcnpkg('rxwindow')
   then do
      UnloadRXWIN = 1
   end
   else do
      emsq '*** Problem loading window package!'
      return
   end
end

/* popupthe selection menu and select a file name               */

parse upper value scrsize() with "rows ncols

wloc.ul.col = trunc((ncols/2)-(fnwide/2)-1,0)
wloc.ul.row = max(1,trunc((nrows/2)-(fnum/2)-1,0))

w = w_open(wloc.ul.row,wloc.ul.col,fnum+2,fnwide+2,wnorm)
call w_hide w,'N'
call w_border w

attrib. = wnorm
attrib.item = whiqh

do i = 1 to fnum
  call w_put w, i+1, 2, fname.i, fnwide, attrib.i
end
drop attrib

call w_unhide w

do forever
  key = inkey()
  select
    when key = up then do
      if item > 1
        then do
          call w_attr w,  itemtl,  2, fnwide, wnorm
          item = item - 1
```
(continued)

listing 4-Mansfield *REXX includes a global variable* **manager that** *stores values between program runs. This script section recalls the previous BASIC-52 file name and subdirectory, then creates a bouncing-bar menu with that file highlighted.*

file transfer script. In addition to the features of Listing 2, it skips unnumbered lines and verifies that the line didn't cause any errors. As near as I can make out, ASPECT doesn't support the functions needed for that level of error checking and the ASPECT code is much less readable.

When you are working on a BASIC-52 program, you tend to download the same file repeatedly. Mansfield REXX includes a "global variable" manager that can maintain variables between program runs; your program can save default settings, file names, and so forth during one session and recall them during the next without special contortions. Listing 4 shows how to fetch the previous file name and directory from the global variable manager.

Mansfield REXX also includes a straightforward screen window package. The script creates a bouncing-bar menu of all the *.BAS files and highlights the previous file name, so a single key press downloads that file.

The bottom line of all this is that a REXXTERM script handles everything from the point where I finish editing the file. I can invoke REXXTERM, shell to KEDIT to edit the file, return to REXXTERM and invoke the script. For repeated downloads the transfer requires five keystrokes and I'massured that it is done correctly every time.

Similar REXXTERM scripts automate file transfers to my EPROM emulator and EPROM programmer. In some cases the scripts read information back from the device to set or check operating conditions. For example, my EPROM programmer script verifies that the hex file fits within the EPROM's address limits before starting the programming process.

The REXX global variable manager comes in handy in other places, too. For example, Sage Software's Polymake controls my compilers and linkers, so it is easy enough to update the current directory **and** hex file name during each run. Under OS/2 each new command window automatically switches to that subdirectory.

Yes, all of these scripts work equally well under either DOS or OS/2. The advantages of OS/2 are that all the programs run without memory limitations and multitask quite nicely while you are doing something else. But you surely don't need OS/2 to get useful work done.

BBS ROBOTICS

Anyone who makes a long-distance call to a BBS knows that "think time" is best done off-line. Answering a question may take some research, but minutes are precious when the phone is off-hook. You should download all the messages, hang up and reply to them while the clock is stopped, then call again to upload your replies. Because the BBS is (presumably) automated, there is little reason for you to interact with it: let two robots do the talking.

The two most complex scripts I've written automate the mechanics of working with the Circuit Cellar BBS. A REXXTERM script handles dialing, downloads new messages, and uploads replies. A KEDIT macro parses the downloaded message file and re-

```
          call w_attr w, item+1, 2, fnwide, whigh
        end
    end
    when key = down then do
      if item < fnum
        then do
          call w_attr w, item+1, 2, fnwide, wnorm
          item = item + 1
          call w_attr w, item+1, 2, fnwide, whigh
        end
    end
    when key = enter then do
      leave
    end
    when key = esc then do
      leave
    end
    otherwise nop
  end

end

call w_close w

if key = esc
  then do
    'emsg *** Transfer cancelled'
    signal halt
  end

                                        /* remember for next time */
address dos 'globalv setlp lastbas' fname.item

fname.item = space(fname.item,0)           /* strip blanks */
fspec  =  curdisk||  curdir  ||'\'|| fname.item
'message File spec is' fspec

<<< continue with Listing 3 >>>
```

Listing 4—*continued*

duces replying to a message a matter of a single keystroke. All I have to do is type the text.. .that's not automated yet!

Lack of room prevents me from listing the script here, but the whole thing is available to download from the BBS. Imagine: the call you make to download the file may be the last manual call to the Circuit Cellar BBS you'll ever have to make.

WRAPPING UP

You can avoid writing tools by buying them from commercial suppliers. For example, the Basikit program available from MDL Labs handles BASIC-52 interfacing quite nicely (and with many more features than my scripts!), while TAPCIS automates dial-up access to CompuServe (if not the CCBBS). The key point is that you can get better and faster results by letting the computer handle repetitive chores and using your time for more productive tasks.

The BBS files for this column include the full source code for the scripts I've discussed here. The Procomm scripts require Procomm Plus 2.0 and the REXX scripts run under Mansfield Software's REXX with either REXX-TERM or KEDIT. Even if you have another con-u-n program and editor (quite likely.. .), the code should give you a startingpoint foryourown tools.

OK, I admit that muffler welding is more exciting. But tedious typing is much less interesting-and quite unnecessary. Write a tool today! ✣

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

SOURCES

Datastorm Technologies
P.O. Box 1471
Columbia, MO 65205
(314) 443-3282

Mansfield Software Group
P.O. Box 532
Storrs, CT 06268
(203) 4298402

Quercus Systems
P.O. Box 2157
Saratoga, CA 95070
(408) 257-3697

MDL Labs
15 Deerfield Rd.
Chappaqua, NY 10514
(914) 2380416

IRS

4 19 Very Useful
420 Moderately Useful
421 Not Useful

Reducing Power Consumption

Breathing New Life Into Data Logging

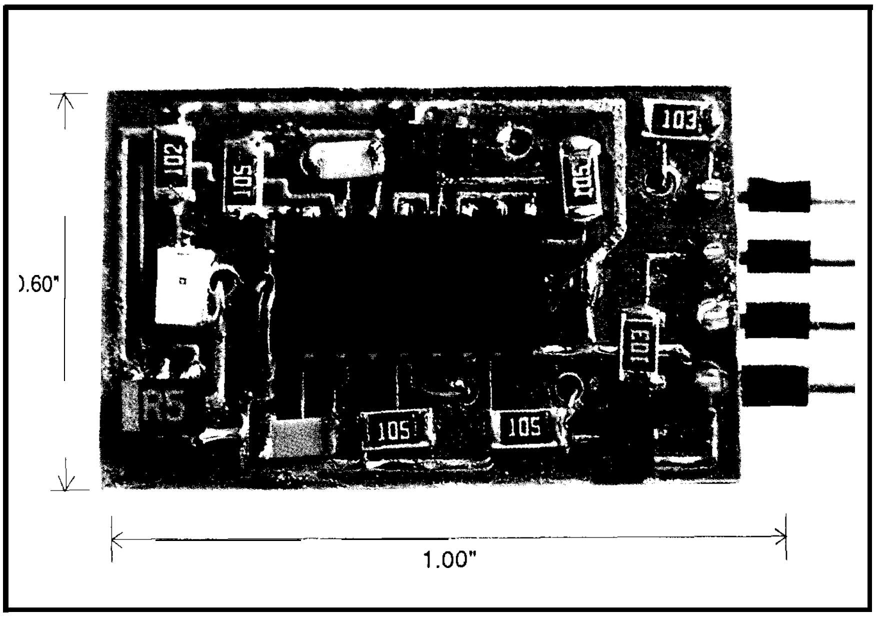
FROM THE BENCH

Jeff Bachiochi

"Recycle" and "conservation" are two of the 2bth century's latest buzz words. We all are becoming increasingly aware of our dwindling resources and growing waste problem. It kills me when an appliance is ditched because it's cheaper to buy a new one than to repair the old one. If you take your IBM back to the dealer, chances are the whole motherboard will be replaced. Few repair shops can or will do component-level troubleshooting and replacement.

It seems to me price should reflect not only the cost of raw parts and labor, but also the cost of disposal. An example of this is the "return deposit" on carbonated beverage containers. This generally means higher costs for goods sold. If we all have to pay more, it then makes financial sense to fix and not discard.

Our energy dependence on both foreign and domestic oil will continue until a viable alternative is discovered. TI's latest breakthrough in solar cell technology demonstrates the United States has the capability of leading the world in finding new and renewable sources of energy. Meanwhile, brownout alerts are predicted each summer when we crank up our air conditioners. Conservation is presently our only hope. So, we sweat a little, carpool when possible, quit watering the lawn, and recycle metal, glass, plastic, and paper.



Jhe *power control* module is small enough to sit on *fop* of a 9-V *battery clip.*

SOMETHING'S UP, BESIDES THE UTILITY BILL

My house is one of seven served by a "community well." The well's pump is located about 200 feet below ground and fills a 375-gallon storage tank with cold clean water. The energy cost for pumping the precious liquid averages about $300 a year ($25/month). That divides to about $50 a year per household. Not too shabby as long as repairs or maintenance are not needed.

Cindy, the designated treasurer and receiver/payer of the electric bill, stopped by the other day to chat. It seems the bills were increasing by leaps and bounds, topping over $60 for the last month. We had a mystery here. Either someone was drinking an awful lot of water, or the system had problems. Was the usage one of high demand during certain parts of the day or was it a constant increase. This is a job for "Data Logger"!

A quick inspection of the well house (pit is more appropriate) revealed a minimum of useful information. One, there were no leaks here and two, the only wiring accessible were the wires running down the well casing to the motor below. This means no power available, except when the pump is on. Hmm. I need to log pump usage with a battery-powered logger.

HOW LOW CAN YOU GO?

You may remember a project I presented a couple of years ago: the RTC52. This small, 80C52-based microcontroller used a built-in BASIC interpreter which really made development a snap. It actually took longer for me to find my prototype than it did to write the small BASIC program needed to log pump usage (see Listing 1).

Using a simple optoisolator with bipolar LEDs like the NEC 2505-1, I could monitor the AC power going to the pump. A series 47k resistor limits the LED current to a few milliamperes. A pull-up resistor on the open-collector output of the isolator will be pulled low whenever AC is present (which means the pump motor is running). The pull-up, along with a capacitor on the open-collector output, will filter out and prevent the AC zero crossings from accidentally being sampled-which would be interpreted as a "pump-off" indication-by smoothing the AC signal.

Let's see. A simple linear regulator on a 9-volt battery will give the controller the necessary 5 volts. A quick check with the current meter shows about 150 mA of current is necessary. Whoa! A 9-volt battery only has 550 mAH of useful life. Three hours of logging is useless.

THROW OUT ANYTHING NOT TIED DOWN, WE'RE SINKING FAST

Time to whittle down the load. About 50% of the supply current can be saved just by substituting HC parts where available and removing the unnecessary ones (MAX232 and the like). A 50% reduction isn't going to be enough. The CMOS version of the microcontroller has an additional feature over the standard NMOS version: PD and IDLE (power-down) modes. These modes conserve power by shutting down the internal oscillator. (Note: The Matra-Harris part has static registers and can be operated with an external clock down to 0 Hz. This is different than internally disabling the clock, which is what power-down does.) Once in the idle mode, only an interrupt or a complete reset can release it again. A reset is the only action which will revive the controller from the PD mode. A savings of about 15 mA is possible by using one of the power down modes. Since CMOS uses more power the more often it switches (the faster the clock speed), slowing down the clock will also save the same 15 mA. With all the power-saving devices employed, the consumption is still around 25 mA for the whole system. That's still less than one day on a 9-volt battery. Harumph.

Steve tackled some of these same issues in his article "Build a Low-Power Data Logger" in issue #15 of CIRCUIT CELLAR INK. If you haven't read it, shame! If you have, then you probably know that by shutting the power down completely, a great deal of current can be saved. Well, at least while you're waiting for your next sample.

TIMEBASE-POWER CONTROL-LOGGER

This data logging system is made from three parts. The "Timebase" which produces an accurate stream of trigger pulses equal to the rate at which you wish to sample your data. The second part, the "Power Control," turns on the power supply at each timed trigger pulse. Finally the "Logger," which samples and records the digital or analog data, and upon completion of its task, signals the Power Control to turn off the power until the next trigger pulse.

The RTC52 has the ability to run a program on power-up. In this case, it should sample the water pump's state and, if it has changed, record the time of the change, then signal the Power Control that it's done. See Figure 1.

```
10 REM PORT1.0 = LOGGED INPUT (0=ON 1=OFF)
20 REM PORT1.1 = POWER CONTROL OUTPUT (FALLING
    EDGE = TURN OFF)
30 REM PORT1.7 = PRINTOUT FLAG INPUT BIT (0 =
    PRINTOUT ROUTINE)
40 REM
50 REM UPON PROGRAM AUTOSTART, CHECK FOR
    PRINTOUT FLAG
60 TP=TP+1 : GOT0 240
70 REM
80 REM IF NOT, LET'S SEE IF FIRST TIME AROUND,
    CHECK LOGGING ID
90 IF (XBY(MTOP+1)=55H.AND.XBY(MTOP+2)=0AAH)
    THEN GOT0 370
100 REM
110 REM FIRST TIME AROUND, LET'S SET UP MEMORY
120 REM WRITE LOGGING ID
130 XBY(MTOP+1)=55H : XBY(MTOP+2)=0AAH
140 REM
150 REM TWO BYTES FOR STATUS STUFF
160 REM (PRESENTLY USING ONLY LS BIT TO HOLD
     LAST ON/OFF STATE)
170 XBY(MTOP+7)=0 : XBY(MTOP+8)=0
180 REM
190 REM SET TP AS POINTER FOR NEXT LOGGING
     TABLE ENTRY
200 REM SAVE TP AT MTOP+3&4
210 TP=MTOP+9 : O=3 : GOSUB 320
220 REM
230 REM TEST FOR END OF RAM
240 XBY(TP)=0 : IF XBY(TP)<>0 THEN GOT0 280
250 XBY(TP)=0FFH : IF XBY(TP)<>0FFH THEN 280
260 REM
270 REM NOW PAST TOP OF RAM, SAVE TP-1 AT
     MTOP+5&6
280 TP-1 : O=5 : GOSUB 320 : GOT0 370
290 REM
300 REM SAVE ADDRESS ROUTINE
310 REM USES '0' AS AN OFFSET TO MTOP, SETS
     PRESENT VALUE OF TP
320 XBY(MTOP+O)=INT(TP/256)
330 XBY(MTOP+O+1)=TP-(INT(TP/256)*256) : RETURN
340 REM
350 REM LOGGING ROUTINE
360 REM FIRST GET OLD (LAST) STATE OF INPUT
370 OS=(XBMTOP+7).AND.01H)
380 REM
390 REM SECOND, READ INPUT BIT FOR NEW STATE
400 NS=(PORT1.AND.01H)
410 REM
420 REM IF THE SAME, THEN LET'S GET OUT OF HERE
430 IF OS=NS THEN 730
440 REM
450 REM IF DIFFERENT SAVE THE NEW STATE
460 XBY(MTOP+7)=NS
470 REM
480 REM GET LOGGING TABLE'S NEXT ENTRY AND END
     OF RAM POINTERS
490 TP=XBY(MTOP+3)*256+XBY(MTOP+4)
500 TE=XBY(MTOP+5)*256+XBY(MTOP+6)
510 REM
520 REM READ THE REAL-TIME CLOCK REGISTERS FOR
     DAY/HR/MIN/SEC
530 FOR X=7 TO 0 STEP -1
540 XBY(TP)=XBY(0E030H+X)
550 TP=TP+1
560 REM
570 REM WHILE LOGGING INFO CHECK FOR END OF RAM
580 IF TP=TE THEN GOT0 730
590 NEXT X
600 REM
610 REM NOW SAVE THE CURRENT INPUT STATE
620 XBY(TP)=NS
630 TP=TP+1
640 REM
650 REM CHECK AGAIN FOR END OF RAM
660 IF TP=TE THEN GOT0 730
670 REM
680 REM IF NOT AT THE END OF RAM YET,
690 REM SAVE THE LOGGING TABLE'S NEW NEXT ENTRY
     POINTER
700 O=3 : GOSUB 320
710 REM
720 REM NOW SHUT THIS SYSTEM DOWN
730 PORT1=PORT1.AND.0FDH
```

(continued)

listing 1 *-Compiled BASIC was fast* **enough to** *handle the deed*

```
740 STOP  : REM SHOULDN'T EVER GET HERE
750 REM
760 REM PRINTOUT ROUTINE
770 REM GET FIRST ENTRY AND LAST ENTRY POINTERS
780 TS=MTOP+9
790 TP=XBY(MTOP+3)*256+XBY(MTOP+4)
800 REM
810 REM LOOP AND PRINT OUT ALL 9 BYTE ENTRIES
820 FOR X=TS TO TP STEP 9
830 PRINT XBY(X)*10 + XBY(X+1),"-",XBY(X+2)*10
     t XBY(X+3),":",
840 PRINT XBY(X+4)*10 t XBY(X+5),":",XBX+6)*10
     +XBY(X+7),"-",
850  IF XBY(X+8)=0 THEN PRINT "ON" ELSE PRINT
     "OFF"
860 NEXT X
870 REM
880 REM EXIT
890 GOT0 730
```

isting 1 *-continued*

The RTCIO expansion board has plenty of TTL and 8-bit analog I/O, but I don't need any of that here, so I'll depopulate it. The real-time clock/calendar will serve as my Timebase. Not only will this supply time and date information to the microcontroller for logging, but it will serve as a programmable timebase. The programmability is somewhat limited. However, the four rates defined will cover almost any need: 1/64-second, l-second, l-minute, and l-hour intervals.

The last section, the "Power Control," uses a new device manufactured by Toko: a combination linear regulator with a digital on/off switch. Add to this a CD4538 one-shot, used as a flip-flop with edge-triggered set and reset inputs, and a couple of transistors for level shifting, and you've got a neat little triggerable power controller. See Figure 2.

AND NOW FOR THE CHARTS AND GRAPHS

Figure 3 is a look at the number of sample times available to us and how they accumulate over time. How does this relate to current consumption? My data logging program written in BASIC takes about 400 ms to execute. I'll use a 50-mA current draw as an example. The 9-volt battery is capable of 550 mAH. If the logger is powered continuously, the system will run for:

$$\frac{550 \text{ mAH}}{50 \text{ mA}} = 11 \text{ hours}$$
$$= 39{,}600 \text{ seconds}$$

If a sample takes 1 second, then that's about 40,000 samples. Since my program takes 0.4 seconds to take a sample, we'll get:

$$\frac{39{,}600 \text{ sec}}{0.4 \text{ sec/sample}} = 99{,}000 \text{ samples}$$

Now we're getting somewhere.

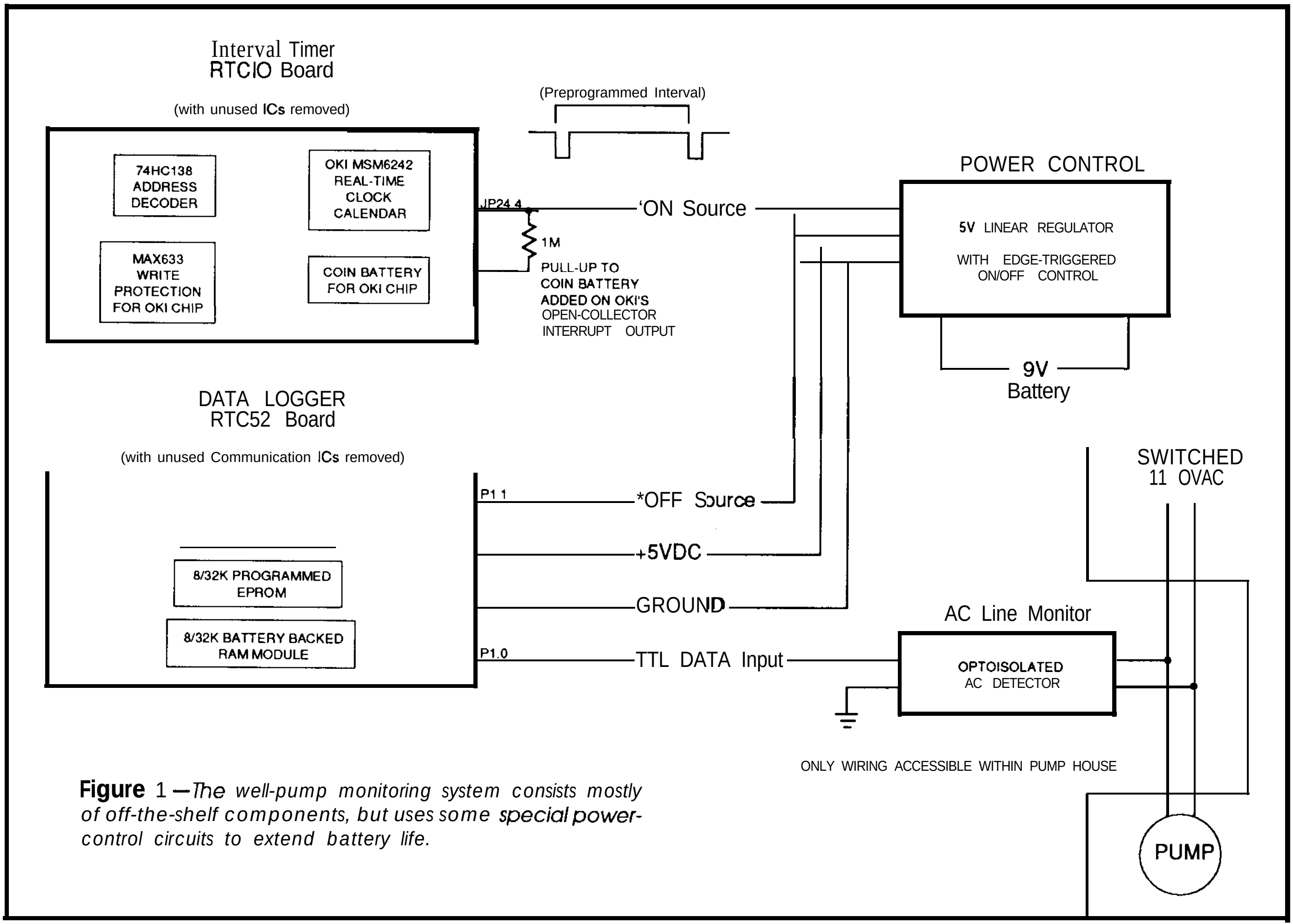


Figure 1 *—The well-pump monitoring system consists mostly of off-the-shelf components, but uses some special power-control circuits to extend battery life.*

HOW ABOUT OTHER LANGUAGES?

Indeed, this BASIC interpreter is not the fastest thing on wheels, but it is great for solving most of my problems. I ran the Systronix BCI51 Integer BASIC Compiler on this program and got some interesting results. The program grew up to about 4K of code and ran about eight times faster! This means eight times as many samples are possible, around three-quarters of a million samples. I didn't have the time to write and debug the routine totally in assembler. I would have to use assembler, though, if I wanted to sample at the highest rate of 1/64 second. At that rate, sample-to-sample time is only about 16 ms, which is faster than even the compiled code would run. I estimate an assembler routine would get bored just hanging around between samples.

Compare these numbers to the chart and you will see we are in the range of a week's worth of samples every second or a year's worth of samples every minute. And that's without having to write assembler. After all, I'm interested in taking samples here, not developing code!

TOTAL CONTROL FOR NEXT TO NOTHING

The "Power Control" circuitry does require some power 100% of the time. The Toko device has a quiescent current of only 10 pA, however the CD4538 and level shifters draw about 10 μA. This is 1.5 mAH per week; comparable to typical losses encountered while the battery sits on the shelf unloaded.

The standard 9-volt battery does not have a high output current, so current conservation is still an important factor when using the 9-volt cell as your source. Kodak makes both a standard 550-mAH and a high-energy 1.1-AH 9-volt battery. Not only does the high-energy cell have a higher output, but it has better shelf life characteristics.

STORAGE SYSTEM

There are few ways to store information. The first and simplest is to store the information as a synchronous stream; One sample for each bit time. The number of samples times the number of bits per sample will equal the amount of space necessary to save the data.

A second approach is to save only the changes in information. With analog data this means the difference between the last sample and the new sample. With this approach, we assume the bit resolution being stored each sample time is a maximum change per sample time and is less than the absolute maximum resolution, which would require more bits of storage per sample time. With digital data, such as I am logging here, I only need to indicate that the information has changed and how long has it been since the last change. This will save storage space, since I know it will not change frequently. The "on" portion of a pump cycle should last about ten minutes. The "off"

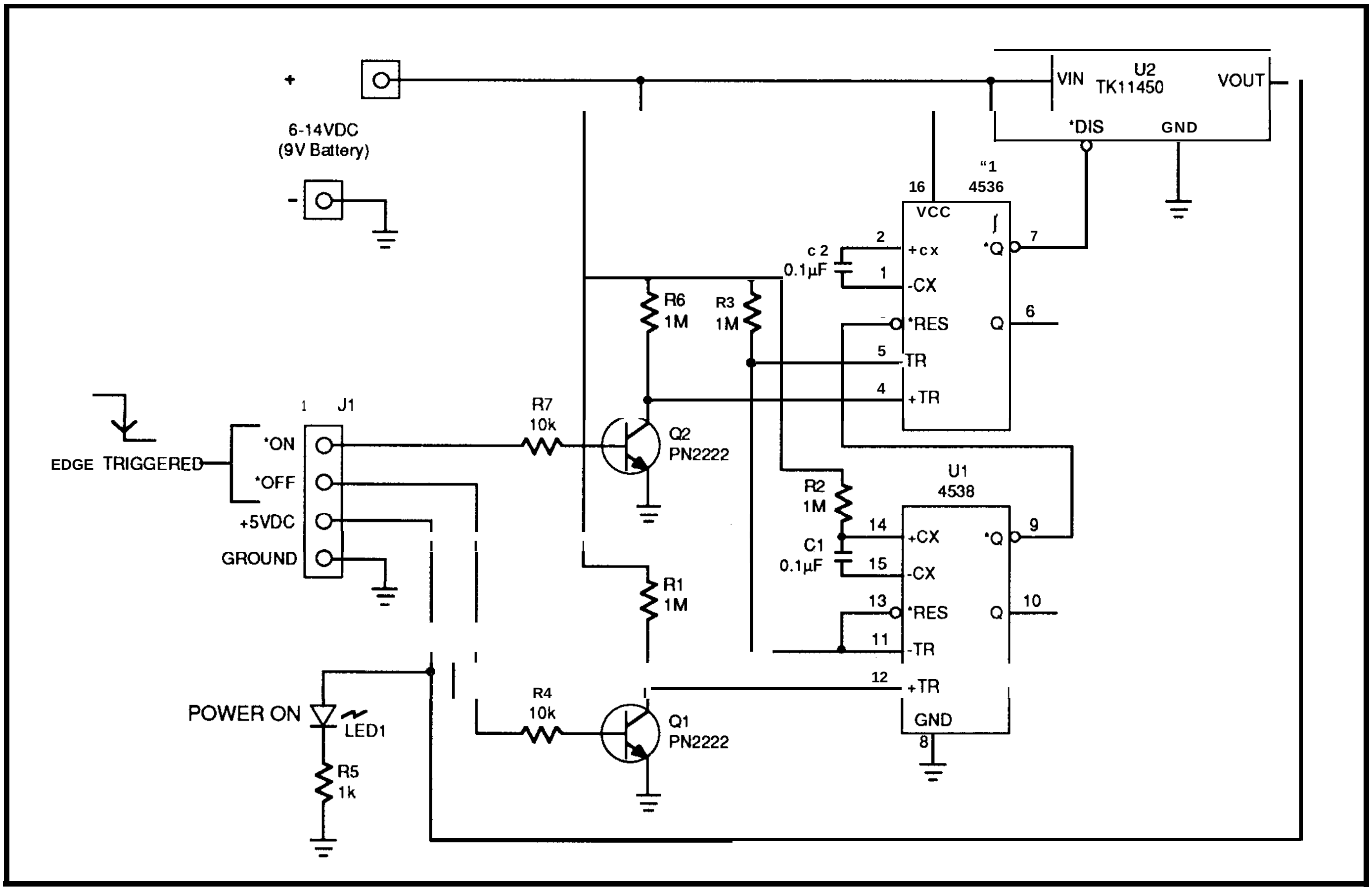


Figure 2—*The* **power** *control section uses a new device from Joko* **which** *is a combination linear regulator and digital on/off switch.*

		Sample Rate			
		1/64 second	1 second	1 minute	1 hour
Time Period	second	64	1	–	–
	minute	3,040	60	1	–
	hour	230,400	3,600	60	1
	day	5,529,600	86,400	1,440	24
	week	38,707,200	604,800	10,080	168
	year	2,002,774,400	31,449,600	524,160	8,736

Figure J-Each *column shows the total number of samples at the given sample rate that would be collected over the time period shown at the left.*

portion of the cycle will depend on the overall rate of water usage.

Nine bytes of storage are used for each power transition detected. Setting aside 30K of RAM for logging will give me room for about 3000 transitions, or ten times what I predict I'll need for a week's worth of logging. Using a timebase of once per second will give plenty of resolution.

Prior to actually hooking this up to the pump's wiring, I must attend to a few more details. Clearing the NVRAM will make sure no meaningless data is present. Checking the clock for correct date and time, and setting the timebase for the necessary once-per-second pulse rate.

ON YOUR MARK, GET SET, LOG IT!

While the logger is collecting data, I am going to delve into the world of surface-mount components. Since the Toko device was only available in SOP (small-outline package), and most of the other parts are available in surface mount as well, I think this circuit would be handy if it was mounted on the 9-volt battery clip.

It is difficult to realize the true physical sizes associated with surface-mount parts when using CAD layout software. On the screen you can zoom in until the parts are large and easy to work with. Once you plot the artwork, however, reality hits hard. This is truly equivalent to brain surgery. There are special techniques which must be used when assembling a surface-mount board, but I'll save that for a later column.

WHAT'S THE DIAGNOSIS?

The data's back. Let's get a dump of the data to the printer (a sample is shown in Figure 4). A typical "on" time seems to be about 20 minutes (twice what I expected, sounds like potential pump problems). The "off" times are shorter (this means more usage). The cycling continues all night (oh-oh, most likely a leak somewhere in the system).

```
10 - 12 : 16 : 1 -ON
10 - 12 : 37 : 23 -OFF    (ON FOR 21 : 16)
10 - 13 : 17 : 10 -ON     (OFF FOR 39 : 47)
10 - 13 : 38 : 50 -OFF    (ON FOR 21 : 40)
10 - 14 : 22 : 13 -ON     (OFF FOR 43 : 23)
10 - 14 : 44 : 14 -OFF    (ON FOR 22 : 1)
10 - 15 : 30 : 44 -ON     (OFF FOR 46 : 30)
10 - 15 : 53 : 13 -OFF    (ON FOR 22 : 29)
```

Figure 4-A *partial line printer output with comments.*

The first part of the problem, as I saw it, we could handle. The submersible pump was hoisted out of the well. Sure enough, iron deposits had all but choked the pump impellers. A total dismantling and thorough cleaning was necessary to bring the pump back up to snuff.

Finding a potential leak in the system was another matter. With all the shut-offs closed (one at each house), the water tank was still continuously filling and draining, (At least at this point the filling cycle was down to under ten minutes.) Shutting off the pump's inlet value to the tank ensured no water was draining back into the well. (Too bad, fixing the check valve would have been an easy task.) The leak was somewhere in the buried water main. Since no one in the neighborhood owns excavation equipment, we gave in and called the plumber. One and a half days later a crack was found in a joint coupling. If only I had built that moisture content detector, why I'll bet I could have.. . ✣

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

IRS

422 Very Useful
423 Moderately Useful
424 Not Useful

SILICON UPDATE

Tom Cantrell

Kynar To The Rescue

The Ultimate Sensor?

Kynar isn't silicon, but it is still a material of which every chip jockey should be aware. After all, our wondrous computers ultimately have to connect with the real world to do something useful.

The secret of Kynar is the piezoelectronic effect, first discovered in quartz over 100 years ago. In essence the piezo material itself, like a motor/generator, is able to transform electricity into work and vice-versa. Later, certain ceramics were found to exhibit the piezo (Greek for "pressure") effect.

Originally Kynar, manufactured by Pennwalt Corp., was a just another polyvinylidene fluoride (a.k.a. PVDF) semicrystalline resin with main features of toughness and flexibility for use in conduit, pipes, wire jackets, and so on. Yawn. However, in the early '70s, piezo researchers searching for ever better materials discovered that a variety of organic materials (including human bone) exhibit the effect. Oh, and by the way, once exposed to a strongelectric field at elevated temperature-a process known as "poling''-to permanently align the molecules, PVDF works far better than any other material.

Talk about a market opportunity dropping in your lap! Today, backed by a new owner and a dedicated marketing and sales arm, Kynar piezo film serves the glamorous world of high-tech electronics; a far cry from its humble "plastic pipe" beginnings.

The rest of the article will discuss the key properties of Kynar piezo film summed up in Figure 1. Ha ha, just kidding. I have neither the time nor the ability (what the heck is "Young's Modulus" anyway?). I'd just rather think of it in simple terms: you do something to the stuff and voltage comes out, you feed it voltage and it does something.

So instead of getting in over my head (more than I already am), I'll just try to sum up kind of how Kynar works and, more importantly, show you that even an amateur can get it hooked to a computer and do some pretty neat things.

Instead of getting in over my head, I'll just try to sum up kind of how Kynar works and, more importantly, show you that even an amateur can get it hooked to a computer and do some pretty neat things.

Fortunately, this rather exotic technology is readily available to all in the form of some low-cost evaluation kits. The "Basic Design Kit" at only $50 contains all you need to demonstrate a variety of applications. It also includes a 90-page technical manual that is quite good, though I must confess I only skimmed the portions similar to what is shown in Figure 1. For the more committed, the manual includes all the details of Kynar's physical and electrical properties and example computations.

As mentioned, a piezo material like Kynar can perform work (i.e., move) in response to an applied voltage. The material reacts in an AC manner with opposite polarity producing opposite motion. One obviousapplication is as a speaker. It is quite novel to listen to music emitting from what looks like a small sheet of tinfoil . As expected, such a tiny mass doesn't yield much low-end punch (Figure 2), but on the other hand, can generate frequencies far higher than audible (question for stereo buffs: do they still use piezo for tweeters?). Anyway, the point is that piezo film works fine for chimes, alarms, and tonegenerators with advantages of light weight, low power, and reliability.

Another application along these lines is the so-called solid-state fan. A number of Kynar filmstrips are laminated together making a piezo paddle which is driven back and forth. The main advantages compared to a motor-driven fan are the small size and efficient, low-power operation.

You may have seen the various novelty insect replicas whose transparent wings flap without apparent mechanical input. Yep, piezo film strikes again.

The list of interesting "work output" applications of piezo goes on: active vibration damping/noise canceling, ink-jet printer pump, deformable mirrors, optical shutters, and so forth. However, other than use in a "singing mode"

Typical Properties of Piezo Film

Property	Symbols	Values	Units	Conditions
Thickness	t	9, 16, 28, 52	μm	
		110, 220.800		
Piezo Strain Constraint	d_{31}	$23x10^{-12}$	(m/m)/(V/m)	
	d_{32}	$3x10^{-12}$	or	laterally clamped
	d_{33}	$-33x10^{-12}$		
	d_t	$-22x10^{-12}$	(C/m²)/(N/m*)	
		0.16	(C/m²)/(m/m²)	
	e_{33}		or	laterally clamped
			(N/m²)/(V/m)	
Piezo Stress Constant	g_{31}	$216x10^{-3}$	(V/m)/(N/m²)	
	g_{32}	$19x10^{-3}$		
	g_{33}	$-339x10^{-3}$	or	laterally clamped
	g_t	$-207x10^{-3}$	(m/m)/(C/m²)	
Electromechanical	k_{31}	12	%	@100 Hz (Vf,)
Coupling Constant	k_t	29	%	@100 MHz (VF_2/Vf_3)
Permittivity	ε	$106x10^{-12}$	F/m	@10 kHz
Relative Permittivity	$\varepsilon/\varepsilon_o$	12		@10 kHz
Capacitance	C	$379x10^{-12}$	F/cm²	28 μm Film @10KHz
Acoustic Impedance	Z_a	$3.9x10^6$	kg/m²-sec.	3 Direction
	"	$2.7x10^6$	"	1 Direction
Electrical Impedance	Z_e	1350	ohms	100cm² for 9μm film
				@ 1 kHz
Speed of Sound	v_s	2.2x103	m/sec.	3 Direction
	vs	$1.5x10^3$	"	1 Direction
Pyroelectric Coefficient	P	$-30x10^{-6}$	C/(m²K)	@ 20°C
Volume Resistivity	ρ_V	$1.5x10^{13}$	ohm-m	@ 20°C
Surface Resistivity	$R_\square$	<2.0	ohms/square	Aluminum
of Electrodes		co.5	" "	Silver
Dissipation Factor	tan-δ_e	0.015 0.02		@ 10 kHz
Mechanical Loss Tangent	tan-δ_m	0.10		
Dielectric Strength	E_B	75	V/μm	
Max. Operating Field	E_O	10	V/μm	@ DC
		30	V/μm	@AC
Density	ρ	$1.78x10^3$	kg/m³	
Water Absorption		0.02	%	By Weight
Tensile Strength	T_B	$140\text{-}210x10^6$	N/m²	1 Direction
at Break	T_B	$30\text{-}55x\ 10^6$	N/m²	2 Direction
Elongation at Break	S_B	2.5-4.0	%	1 Direction
	S_B	380-430	%	2 Direction
Elongation at Yield	S_Y	2-5	%	
Young's Modulus	Y	$2x10^9$	N/m²	

Figure 1—*It isn't necessary to understand all the characteristics of piezo film in order to use it effectively.*

(more later), thisarticle will focuson piezo film application as a sensor input, a role at which it is amazingly versatile.

I SENSE, THEREFORE I AM

Whatever the situation, the basic Kynar Piezo Film sensor is the same: a thin sheet (actual size and thickness depends on the application) of Kynar with a film of metal (various types, depending on application) deposited on each side to serve as the electrical connection. A typical sensor element (part number DT1-028K) is shown in Figure 3. The small strip (40× 15 mm) is light and flexible. Kind of like a folded-on-itself piece of Scotch tape (though stronger) and the price is right (50 cents). Just connect a wire to the metal film on each side of the Kynar strip and you're in business.

As shown in Figure 4, the most obvious application of piezo film is as a switch. "Pushing" the switch generates an AC spike corresponding to the down/up stroke. The basic advantages over electromechanical switches include reliability (no contacts or moving parts), no need for power, and RF "silence" since, unlike a regular keypad, no scanning is required.

The basic switch of Figure 4 does have operational weaknesses that can be overcome by a mixture of mechanical and electrical countermeasures.

First, the output isn't clean in the sense that the voltage generated for each keypress may vary widely in amplitude, frequency, and duration. This might be helpful in some cases, but often it isn't. Mechanical improvements involve physically configuring the film with a "crease," "dimple," or some other structure that yields a "snap" action which both boosts the output and makes it more repeatable.

A basic problem is the dynamic nature of the piezo effect. Output is only induced for changes in stress, not static stress levels. If you think it can be otherwise, I've got a perpetual motion machine you may be interested in buying, and I'll throw in the Brooklyn Bridge for free.

The bottom line is while a piezo switch press can be detected, it is not possible to interrogate the switch's instantaneous state (open or closed) directly. **Here** is where

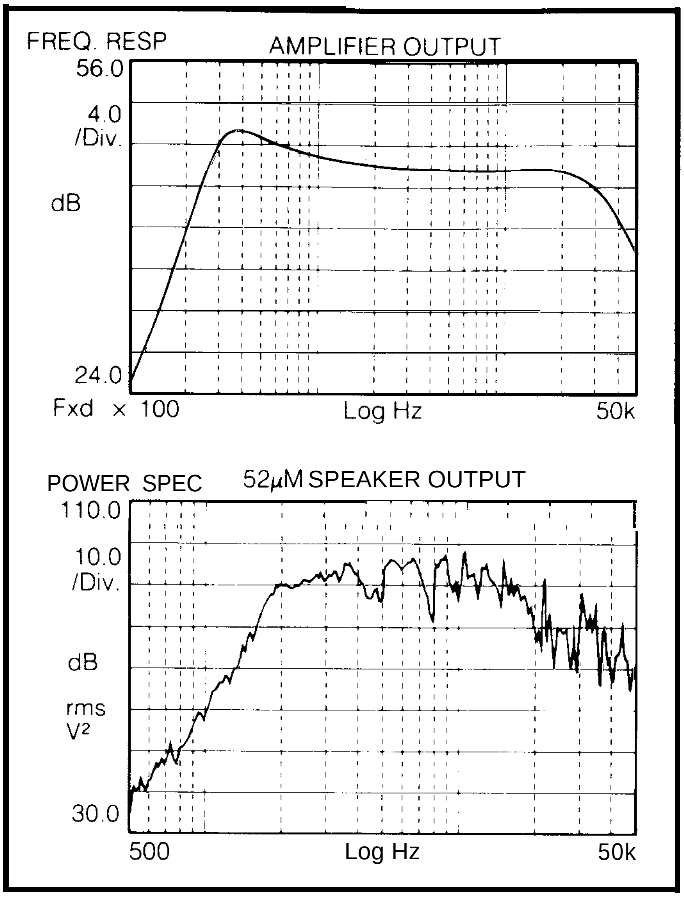


Figure 2-One novel use of Kynar is us a flat speaker

the "singing" idea (Figure 5) comes into play. This scheme utilizes both sides of the piezo effect (energy -> work, work -> energy) to make the switch appear static rather than dynamic. Two piezo films are bonded to opposite sides of a substrate. The output piezo ("speaker") is driven with a frequency corresponding to, and inducing, the natural resonance of the substrate. This mechanical vibration generates a corresponding charge in the input piezo ("microphone"). The key point is that any physical contact with the apparatus will change or suppress the output frequency. The host interrogates the switch by "listening" to whether the switch is "singing" (open) or "gagged" (closed).

The real benefits of singing are found in more esoteric static variable sensing applications such as measuring pressure, load, or fluid level. By refining the analysis of the received "song," these sensors can measure static variables with a surprising degree of range and accuracy. For example, piezo-based scales offer range from grams to hundreds of pounds with 0.1% accuracy.

SHAKE IT UP BABY

The high frequency response and good physical characteristics (light, flexible, and tough) of piezo film make it ideal for impact and vibration sensing in everything from rock-'em/sock-'em toys to self-diagnosing machines.

Basic impact sensors are a snap. Just adhere a piece of film somewhere on the impacted part. The toughness and simplicity of this approach works well for counting applications of many types, especially those with higher impact or harsher environment than tolerable by a mechanical switch.

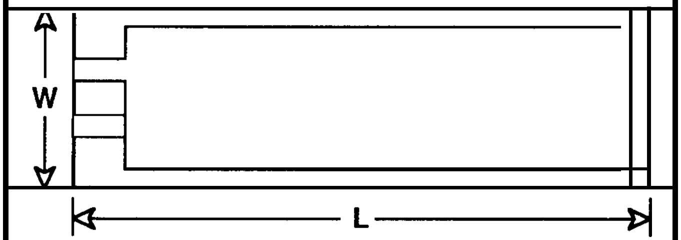
The flexible nature of Kynar can also be put to advantage in cuff-type sensors often used in medical gear to measure blood pressure or respiration rate. The same idea can be used to solve tough problems like measuring the output of a fuel injector: a piece of film taped around the injector line detects the vibration of each fuel pulse.

High frequency and precision vibration sensing is the true forte of piezo. For instance, machines with bearings will often exhibit minute, but discernible, changes in vibration as the bearings begin to fail. Rather than compromising with costly inspection or routine maintenance, the ultimate solution is to attach some piezo film to the machine and hook it up to a micro/DSP driving a "Fix me or else" LED!

Another interesting application is the CR-M01 (a.k.a. "Vibrasense," $25), a small module that contains a piezo film sensor and weighted lever arrangement. The shape of the sensor and lever arrangement can be tuned to adjust sensitivity in each axis. The built-in circuit includes a sensitivity adjustment (resolution in hundredths of a g) and latching LED. One use for Vibrasense is to ship the module along with other fragile goods. Then, if the "goods" are turned into "bads" in transit, a few hours under the glaring LED will encourage the fumble-fingered to confess.

As if the piezo effect isn't enough, it turns out the Kynar also exhibits excellent pyroelectric response, particularly in the 7-10-μm infrared range which just happens to correspond with the IR radiated by humans. Thus, piezo film is now finding use in the motion sensor and security markets.

DT Patterns
Thick (Ag Ink) Metallization



	Dimension (L x W):	Piezo Film Thickness:
DT1-028K	40mm x 15mm	28μm
DT1-052K	40mm x 15mm	52μm
DT2-028K	70mm x 15mm	28μm
DT2-052K	70mm x 15mm	52μm
DT3-028K	85mm x 22mm	28μm
DT3-052K	85mm x 22mm	52μm
DT4-028K	165mm x 22mm	28μm
DT4-052K	165mm x 22mm	52μm

Figure 3-A typical Kynar Piezo Film sensor consists of a thin sheet of Kynar with a film of metal deposited on each side.

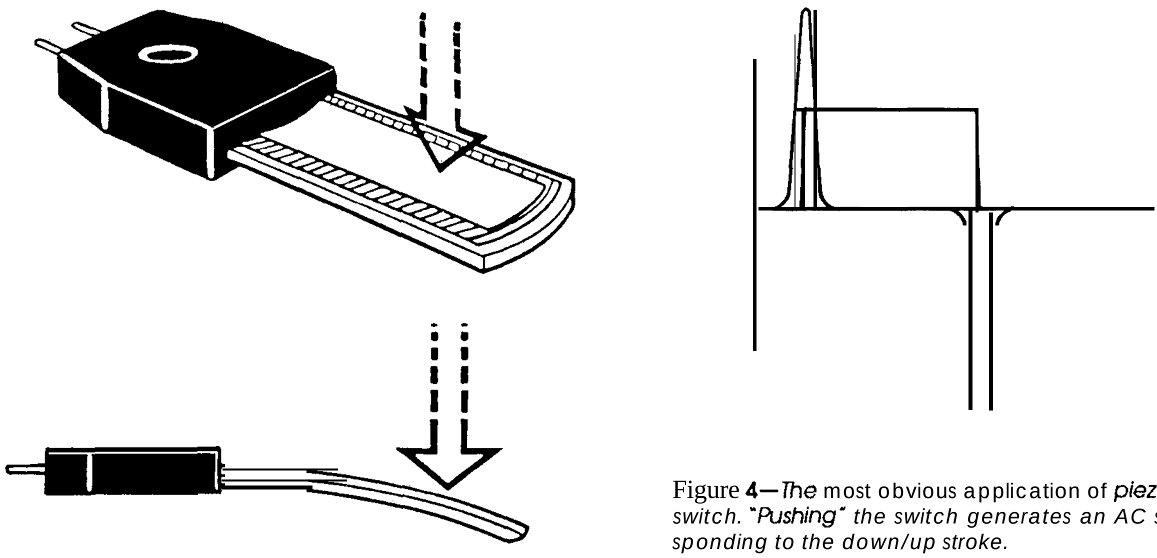


Figure 4—*The most obvious application of piezo film is as a switch. "Pushing" the switch generates an AC spike corresponding to the down/up stroke.*

Advantages of the piezo motion sensor include full 180-degree field of view, small size and weight, and very low power consumption. For instance, the PIR180 motion sensor has a 20' range which is fairly competitive with typical security light ultrasonic detectors. However, the piezo-based unit is tiny--only about an inch on a side and weighing less than an ounce-and consumes very little power (2 mA @ 5 V). Standard versions feature in-module logic to condition and process the signal, producing an easy-to-interface TTL-level "motion detected" output.

INTERFACE-A PIEZO CAKE

Well, enough of this theory, it's time to fiddle with some of this Kynar stuff and see what happens. Armed with various Design Kits, an oscilloscope, and a BASIC-

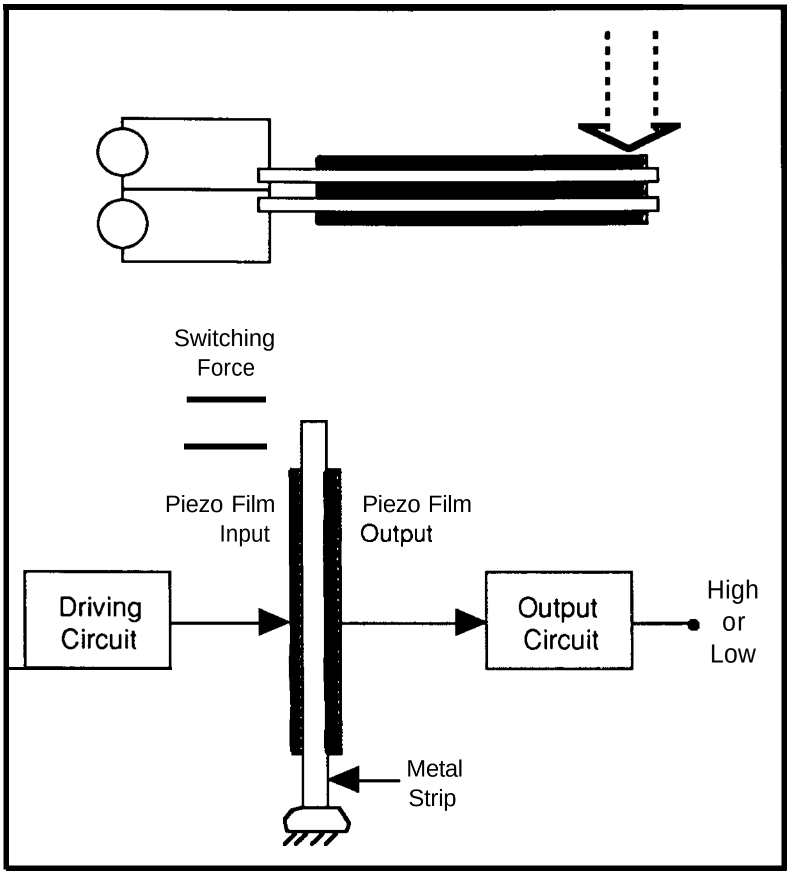


Figure 5-*A modification to the simple switch uses one piece of film as a* "speaker" *and a second piece as a 'microphone.' When the* button is *pressed, the speaker is muted and nothing is picked up by the microphone.*

based single-board computer with 8-bit ADC, I plunged forward.

First, I built a piezo speaker setup which mainly involves assembling a small IC-based preamp. The schematic is included and, an extremely nice touch, the Radio Shack part number for each component is noted. After a refreshingly quick visit to the 'Shack, the 3"×6" foil "speaker' was soon blaring music at which point interesting effects associated with various bending/ mounting schemes could be heard (I'm told gluing it to a balloon works well).

Now, it was time to actually connect a sensor to the computer and see what happens. Figure 6 shows that the simplified electrical equivalent of piezo film is a voltage generator whose output depends on the induced charge (Q) and film capacitance (C_f). Hmm.., didn't I see something in one of these data sheets about hundreds of volts...? Indeed, under terminal stress, a small (12 x 30 mm) foil can generate 830-1275 volts!

Basic Design Kit items at hand included eight low-cost sensors (DT1-028K), one shielded version (SDT1-028K), and a sensor (DT2-028K) bonded to a credit card. I choose the latter and, figuring a little care was in order before connecting hundreds of volts to my computer, hooked it up to an oscilloscope. Now, I'm no scope jockey, so I spent a lot of time fiddling with the credit card and the scope dials before finally convincing myself it was safe to proceed.

Next, I connected the credit card to the controller's ADC which was configured for -5V to 5V operation (remember the sensor puts out AC!). Adding a few lines of BASIC to sample and threshold check the ADC was easy and shortly thereafter, the controller would exclaim "OUCH" as I pounded on the credit card.

But, for analog reasons undiscernible to me, the sensitivity was poor, that is, I had to really pound the card and the generated signal was only a few percent of the ADC full-scale. Guess I need an amplifier.

Though the manual shows some generic op-amp interface circuits, I simply used the little audio amp from the speaker experiment which worked just fine for fiddling around. Soon, I had thecontroller quantifyingitspain with

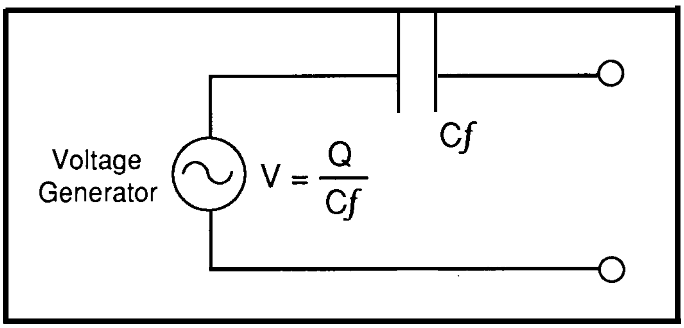


Figure 6—*The simplified electrical equivalent of piezo film is a voltage generator whose output depends on the inducedcharge (Q) and film capacitance (C_f).*

a full 8 bits of accuracy: "Ouch, that REALLY hurt."

Next, I hacked my own version of the singing switch by taping one of the DT1-028K to the piezo speaker. Then, by driving the speaker (classical music doesn't work-too many quiet passages-try rock and roll) and monitoring the driven sensor with a few lines of BASIC I easily implemented a static touch switch.

How about the pyroelectric effect? I positioned a table lamp near the DT1-028K sensor and discovered that turning the light on and off seemed to produce a meaningful result.

But wait a minute. If I turn on the light, the ADC should show activity as the film heats up. But, when the film stabilizes at the higher temperature, the output should level off, right? So, how come the output isn't leveling off, no matter how long the light is left on?

Well, there are other ways to test pyro response, I thought, reaching for some matches. As events quickly unfolded, I learned at least three things. First, it's hard to hold burning matches in close proximity to delicate items whilesimultaneouslyfiddling with a scope. Second, Kynar won't burst into flame, but it does suffer horribly when torched. Finally, and most importantly, I learned a valuable lesson about piezo film sensor design.

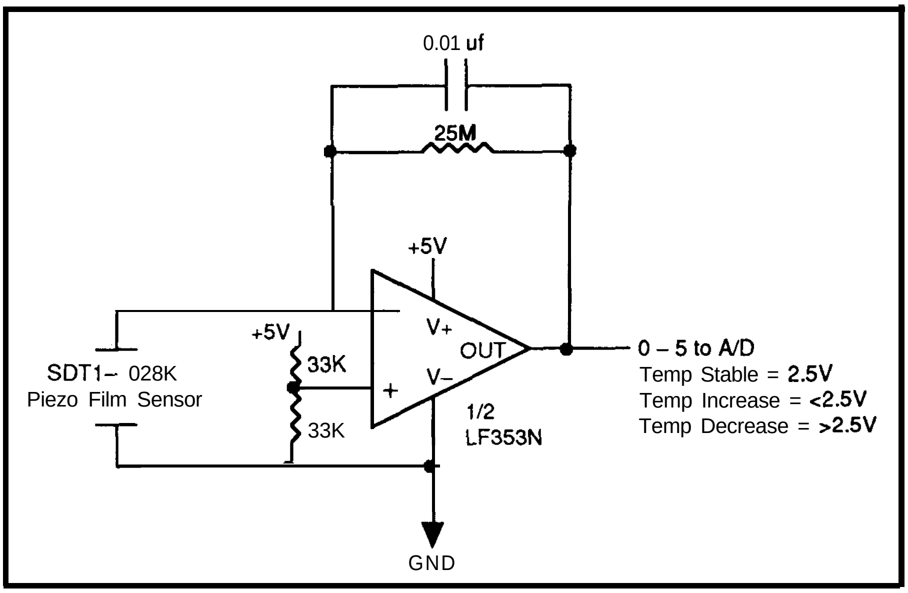


Figure 7—With the aid of a simple amplifier, the piezo sensor con be used to sense heat.

I discovered piezo film is kind of like the old Chinese proverb: "Be careful what you wish for, it might come true." The problem is that piezo, the "ultimate sensor," lives up to its billing and really senses lots of things whether you want to or not!

A little further (fortunately, less eventful) experimentation confirmed that I wasn't seeing a thermal effect at all.

In fact, checking the film output with a scope showed that turning the lamp on did induce piezo pulse-pulses with a suspicious 16.6-ms period. Yes, apparently my "thermal" sensor was actually a "60-Hz" sensor. Though a 60-Hz sensor may be neat, the point is that you have to be careful to configure your piezo-based system to measure what-and only what-you really want to measure. Turns out blocking ambient 60 Hz is a primary function of the shielded version of the sensor (SDT1-028K) also included in the kit. Shielding must be a nontrivial task. A basic DT1-028K sells for 50 cents, while the shielded SDT1-028K goes for $40!

In real-world piezo design, you'll find much attention given to sensor isolation and/or signal conditioning to extract the variable of interest from the myriad of noise sources the sensor will otherwise pick up. After all, it wouldn't do well at all to have piezo sensors that work fine, only as long as lighting, temperature, acoustic interference not to mention vibration are all controlled. The concept of common-mode rejection using multiple sensors is often the only solution. For example, an impact sensor mountedinamoving/vibratingenvironmentcanbepaired with another equal, but unimpacted, sensor. Then, the difference between sensor readings serves to isolate the impact signal from ambient vibration.

Ultimately, after switching to the shielded sensor, I was able to observe the pyroelectric effect working as it should. Along the way (after yet another trip to the'Shack), I cobbled up a more conventional interface circuit (Figure 7). I'm not sure if it's ideal, but I figure it isn't bad for a guy much more at home with opcodes than op-amps! Now, the controller begs for mercy as the flame gets closer.. .

Once the wilder side of piezo-its propensity to take in any signal off the street-is tamed, I think you'll find that products like Kynar can occupy a valuable spot in your sensor bag of tricks. ✣

CONTACT

Atochem North America
(formerly Pennwalt Corporation)
Piezo Film Sensor Division
P.O. Box 799
Valley Forge, PA 19482
(2 15) 666-3500

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

IRS

425 Very Useful
426 Moderately Useful
427 Not Useful

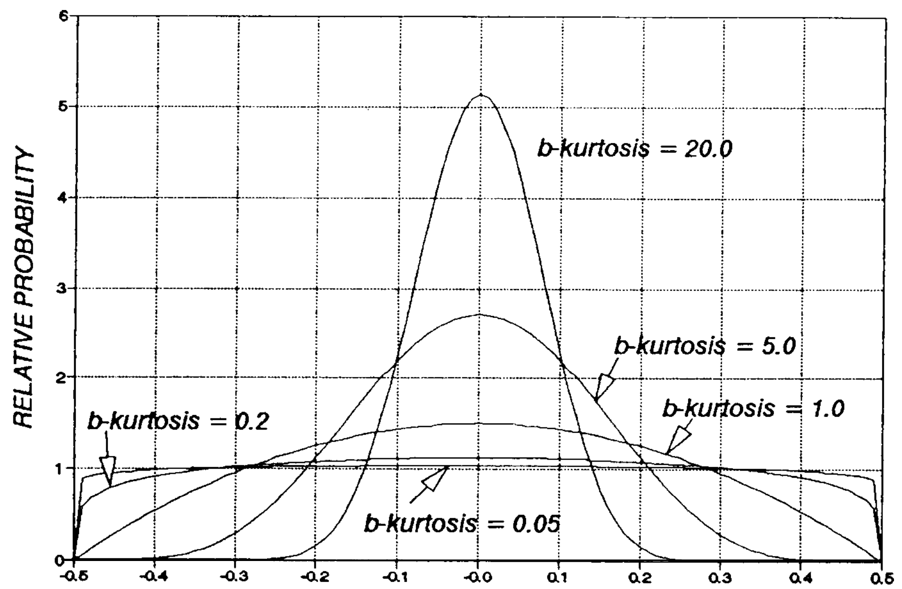


Figure 2— *When unskewed, all the distributions are centered around 0.0.*

inclusive. Within these limits each card has the same probability of being dealt, so here again the probability density distribution is rectangular.

3) The grades on a test given to a group of students lie between zero and 100 and never outside these bounds. The group average will lie between those limits. These distributions are seldom rectangular.

4) The Fog Test is used in the automotive industry to determine the tendency of materials to deposit fog onautomobile windows. Theamount deposited on the glass is characterized by the reflectance of its surface as measured by a Glossmeter. The reading can never exceed 100 and has, in fact, a vanishingly small chance of attaining that value. Similarly, a reading of zero is highly unlikely, and one that lies below zero or over 100 is impossible. Between these limits, however, the readings cluster about some value.

To represent such data, the normal curve is sometimes truncated; extreme values are removed and the ordinates recalculated to return the sum of all the probabilities to 1 .O. A truncated curve shows appreciable probabilities of measurements at the limits, which may not in fact exist, as in example4 above. Extreme values could also be removed from the normal curve by raising the x axis (that is, subtracting a constant from all probabilities) and using a factor to adjust the curve, but the probability of

THE SPREADSHEET PROGRAM

Figure 3showed a spreadsheet designed forcalculating bounded probability distributions. The columns include the majority of the calculations,

The second (B) column contains values of xfrom -0.5 to +0.5 in steps of 0.01. In the first column these numbers have been converted into corresponding values from 0 to 100, since these limits are rather common in bounded distributions. The third (C) column contains the probability for each value of xfound from equation (1), using the values of b-kurtosis and b-skew entered in their header locations (C3 and C4 respectively). The second number in the third column (Row 12) thus has the formula

```
@EXP((C$3*C$4)*@LN(0.5+B12) t C$3/C$4*@LN(0.5-B12))
```

The exponential form was used because the spreadsheet did not calculate some quantities well by direct exponentiation.

The integral of the third column is calculated by the expression

```
(@SUM(C11..C110) t @SUM(C12..C111))/200
```

Since the sum of all the probabilities must be 1 .0, the fourth (D) column is derived from the third by dividing ail the values in the third column by the integral of the third column: thus it expresses the "adjusted" probabilities. This column is also integrated, to verify that the sum is actually 1.0, by the same formula used in the third column.

The fifth (E) column calculates the weighted first moment of each probability about the Paxis. The first row of this column uses the formula

```
+B11*D11
```

which is copied down the entire column. At the bottom of this column the mean is

```
@AVG(E11..E111)
```

The weighted second moment about the mean is found in the sixth (F) column. The first row in this column uses the formula

```
+D11*(B11 - E$114)^2
```

where location E114 contains the mean. At the bottom of the sixth column the variance is calculated as

```
@AVG(F11..F111)
```

the standard deviation beneath it being the square root of thisvalue. Similarly, the seventh (G) column shows the third moment about the mean. The skew at the bottom of this column uses the formula

```
@SUM(G11..G111)/(@SUM(F11..F111)^1.5
```

Probabilities are accumulated in the eighth (H) column. The first value is 0.0; thereafter the values are calculated the formula

```
+H11 t 0.005*(D11 t D12)
```

in the eleventh row, copied down the entire column. The accumulation attains a value of 1.00000. The ninth (I) column contains the Identical accumulation shifted up one row, while the tenth (J) column is a copy of the second. The eighth (H), ninth (I), and tenth (J) columns are used for finding nonparametric characteristics of the curve by reverse linear interpolation. The formula for the median is

```
@VLOOKUP(.5,H11..J111,2) t .01*((.5-@VLOOKUP(.5,
    H11..J111,0)) / (@VLOOKUP(.5,H11..J111,1) -
    @VLOOKUP(.5,H11..J111,0)))
```

Quartiles are determined similarly using appropriate limits.

The header contains values of the mean, standard deviation, and skew transferred from their locations within the sheet. It also shows the location of the probability curve's peak (from the expression given in the treatment of the derivative) as well as the peak-mean difference. Nonparametric constantsare calculated in the header.

K = 1 .0000 (B-KURT)	MEAN = -0.28149	MN BOGEY = 0.1	MEDIAN = 0.3173
S = 1.3000 (B–SKEW)	STD DEV = 0.15447	QUART 1 = 0.1923	QUART3 = 0.4077
C.F. = 8.218747E+00	SKEW = –0.08742	MIDQUART = 0.3000	QUARTDEV = 0.1077
PEAKATX = 0.430862	PEAK-MN = 0.149368	BOWLEY SKEW = –0.1608	

[X]	X	F(X)	F(X) CORR	I MOM	II MOM	III MOM	F(X) CUM	NEXT CUM	X
1.0	-0.50	0.0000E+00	0.0000E+00	0.00000	0.00000	0.00000	0.00000	0.00000	-0.50
1.1	–0.49	4.5732E–08	3.7586E–07	-0.00000	0.00000	-0.00000	0.00000	0.00000	-0.49
1.2	-0.48	5.8021E–07	4.7686E–06	-0.00000	0.00000	-0.00000	0.00000	0.00000	-0.48
1.3	–0.47	2.5615E–06	2.1053E–05	-0.00001	0.00001	-0.00001	0.00000	0.00000	-0.47
4.9	-0.07	3.8775E–02	3.1868E–01	-0.02231	0.03937	-0.01384	0.03036	0.03368	-0.07
5.0	-0.06	4.1985E–02	3.4507E–01	-0.02070	0.04024	-0.01374	0.03368	0.03727	-0.06
5.1	–0.05	4.5371E–02	3.7289E–01	-0.01864	0.04098	-0.01358	0.03727	0.04114	-0.05
.		.	.	.		.	.		.
9.8	0.48	3.1973E–01	2.6278E+00	1.26133	0.10355	0.02055	0.96428	0.98871	0.48
9.9	0.49	2.7474E-01	2.2580E+00	1.10643	0.09817	0.02047	0.98871	1.00000	0.49
10.0	0.50	0.0000E+00	0.0000E+00	0.00000	0.00000	0.00000	1.00000		0.50
		INTEGRAL=	COR INT=	MEAN=	VAR=	SKEW=			
		1.2167E–01	1.0000E+00	0.28149	0.02386	a.08742			

Figure 3—*The* **use of a** *spreadsheet greatly eases the calculation of bounded probability distributions.*

Summarizing Your Data

PRACTICAL ALGORITHMS

Charles **P. Boegli**

Properties of a Bounded Probability Density Function

One mission of Statistics is to manage large amounts of data by summarizing it in various ways. Available for this work are a number of probability density functions which experience has shown can represent various types of data. When data fit any such function, reasonably confident assertions can be made not only about where additional data are likely to lie, but also about the process that produced the data,

The "normal" population curve describes a wide variety of data sets. It implies, in general, that the most probable value of a dimension lies at the mean (i.e., the average) of all the measurements. Anotherquantity (the "variance") denotes how closely the measurements cluster around the mean. If the variance or its square root, the "standard deviation," are small, the measurements are for the most part close to each other, while a largestandard deviationindicates the data spread widely about the mean.

The normal curve, irrespective of the standard deviation, shows small but finite probabilities of measurements that lie at very large distances in both directions from the mean. When thevarianceissmall, these probabilities are so small that little is lost by ignoring them. Not all the other statistical distributions share this property, but the great majority does allow finite probabilities of greatly divergent measurements on at least one side of the mean.

Certain distributions, however, are bounded in the sense that no possibility whatever exists that they can exceed certain limits. Here are four examples of such distributions:

1) A computer can be programmed to produce a set of pseudorandom numbers between, say, 0 and 10. Though the quantity of generated numbers may approach infinity, one that lies below 0 or above 10 has no possibility of existence. The distribution of numbers between 0 and 10 should be rectangular, if enough numbers are generated; any number between the limits is as likely to exist as any other number.

2) If all face cards are removed from a deck and the deck is shuffled, a single card that is dealt may be anything from one (an ace) to ten. If that card is returned, the deck reshuffled, and another card dealt, the same limitations apply. The process can be repeated ad infinitum and no card will appear that does not lie between one and ten

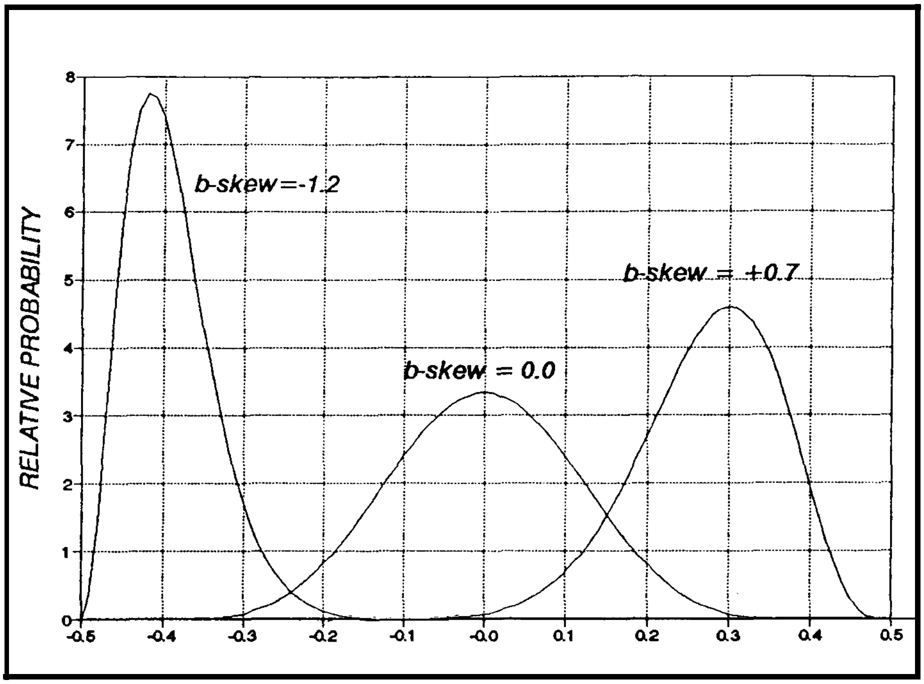


Figure 1 —*Several* **typical** *distributions with fixed k are shown.*

INTEGRAL

Except for special values of the exponents, the integral of the equation between the limits is not easy to find. It should be remembered that the expression for the Gaussian (normal) distribution also cannot be analytically integrated over limits other than from $-\infty$ to $+\infty$.

Where k = 1 and s = 0, the integral of the expression from -0.5 to +0.5 is 0.16666a. Thus, for this condition, the integral of the probabilities is 1 .O when a = **6.0.** Integration for other special values of *k* and s appears possible. The availability of spreadsheet computer programs makes analytical integration less important than it was in the past, since numerical integration is always possible.

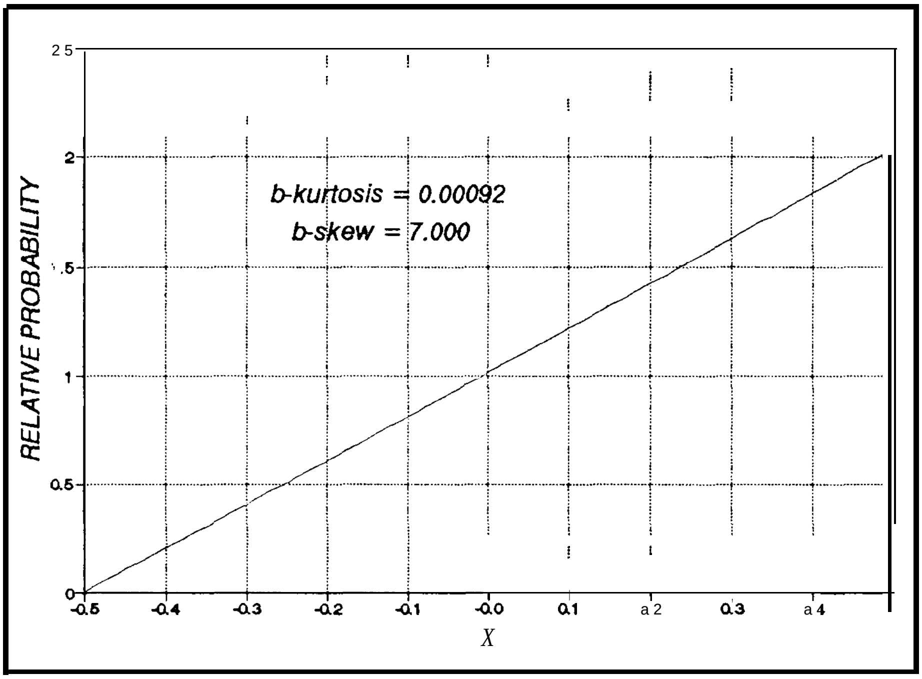


Figure 5—*This highly skewed distribution is nearly a straight line even though the endvalues are slightly in error.*

CHANGE OF LIMITS

To change the bounds of the distribution one uses the transformation

$$x = \frac{(X - L)}{(U - L)} - 0.5 \qquad (3)$$

in which X is the original variable, *U* is the original upper bound, and *L* is the original lower bound.

For a distribution lying between 0 and 100, for instance,

$$x = \frac{X}{100} - 0.5$$

which moves the original lower bound from 0 to -0.5 and the upper from 100 to +0.5. When the distribution has been characterized between the new bounds, reverse transformation restores the original values.

Changing the limits of the normal distribution by linear transformation is impossible, since both limits are infinite. We might alter them by substituting for x a function that has a finite value when x becomes infinite; for example, $\tan^{-1}x$ could be substituted for x in the normal distribution. This substitution distorts the distribution severely, and alters the integral between the limits.

DISTRIBUTION CHARACTERISTICS

The quantity s, which we have called the b-skew, can lie between $-\infty$ and $+\infty$, the midvalue being 0.0. It locates the peak in the distribution curve. Ifs is positive, the peak lies in the region from 0.0 to 0.5, while if it is negative the peak is between-O.5 and 0.0. For a given *k*, the substitution of-sforsproducesamirrorimageofthedistributioncurve about the *P* axis. Figure 1 shows typical distributions with fixed *k.*

Figure 2 illustrates several unskewed (s = 0.0) distributions described by equation (1). When the value of *k* exceeds 1.0, the probability density tails off smoothly to zero at the limits; the larger *k* is, the narrower the peak. When *k* is less than 1 .O, the distribution takes on a markedly different character. It no longer tails off to zero at the limits, but attains that value rather abruptly, and the peak becomes quite broad. As *k* becomes very small, the distribution approaches rectangular.

Investigation of the distribution characteristics was greatly aided by a spreadsheet fully detailed in the sidebar on page 98. Figure 3 presents one calculation done in this manner. Locating medians and quartiles proved to be an interesting challenge. They are obtained by reverse interpolation between table values, which in this case was done by adding an offset probability accumulation to the table, and making extensive use of vertical lookup capabilities.

The spreadsheet calculates a number of other characteristics of the distributions. The mean and the standard deviation are found in the usual ways, using the adjusted probability distribution. The sum of the third moments divided by the 3/2 power of the sum of the second moments yields the conventional skew.

Nonparametric constants found by the spreadsheet program include the median and quartiles; for unskewed distributions, the quartiles lie at equal distances from the median, the midquartileequaling the median. The "Quartile Deviation," a nonparametric equivalent to the standard deviation, is one-half the difference between the first and third quartiles. The "Bowley skew" (quartile coefficient of skewness), defined by

$$\frac{(Q_3 - 2Q_2 + Q_1)}{(Q_3 - Q_1)}$$

is a nonparametric skew measurement.

extreme values would remain at zero. This technique is evidently seldom used.

This article describes an apparently novel probability distribution that is bounded on both sides, yet appears to have enough flexibility to describe a wide variety of measurements subject to these limitations. Although the equation is entirely empirical, it makes interesting implications about bounded data, some of which are detailed here. Since an equation of this type has little utility unless it serves for calculation, I've also included illustrations of its application.

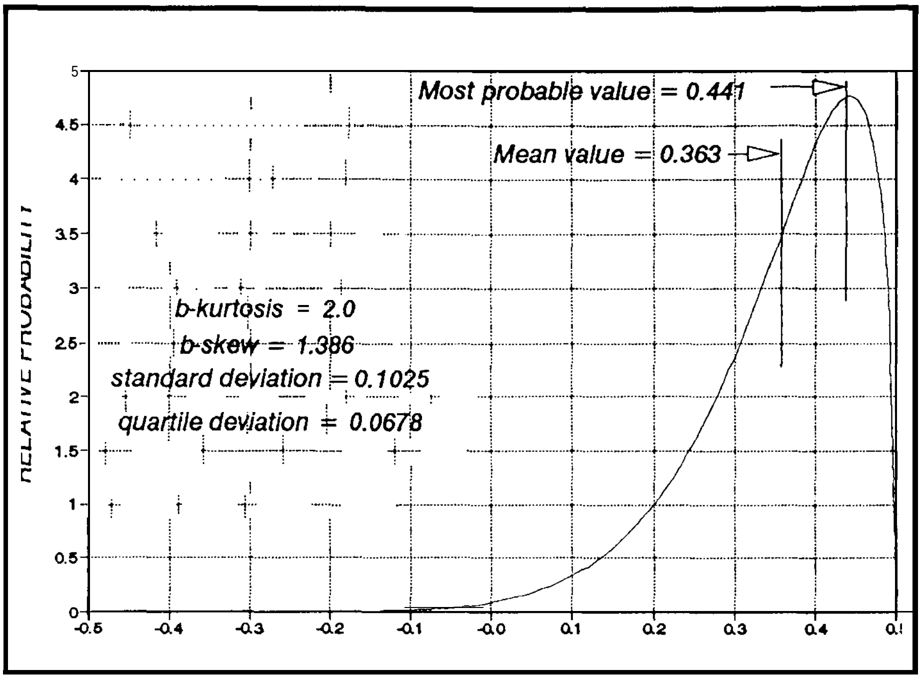


Figure 4—*The location of the peak in the distribution, and the not mean, represents the most probable value*

THE BOUNDED PROBABILITY DENSITY

The empirical equation for the probability density function describes a population that exists from -0.5 to +0.5, and is defined to be identically zero outside these bounds. Its equation is

$$P = a\left[(0.5 + x)^{\exp(s)} \times (0.5 - x)^{\exp(-s)}\right]^{k} \qquad (1)$$

in which P is the probability and x the location. The parameters k and s indicate the kurtosis (sharpness) and "skew" of the distribution. Kurtosis and skew are well defined for normal distributions, and the meanings ascribed to them here are not the same as those in common usage. For clarity's sake, the kurtosis as defined here will be called the b-kurtosis, while the skew as defined here will be called b-skew. The constant a adjusts the integral from -0.5 to +0.5 to be 1.0.

With only two arbitrary constants, this equation appears to describe a wide variety of bounded distributions. Of course, in any practical distribution, the bounds are not -0.5 and +0.5, but conversion of other limits to these is easy by linear transformation.

DERIVATIVE

The derivative of equation (1) with respect to x is

$$\frac{dP}{dx} = a\left[\begin{array}{l}\left[k \times \exp(s)\right] \times (0.5 + x)^{k \times \exp(s) - 1} \times (0.5 - x)^{k/\exp(s)} \\ -\left[\dfrac{k}{\exp(s)}\right] \times (0.5 + x)^{k/\exp(s)} \times \left((0.5 - x)^{k/\exp(s) - 1}\right)\end{array}\right] \qquad (2)$$

The derivative shows that:

a) At the lower limit, the slope of the curve is 0.0 when $k > 1/\exp(s)$. When $k < 1/\exp(s)$, the slope is infinite. When $k = 1/\exp(s)$, the slope is indeterminate.

b) At the upper limit, the slope of the curve is 0.0 when $k > \exp(s)$. When $k < \exp(s)$, the slope is infinite. When $k = \exp(s)$, the slope is indeterminate.

c) A maximum exists at $x = 0.5[((\exp(s))^2 - 1)/((\exp(s))^2 + 1)]$ which, it should be noted, is independent of k.

PEAK-MEAN DEVIATION

A distribution bounded on both sides must be skewed unless its mean lies midway between the bounds. To the extent that equation (1) describes such a distribution, the distance between the mean and the peak (most probable value) can be found with the spreadsheet program. The operation is somewhat clumsy in that a series of adjustments must be given to the b-kurtosis and b-skew to arrive at any specifed mean and standard deviation.

To surmount this difficulty, a macro was written that continually adjusts the skew to attain a 'bogey" standard deviation for any given b-kurtosis. Having determined the skew, the macro then increments the b-kurtosis and repeats the calculation. In this manner, data were derived forvarlousvaluesof b-kurtosisfrom 0.5 to 256 in step-ratios of 2.0, using means from 0.1 to 0.45 in steps of 0.5. The accumulated data are plotted in Figure I.

With this figure, knowing the "reduced" mean and standarddeviations,onecanestimatethe "reduced" peak-mean difference. Finding the most probable value of the distribution is then merely a matter of adding the difference to the "reduced" mean, and restoring the values to their original bounds.

A similar macro was written for nonparametric quantities. Figure II performs the same function as Figure I but uses midquartiles and quartile deviations instead of means and standard deviations.

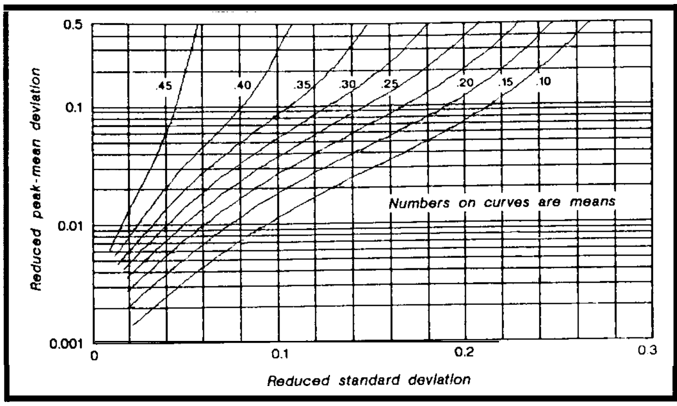


Figure I—*The b-kurtosis and b-skew must be adjusted to attain a specified mean and standard deviation.*

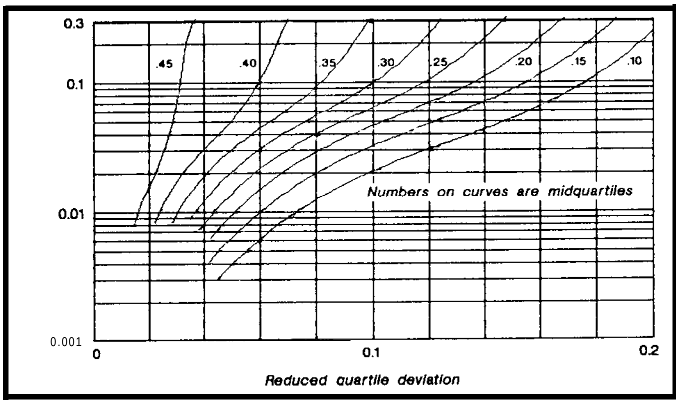


Figure II-*Similar to above, but using midquartiles and quartiles.*

DISTRIBUTION BEHAVIOR SUMMARY

Behavior of these distributions can be summarized in numerous ways. For the purposes of this article, the distribution mean was first used as a parameter. For each mean between 0.1 and 0.45, a curve was derived relating the peak-to-mean distance to the standard deviation. These curves are presented in the sidebar above. This choice was made because of common familiarity with the calculation of the mean, and also because the determination of "central tendency" is one primary object of this work. Similar curves have been drawn for nonparametric distribution parameters.

The mean ordinarily indicates central tendency because in unskewed normal distributions it represents the most probable value. The bounded distributions considered here must be skewed except in the trivial case where b-skew = 0.0. The location of the peak, not the mean, represents the most probable value, as Figure 4 shows. The curvesabove can be used to locate the peak when the mean and standard deviation are known.

CALCULATION EXAMPLES

To illustrate the use of the curves, we use a set of reflectance readings from eight Fog tests run on a single material. The readings from the glasses had a mean value of 82.2 and standard deviation of 4.47. Since these readings are on a scale of 0–100, the reduced mean is 0.322 and the

SKEW IN BOUNDED DISTRIBUTIONS

An interesting characteristic of bounded distributions, discovered while using the spreadsheet macros mentioned in the second sidebar, is that for any value of b-skew there exists a b-kurtosis for which the skew is zero. To one side of this value the skew is negative and to the other side, positive. The quartile coefficient of skewness does not similarly go through zero. One example of this situation is where k = 48.690 and s = 0.80. Reduced value of the mean is 0.3238, the standard deviation 0.0328, the midquartile 0.3278. and the quartile deviation 0.0224. The Bowley skew is -0.0374 but the skew is 0.0000.

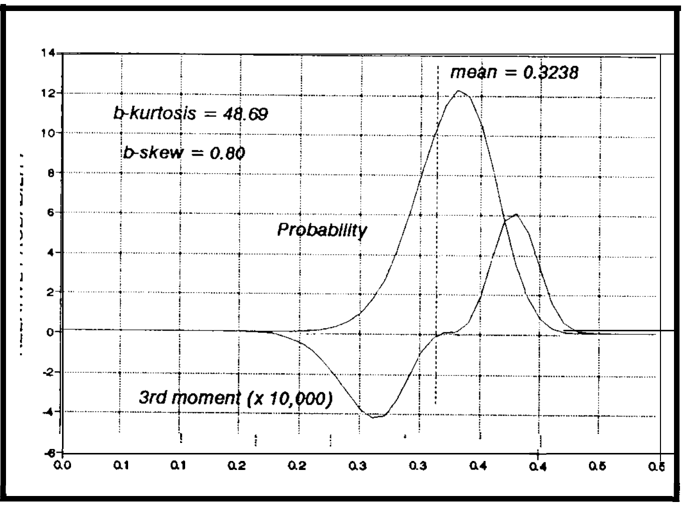


Figure III—*This shows the distribution and the third moments over an expanded portion of the range.*

Although this is at first surprising, Figure III explains it. This graph shows the distribution and the third moments over an expanded portion of the range. The shape of the curve displaces the mean to the left of the peak enough that integrated third moments on each side of the mean are equal, The equality of the integrals of the third moment on each side of the mean is quite evident.

The implication is that even though bounded distributions are usually skewed, the customary calculation of skew may not disclose it. This effect is unlikely with nonparametric quantities, and has not been observed so far.

reduced standard deviation is 0.0447. Figure I shows that the peak-mean deviation is 0.012, making the most probable value 82.2 + (100)(0.012) = 83.4.

Although the curve can approach a rectangular distribution closely, the approximation is not perfect; a broad maximum exists between the limits. The most probable value of a rectangular distribution (b-kurtosis → 0) scaled between 1 and 10 is always shown as 5.5.

The probability function represents distributions pretty well even if they have finite probabilities at the end points, but not beyond them. Consider, for instance, a deck consisting of a single ace, two twos, three threes, and so forth up to ten tens. The sum of all the numbers on the 55 cards is 385, making the mean value 7.0. The reduced mean is 0.16269. Figure 5 shows a curve for b-skew = 7.000 and b-kurtosis = 0.00092, for which the reduced mean is near 0.16667; the distribution is nearly a straight line even though the end values are slightly in error. For this highly skewed distribution, the most probable value is correctly shown as 10 (–); the mean (7.0) merely represents the average of all numbers dealt after sufficient dealings. ✣

I want to express sincere gratitude:

To Messrs. Gerry Goldfinger, Chuck Stukins, and Michael Leonard, my previous associates at General Motors Inland Division, for many stimulating discussions, and also because frequent disagreements never tempered our mutual respect.

To Mr. Jim Nuckols and especially to My. Mark Mattix, whose shared interest in computers made us friends, and who were unfailingly helpful in supplying tools that I needed.

And lastly to my own computer, which in minutes makes calculations that would have required months of intense labor thirty short years ago.

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

IRS

428 Very Useful
429 Moderately Useful
430 Not Useful

CONNEC-TIME

Conducted by
Ken Davidson

Excerpts from fhe Circuit Cellar BBS

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

We're going to deal primarily with simple (or what should be simple) I/O peripherals in this installment of ConnecTime. We'll start with a discussion of some **of** *the quirks of the venerable 8255 PPI fhe astute programmer must be aware of when dealing with the chip. Next, we'll look at accurately generating time delays, and finally we'll discuss dealing with the dreaded RS-232 connection.*

Msg#:37143
From: WALTER CRUDUP To: ALL USERS

Quick question. I've got an 8255 connected to an 8051 at address A000h to A003h, and set to mode 0. Will I bc able to read the last data1 sent **to a** port configured as output if 1 do a read of this port?

Msg#:37154
From: KEN DAVIDSON To: WALTER CRUDUP

Yes, you can read the state of an output port on the 8255 by simply reading the port back. Makes a nice quick sanity check to make sure the part is working.

Msg#:37168
From: WALTER CRUDUP To: KEN DAVIDSON

Thanks for the info Ken. Right after 1 posted this message I went back and read the Intel data book again looking for this info (for about the tenth time) and there it was, stating that a read from an output port will return the data written. It is truly amazing how things can be overlooked.

Msg#:37188
From: BOB PADDOCK To: JEFF BACHIOCHI

Toshiba 82C55s won't let you read back the command register.

Msg#:37198
From: KEN DAVIDSON To: BOB PADDOCK

That's right. None of the 8255s I've ever used will allow you to read back the configuration port; only the I/O ports.

Msg#:37231
From: BOB PADDOCK To: KEN DAVIDSON

I know at least one of them did let you, but I don't remember who's, because that "feature" was in some of our equipment's self-test code. And we had to change the code when we found that not all of them would work that way. I know that it was the Toshiba part that brought this to our attention.

Msg#:37257
From: ED ROBINSON To: WALTER CRUDUP

Beware, I seem to recall having problems with the output port read if there was a write to the control port. Even if the mode of the output port was not changed.

Msg#:37282
From: KEN DAVIDSON To: ED ROBINSON

That's right, too. When you change thecontrol port, all theoutput ports lose their values, even if they are still output ports in the new control word.

The 8255 can be a strange beast to work with. I guess I've used it so much that I've just taken these idiosyncrasies for granted.

Generatingaccuratetimepeviodscan be tricky in the world of event-driven microcontrollers. Here, we look at some ways for doing if using an 8031 processor.

Msg#:36294
From: AL DORMAN To: ALL USERS

I am doing a project where I need to set some BCD switches and tell the microcontroller (80C31) to turn on a port pin for O-9 minutes and /or O-90 seconds. In an attempt to not reinvent the wheel, does anyone have a suggestion on how they have done this in the past to get an accurate one-second count. I plan on using the internal timer interrupt and trimming it to get my one-second interrupt. 1 cannot use the external interrupt because I am using P3 for switch strobing and external device driving. P1 is used for switch inputs and option switch inputs. Should I use a specific crystal? Is the internal timer interrupt stable enough to perform the timing I need? Any comments?

Msg#:36342
From: SANJAYA VATUK To: AL DORMAN

Just how accurate do you wanna be? If you need to be dead nuts, you are better off grabbing a zero crossing from the 60-Hz line (assuming your widget is line-powered). See Ed's line-monitoring thingy in issue #15 of Circuit Cellar INK for details on how to do this. If you want to bc real close, by all means use the timer. Otherwise, you are probably better off using a 12-MHz crystal if you don't need RS-232. This divides down into a l-MHz cycle frequency, which makes calculation _much_ easier: 1 µs per cycle. Aside from any inaccuracies in your oscillator, keep in mind the variability of the 8031's interrupt response time. At 12 MHz, the latency can vary between 3 and 9 µs (7 µs if you don't use the MUL or DIV instructions).

Interestingly enough, I was just playing around the other day trying to see how accurate a delay loop I could create using an 11.0592-MHz crystal, using nested loops. (All I _really_ needed to do was flash an LED for half a second, but I thought I'd try and make it _exactly_ half a second!) I finally got it 99.99+% accurate, including the call and return. Believe me, 12 MHz is easier!

Msg#:36360
From: AL DORMAN To: SANJAYA VATUK

I do have some 6-MHz crystals, so I'll multiply everything by two and try that. It doesn't have tobedcad nuts, but in my application if another coin is inserted, the time accumulates relative to the basic set time. So if you set the timer for one minute and insert 100 coins, you should get 100 minutes of accumulated time. This could accumulate any timing error if it is in the seconds and cause problems. Microsecond errors shouldn't be a problem.

Finally, we all know the frustration of trying to get a pair of devices to talk to each other over an RS-232 connection. The last discussion offers some suggestions for dealing with such difficult situations.

Msg#:37323
From: CHESTER D. FITCH To: ALL USERS

Hi. Here's a project idea for all you analog types.

I just finished a frustrating job of interfacing two computers together through several serial ports. What made it difficult was that I could not really see the characters fly back and forth between systems. I could see SOMETHING get passed between systems, but I was using a protocol that required that I see what the characters WERE in order to debug the software. I eventually had to borrow a $15,000 network analyzer from someone at work in order to solve my problem.

This problem has led me to consider: why can't we come up with a simple (cheap) analyzer? My thought was that a relatively inexpensive embedded microcontroller/display board/and serial-line interface would solve this problem. Since I do software for a living, I see no problems with that end.

What I would like to ask is: do any of you analog types out there know of an interface circuit to RS-232? It should be transparent on the line (i.e., no loading)? While I dabble in digital circuitry a lot, and can build an analog circuit from schematic with the best of them, I do very little (or no) analog design.

I would appreciate hearing from anyone about this. Thanks.

Msg#:37326
From: JIM STEWART To: CHESTER D. FITCH

A friend of mine once wrote a program for a PC that did what you want. We connected the two receive lines from COM1 and COM2 to the RS-232 line under test and wrote software to display the transaction on a split screen. It worked pretty slick, although I don't have any more information on it other than this. It would be a nice thing to have in the public domain.

Msg#:37341
From: FRANK C. SERGEANT To: CHESTER D. FITCH

Regarding snooping on the serial lines, there are a number of possibilities. First, how fast does it have to go? Second, can you get an extra PC or laptop near the two devices that you want to monitor?

If these first two tests are passed, here's what I would do first. I'd route the two devices' serial lines *through* the extra PC (it would need two serial ports). Then the extra PC keeps checking the two ports. When a character is ready from device A, it reads it, stuffs it in a buffer, and forwards it to device B. When a character is ready from device B, it reads it, stuffs it in a buffer, and forwards it to device A. Providing the timings work out (and the extra PC is available) this should work like a dream. Perhaps, you can *make* the timings work out with handshaking.

I presume you want to know not only the characters but the exact interleaving of them. In this case, stuff characters from both devices into the same buffer, but with tags attached to indicate where they came from. A simple polling of the serial ports is probably sufficient. (If it's not fast enough, you might need interrupt routines for the serial ports in the extra PC.)

So, to summarize, your two devices *think* they are talking to each other and don't realize the messagesare relayed through the extra PC. The program running on the extra PC could leave the characters in the buffer for later display (fastest) or could display them on the screen as they are received.

Alternatively, you could tap into the two serial data lines, feeding them into, perhaps, a 4049 or 4050 CMOS buffer, diode clamped so they won't go negative, and then into a parallel port on any computer and do the serial decoding in software. Use sockets for the 4049s as they are their own fuses. For a test setup, incredibly sloppy hardware will work just fine! It's all digital, you don't need to worry about analog. (I've almost always been able to drive serial lines with just Oand 5 volts—no negative voltage.) So, while you might not want to ship a product with "quick and dirty" RS232, this should be no problem in your *laboratory*, huh?

On the other hand, if your devices *must* run at 115,200 bps with no handshaking or delays allowed, the second of my two methods might still work if you have a fast enough computer. In most cases, I imagine, it would be possible to slow down the data rates to 300 bps or 110 bps to make it easy for the snooper to keep up.

Msg#:37412
From: ED NISLEY To: CHESTER D. FITCH

Turns out I needed something like that for a Firmware Furnace column I recently completed...

I made up an octopus connector that broke out the transmit and receive lines between the two devices and sent them to the receive lines of two serial ports.That way the devices could talk normally while the two additional ports recorded the conversation in each direction.The devices use plain ol' ASCII, so a terminal emulator worked just fine.

Turned out that the "master" device was COM1 on my PS/2, while the two monitors were COM3 and COM4. One of the nice things about OS/2 is that I could run two copies of REXXTERM to watch COM3 and COM4 while single-stepping the "master" program driving COM1—all at the same time!

There are commercial PC programs available that do this sort of thing with all manner of bells and whistles (back pages of *Dr. Dobb's Journal* and *Computer Language* have 'em), so it's pretty much a solved problem.

Msg#:37713
From: PELLERVO KASKINEN To: CHESTER D. FITCH

Although there are already some answers, nobody seems to mention the commercial product that I bought and have been extremely pleased with: Break-Off II. I bought it from BB Electronics, but think I have since seen somebody else advertise it. The price for an AT (two COM port) version was in the $150 range, if I remember. It included the software and a special eavesdropping cable. You plug the central body with its two connectors into the line between the two communication devices you plan to monitor and the two pigtails to one COM port each of an AT class computer.

You have several choices for the configuration, but the most powerful for my needs has been one where I capture into memory a range of bytes to be analyzed later on. Each data byte is accompanied by one status byte that tells which port the data byte came and all the handshake line status information normally used in the RS-232 type communication. You get the exact order of the signals in each direction. No timing information of course. If I need that, then there is no other way but to hook up a logic analyzer, but its memory length is generally too short for real-life message analysis.

The Circuit Cellar BBS runs on a 10-MHz Micromint OEM-286 IBM PC/AT-compatible computer using the multiline version of The Bread Board *System (TBBS 2.1M) and currently has four modems connected. We invite you to call and exchange ideas with other Circuit Cellar readers. It is available 24 hours* a *day and can be reached at (203) 871-1988.* Set *your modem* **for** *8 data* bits, *1* stop *bit, and either 300, 1200, or 2400 bps.*

IRS

431 Very Useful
432 Moderately Useful
433 Not Useful

SOFTWARE and **BBS** AVAILABLE on DISK

Software on Disk
Software for the articles in this issue of Circuit Cellar INK may be downloaded free of charge from the Circuit Cellar BBS. For those unable to download files, they are also available on one 360K, 5.25" IBM PC-format disk for only $12.

Circuit Cellar BBS on Disk
Every month, hundreds of information-filled messages are posted on the Circuit Cellar BBS by people from all walks of life. For those who can't log on as often as they'd like, the text of the public message areas is available on disk in two-month installments. Each installment comes on three 360K. 5.25' IBM PC-format disks and costs just $15. The installment for this issue of INK (August/September 1991) includes all public messages posted during May and June, 1991.

To order either Software on Disk or Circuit Cellar BBS on Disk, send check or money order to:

Circuit Cellar INK — Software (or BBS) on Disk
P.O. Box 772, Vernon, CT 06066

or use your MasterCard or Visa and call (203) 875-2199. Be sure to specify the issue number of each disk you order. Please add $3 for shipping outside the U.S.

STEVE'S OWN INK

Steve *Ciarcia*

A Standard Column

I've been doing a lot of work lately, work that has me thinking about the question of standards. Specifically, I've spent a lot of time with my nose buried in data books trying to figure out the best components for several boards I'm designing. My job would be easier if I didn't have to thinkabout changing thcboard layout every time I look at another DC-DC converter. I knew that there were a lot of packaging specifications for analog components, but several weeks spent tearing my hair out has brought the point home with tremendous effectiveness. It's also started me thinking that there must be a better way to get components from a factory onto my boards.

My first thought (after the fifth or sixth handful of hair) was that there ought to be strict standards for physical package size and pinout configuration. Maybe there could be a huge government agency to enforce strict compliance with the standards. Maybe they could send out commando teams to blow up the buildings of any company that produced a stupid proprietary package that wouldn't fit on my boards! Maybe I'm going overboard. After all, I don't really want a lack of choices, I just want to be able to work on the electrical side of a design without having to spend hours on the physical package every time I look for an alternate component. Furthermore, the record of standards imposed from outside an industry is not good.

Now, there's an idea. Standards that develop because a lot of people used a particular product or design have tended to be fairly robust and dynamic standards. Look at the IBM PC. In 1981 it was just one of the many personal computer designs floating around. Now, it's the de facto standard for personal computing because thousands upon thousands of people bought the darned things. If I could convince a bunch of engineers that they need to change their designs to use the type of component I favor, it could become a standard. Then my problem would be solved and I'd probably get a reputation as an industry leader or something.

The problem with the second idea is that components aren't desktop computers. I buy a computer with the idea that it should run the wide variety of software and have a good selection of expansion boards and peripherals available. I don't measure my desk and find a case to fit. When I design a board, on the other hand, I have specific applications and system configurations in mind before I ever sit down to draw a schematic. When I pick components, I'm generally looking for parts that will be the best fit for my specific needs, rather than parts that are the most universally available or generically configured. I have to believe that most engineers do the same. Still, it would be nice if there were a bit more standardization on general sizes, pinouts, and specifications for common components. Sigh.

The more I think about the problem, the more I realize that the real issue is caught up in the way that the electronic and computer industries developed. I'm not going to go around saying that thecomponents companies are the last bastion of free-form capitalism, but there has been a lot of freedom in the way the products came to be. Most products have been the result of an idea for a new product or improvement. Innovation reigned, with lawyers standing by to make sure that there was enough innovation in the new products. Standards happened (or didn't happen) by the occasional government action (MIL-spec parts) and by committees generally made up of competitors and large users, each with a particular axe to grind and no strong motivation for real standardization. Despite all of this, a few standards have emerged. There arc accepted package sizes for microprocessors, memory chips, and other ICs. Likewise, there are common standards for many of the "garden-variety" components like resistors and capacitors. Why, then, has it been so difficult to establish anything like a standard for anything having to do with power? Transformers, DC-DC converters, inverters, and the like are all over the spectrum in size and electrical connections. I know that some of the differences have to do with the physical requirements of the component, but I refuse to believe that there are no ways to standardize at least some of the specifications of these vital components.

I don't know what the final answer is. I've never served on a standards committee, so it may be that the metaphysical aspects of standards represent a truly insurmountable hurdle when it comes to power components. I have to believe that we would all benefit if there were a few more things that engineers could count on when we started to design a circuit.