CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

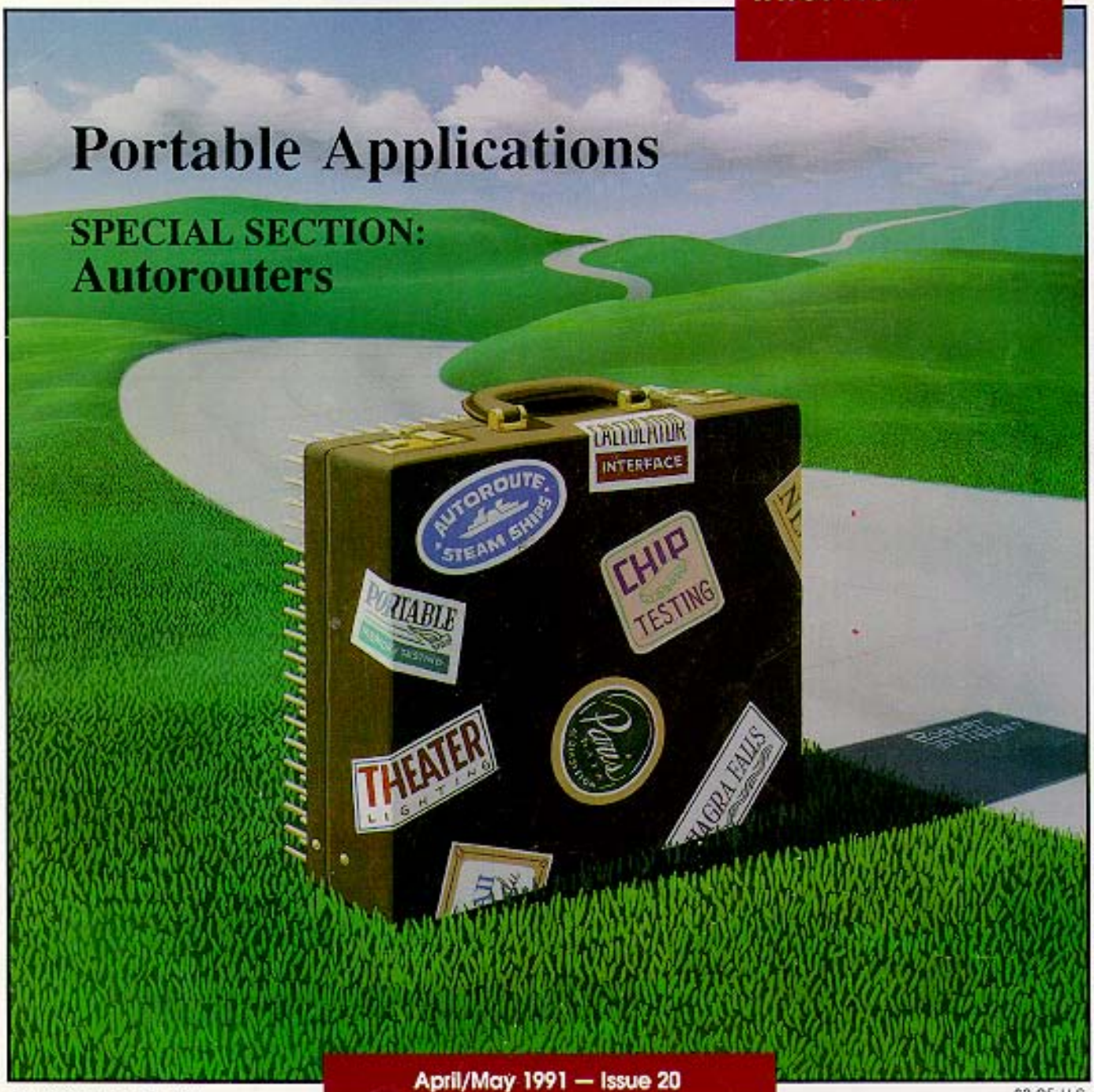
A Portable DRAM Tester

MIDI Sampled Sound

Working with Intel Hex

Portable Applications

SPECIAL SECTION:
Autorouters



April/May 1991 — Issue 20

0 74470 75349 0 04

$3.95 U.S.
$4.95 Canada

Towards More Personal Computing

EDITOR'S INK

Curtis Franklin, Jr.

It wasn't hard to predict. Computers started out as large dramatic things served by cadres of professional keepers. The trend toward smaller, more powerful, and less power-hungry computers started almost immediately. From UNIVAC, through IBM and Digital Equipment, with strong assistance from engineers at NASA, computers just kept getting smaller and smaller. Of course, history accelerated a bit when the folks at Intel decided that a general-purpose microprocessor was the best solution to a customer's request for a calculator engine, and the 4004 was born. If you're reading this magazine, you know the rest. The amazing thing is not that the trend has extended so long, but that there are so **many** people willing to believe that the progress has come to an end.

The power that once required rooms and now sits on desks is being liberated from its steel boxes and transported into, and by, our backpacks, attache cases, and coat pockets. The rush to have computers constantly by our sides assures that serious resources are dedicated to making computers even smaller and more powerful. The technology is now in place to let true "information appliances" become as useful andportable asa Victorinox Champion.

Jerry Pournelle said, some time ago, that the day was fast approaching when the answer to any question (with a known answer) would be available to any person. For all practical purposes, that day has arrived. If you use on-line services, you realize what an important information source they are, and CD-ROMs and other tools are available to a growing number of people. Jerry was, if anything, conservative in his prediction. It is now safe to say that the day is fast approaching when any known answer will be available to any person from any location. The mechanism will be the personal computer as it will evolve.

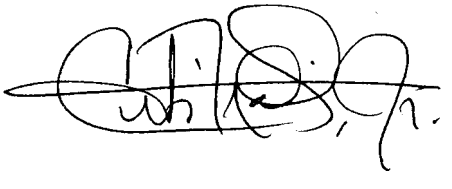
First, the personal computer will be small and light. The default footprint is 8.5" x 11", and four pounds the outside weight limit. There will be a color screen of at least VGA quality, a keyboard, voice input, and a stylus. Why so many input means? We use different technologies for different tasks. For a quick note, a signature, or working with a GUI, a stylus makes sense. For lengthy writing or precise work with numbers, nothing beats a keyboard. Finally, there are times when the spoken word communicates as nothing else can. Given the benefits of a solid user interface, the personal computer will need storage.

The first must in data storage is CD-ROM. Reference works currently available are a rich resource for those who can take advantage. Working storage will be in the form of a 100–200-megabyte hard disk. Floppy disk drives will be available, but not necessarily something always with you—when you need one, you'll know where to find it. On thinking about what I use floppy disks for, I realized that there are only four functions: data storage; hard-disk backup; software installation; and data transportation. The first of these is taken care of by the on-board hard disk. The second two are important, but can easily be done at set times, when an outboard disk drive is convenient. The last gave me the most concern, but it is easily accomplished if you are always connected to a network.

Future laptops will have built-in network support: Not for anything as limiting as an Ethernet LAN, but for a large public network with access through both land-line and cellular telephone systems. Email, voice mail, file sharing, even large resource sharing will all be through the nets. This is a vision of the computer as a communications appliance, replacing beeper, answering machine, cellular telephone, mailbox, and more. It's not hard to imagine my future laptop replacing Daytimers as the symbol of an organized life. For the replacement to occur, however, one more truly difficult obstacle must be overcome-price.

All the features I have described and more can be designed into a computer, but if that computer is priced at $20,000 the influence will be small. If, on the other hand, manufacturing and packaging engineers can produce the features listed at a retail price of $1000 or, better yet, $500, then a real revolution will occur.

Doubters will proffer countless reasons for the computer I've described never to occur. I've pinned my optimism to the broad human drive for a "higher quality of life." We **recognize** that information is an important part of how wedefine "qualityof life," and thecomputer sketched above is a marvelous information tool. It, or something like it, will happen, and help us get on with the ever-evolving business of life.



CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

In This Issue...

FEATURES

SPECIAL SECTION: Autorouters

DEPARTMENTS

Circuit **Cellar** BBS-24 Hrs. 300/1200/2400 bps, 8 bits, no parity, 1 stop bit, (203) 871-1988.

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

Circuit Celbr INK makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors. Furthermore, because of the possible varii ation in the quality and condition of materials and workmanship of reader-assembled projects, Circuit Cellar INK disclaims any responsiblity for the safe and proper function of reader-assembled projects based upon or from plans, descrip tions, or information published in Circuit Cellar INK.

CIRCUIT CELLAR INK (ISSN 08968985) is published bii monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second-class postage paid at Vernon, CT and additional offices. One-year (6 issues) subscription rate U.S.A. and possessions $17.95, Canada/Mexico $21.95, all other countriesS32.95. All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank. Direct subscription orders to Circuit Cellar INK, Subscriptions, P.O. Box 3050-C. Southeastern, PA 19398 or call (215) 630-1914.

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

Letters to the Editor

READER'S INK

JUST DO IT

Kenneth J. Ciszewski's letter (CIRCUIT CELLAR INK #18) prompted me to express something which has been on my mind for some time. Mr. Ciszewski is, of course, correct: "Changes where your code is going" is a better subtitle for my ComeFrom article than "Discover where your code has been." ("Implementing a ComeFrom Statement," CIRCUIT CELLAR INK #15)

Considerationsof subtitling **aside,** I must state that my association with CIRCUIT CELLAR INK's editorial staff has been an extremely valuable experience. They have always been receptive to my ideas and have worked wonders smoothing my grammatical faux pas.

Writing is good for you-just like eating lima beans. It forces you to think in a way which will be intelligible to others-and that may be very different from how you present the same information to yourself. Ask any manager: One roadblock to career advancement for engineers is an inability to write succinctly.

Having said that, I feel compelled to encourage potential authors to remove that roadblock by publishing their work.

J. Conrad Hubert
St. Paul, MN

We accept thecriticism and appreciate thekind words. We'd also like to encourage our *readers to take* word processor *in hand and writeabout the projects they've completed. Most of* CIRCUIT CELLAR *INK's authors are not professional writers-they're professional programmers and engineers. We're used to helping technical professionals become good writers, so start writing!* ***Ed.***

ADVICE AND DISSENT

I recently discovered your magazine and I have been reading it with interest. I am a consulting engineer practicing in the area of connecting devices to computers. Your editorial content produces articles close to my practice.

I have some comments on CIRCUIT CELLAR INK #17. The first is on the "Multichannel Digital Voltmeter Interface." On page 53 is shown a schematic for a full-wave rectifier (i.e., it produces an output for both positive and negative portions of the input signal). The point I want to concentrate on is the capacitor connected to the positive input of the LM324. This terminal can have as much as 259 nA of bias current. The circuit does not include any path for this current except for the capacitor. Thiscurrent and capacitor combination can cause a drift in the circuit of some 25 mV per second. Inaddition, thecircuit will drift into saturation after a long operating time. One solution to this problem would be to place a resistor to ground from the input pin. Using a 1-MΩ resistor will control the offset to less than 250 mV (still a big value). Of course, the amplifier impedance will drop, but that's what design is all about.

Another comment is on the article "Using C for Embedded Control." On page 69 is shown a circuit including a transistor speaker driver. Now, my concern here is in driving an inductive load (the speaker) with a switched transistor. What happens here is the old V = L di/dt problem. The current through the speaker will try to remain flowing as the transistor is switched off. This will cause the voltage to rise. The voltage will rise until the transistor breaks down (around 40 V). This action places a stress on the transistor which might lead to failure. One fix for this effect is to place a diode across the speaker. Here the diode cathode is placed on the positive terminal. This diode will **then** absorb the reverse voltage produced by the speaker inductance. Of course, the speaker will take longer to recover from the pulse thereby possibly causing its sound to change, but that's what design is all about.

And now to the back page: I found the comments on reinventing the computer interesting. A portion of my practice is spent designing and programming small embedded computers. There are still many situations where a special design is indicated. These places occur where the client is planning to market a device and he has specific packaging constraints. Of course, most "special" designs are really packaging exercises. The physical nature of the design is usually much more difficult than the electronic side. On the other hand, the programming typically is special and straightforward in most cases. In the column, there is an implication that the $119 board will take the

place of the $40k design. The article did not state the complexity of the software required to perform the protocol translation. That effort could easily require thousands of dollars of effort.

Frank **Bosso**
Danbury, CT

In his "Editor's INK" column of CIRCUIT CELLAR INK #19, Curtis Franklin, Jr. notes that embedded systems comprising 500,000–1,000,000 lines of code will become more prevalent, and that to produce such systems "you.. .need a high-level language, heavy-duty libraries and support programs, and a debugger that will work with you in the most intimate fashion." It is not surprising that Mr. Franklin has seen the future, since the items that he **has mentioned** are available right now. The language is Ada. Designed directly in response to the requirements of large embedded systems, Ada contains a variety of facilities that are essential for that application domain:

- *Modularization* features that support a variety of composition techniques, including the increasingly popular object-oriented design method, and that separate essential interface information from implementation details;
- *Type-checking*, whichallows early detection of errors;
- Language-defined features for *exception handling* and *tasking;*
- A *separate compilation* mechanism that allows both bottom-up and top-down development and that eases system integration by enforcing interface checking incrementally;
- *Low-level* features, including interrupt handling and control over machine representations.

In addition to these technical features that support embedded systems programming, Ada has the advantage of being a mature language standard (military, ANSI, and ISO) and having production-quality compilers on a variety of platforms. Run-time efficiency of Ada code is comparable to other languages, and sophisticated tool sets (including cross-debuggers integrated within-circuit emulators) are available.

Ada is no stranger to the demands of time-critical embedded systems in practice. It was the language of choice on a number of large projects, including several European air-traffic control systems, avionics software for the Boeing 747400, process-control for a General Electric steel mill rolling application, and many others.

In short, programmers who need to develop large embedded systems do not have to wait for the future for a language and tools; Ada is ready and available now.

Benjamin M. Brosgot
Vice President, Alsys Inc. and

Chairman, Commercial Ada Users Working Group for ACM SIGAda
Burlington, MA

[Editor's Note: *Alsys is a vendor of Ada compilers and development systems.1*

Vaughn Orchard
Marketing Manager
BSO / Tasking
Waltham, MA

We *hope to take another look at the BSO/Tasking C compiler in its de-dongled state.*

We *have decided, based on our experience with several softwarepackages, toinstituteapolicyof not evaluatingsoftware that incorporates* copy *protection. We understand the legitimate property theft concerns of software vendors, but feel that copy-protection schemes, in their current forms, place an undue burden on legitimate, paying customers.*

We *will focus our attention on those products that do not unnecessarily inconvenience their users with intrusive copy - protection schemes.* ***Curtis*** *Franklin*

C COMPILER UPDATE

THE DASTARDLY DANGLING DONGLES ARE DOOMED!

Yes, it's true. We agree with your feelings about those copy protection devices we've been using. [BSO/Tasking C Compiler, "0, Say Can You C?" CIRCUIT CELLAR INK #19.]

As you can imagine, we have had many discussions about them. One customer told us he had to have six different keys on the back of his PC for six different products.

In December we decided to stop using them and just use a serial numbering system for Customer Support purposes. Of course, it will take us a couple of months to get the dongle-related stuff out of the products and the production system. The de-dongling is underway.

We Want to Heai From You!

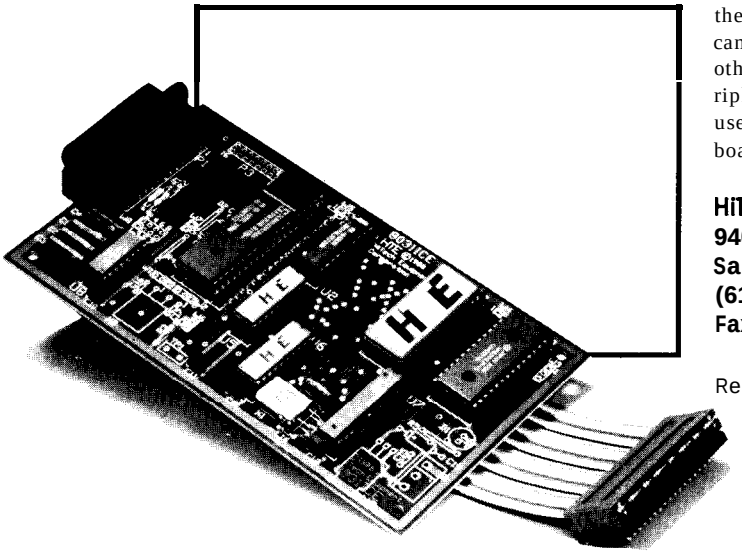
Write letters of praise, condemnation, or suggestion to the editors of Circuit Cellar INK at:

Circuit Cellar INK
Letters to the Editor
4 Park Street
Vernon, CT 06066

FAX: (203) 872-2204
Circuit Cellar BBS: "editor'

In Issue #19, the price of the PSpice Evalution Package was listed as $70. The actual price is $75.

LOW-COST MICROPROCESSOR EMULATORS



HiTech Equipment Corporation has announced the addition of two low-cost microprocessor emulator products to its DryICE family of 8051 emulators. The 80511CE-65 supports the Philips/Signetics 80C652 microprocessor, and the 8051ICE-FA supports the Intel 80C51FA. The 80C652 is a standard 8051 with 256 bytes of internal RAM and an interface to the I²C bus. The 80C51FA is an 8032 with a programmable counter array peripheral added. Both emulators include the SuperMon firmware, which features an on-line assembler and a real-time execute-to-breakpoint command, as well as the standard DryICE program execution and memory interrogation (display, substitute, and fill) commands.

The assembled and tested boards feature emulation cables that plug into the user's target system in place of the target microprocessor. Commands are sent to the DryICE serially through a cable connected to the user's terminal or PC COM port. Terminal emulation software is all that is needed to begin program debugging on a PC. The DryICE accepts user code in Intel hex format.

The DryICE firmware communicates to the PC through an on-board external memory-mapped UART featuring automatic data rate detection. The only requirements on the user are where the code starts and the reservation of six bytes of stack space for the monitor's use. One interrupt must be reserved for the monitor, but this assignment can be dynamically altered. All other 80C652 or 80C51FA peripherals are available to the user. Each of the emulator boards is priced at $329.00.

HiTech Equipment Corp.
9400 Activity Rd.
San Diego, CA 92126
(619) 566-1892
Fax: (6 19) 530- 1458

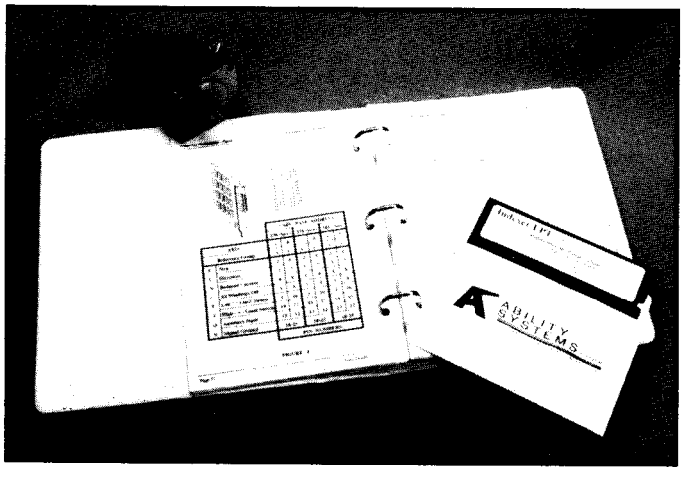
Reader Service #500

LOW-COST MOTION CONTROLLER USES PRINTER PORT

The Indexer LPT from Ability Systems Corporation converts an ordinary IBM-compatible printer port into a multiaxis stepper motor indexer. Each printer port provides sufficient input and output to control two axes of motion. The Indexer LPT supports up to three printer ports, totalling six axes of motion.

Signals for each axis consist of TTL-level outputs controlling "step," "reduced current," and "all windings off." Two limit switches per axis may be wired directly to the printer connector. One auxiliary TTL-level input per axis is provided to allow for additional system sensing. Limit-switch closures automatically arrest motion.

The Indexer LPT software loads as an MSDOS device driver and behaves like a disk file. It is compatible with virtually every programming language, including BASIC, C, Pascal, and even DOS batch files. Since all communication with the Indexer LPT occurs through the DOS device interface, ordinary ASCII files containing the Indexer LPT commands can be quickly constructed with a text editor or word processor, and used to control motion by simply using the DOS `COPY` command.

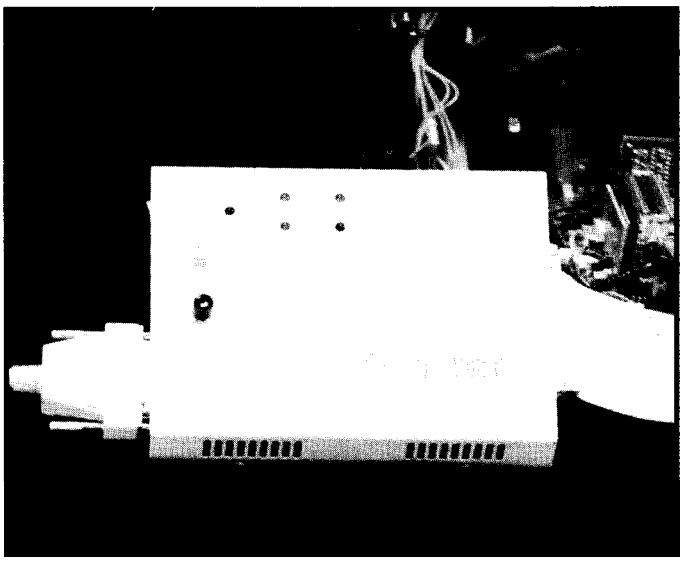


Version 2.0 includes linear interpolation in up to six simultaneous axes, rapid traversal, vector velocity control, and circular interpolation. New features such as a "feed hold" and simplified discrete output specifically target the machine tool automation market. A menu-driven diagnosis program simplifies initial installation and provides a convenient platform to exercise motion hardware. A 117-page instruction manual is included. The Indexer LPT sells for $249.00.

Ability Systems Corp.
1422 Arnold Ave.
Roslyn, PA 1900 1
(215) 657-4338
Fax: (215) 657-7815

Reader Service #501

VERSATILE 805 1 IN-CIRCUIT EMULATOR



A **flexible, easy-to-use in-circuit emulator for the 8051 family of microcontrollers is available from MetaLink Corporation. The iceMASTER,** a 7" x 5.5" x 1" unit, connects any PC (DOS or OS/2) to the target processor through a high-speed (115.2 kbps) serial interface with a standard RS232 cable. Modes of operation include: single chip, microprocessor, watchdog timer enabled, and **DMA active. Two models are available: the Model 200 is the basic emulator; the Model 400 adds a 4K trace buffer, performance analyzer, and full watchdog timer support.**

An advanced interface provides user-configurable color windows that can be sized, moved, highlighted, added, or removed. The main screen windows display registers, bit memory, stack, internal and external data memory, source program, watch, and system status. Pull-down and pop-up menus and user-assignable function/hot keys are available. The contents of any memory space may be viewed from the appropriate window. An on-tine context-sensitive help system provides assistance from any point in the work session.

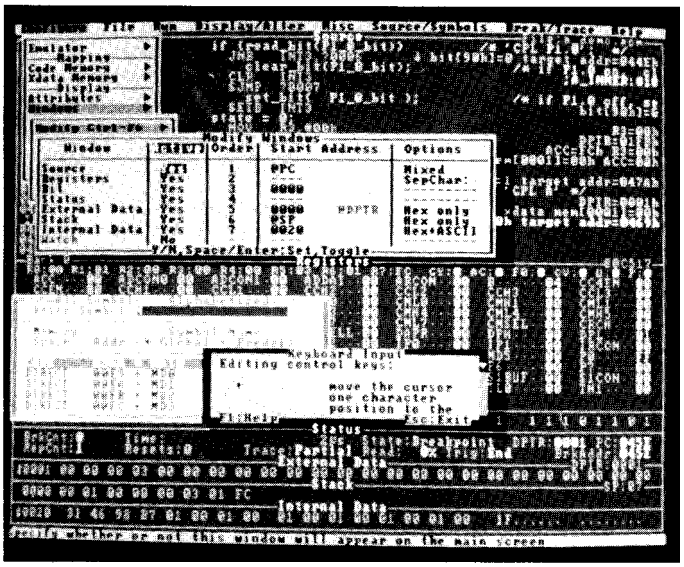
Access **to as many as 128K break and 64K trace triggers is available. These triggers, both simple and complex, can be enabled, disabled, set, or cleared. Simple triggers are based on code or external data addresses or address ranges. Complex triggers are based on code, direct, or bit address; opcode value or class; or immediate operand.**

The 4K-frame trace buffer captures data in real time. Trace information consists of address and data bus values, DMA activity, and user-selectable probe clips. Trace buffer data can be viewed through several display filters, including raw hex, disassembled instructions, instructions mixed with source statements, or source only. The trace can be triggered to begin capturing data on all instructions leading up to, around, or following a breakpoint. An integrated search mechanism allows the location of any label, source line number, or address in the trace buffer, in either the backward or forward direction.

A flexible, accurate, performance analyzer features a resolution of less than 6 microseconds. The time to execute specific portions of a program can be monitored. Up to 15 memory areas, based on code address, module, line number, or label range, can be defined and analyzed. Results can be viewed dynamically during emulation or later for a more detailed analysis.

A large variety of file formats is supported, including Archimedes, Avocet, Intel OMF and hex, MetaLink, and Motorola S.

The price of the Model 200 is $1495.00 including cables. The Model 400 sells for $2799.00. A free demo disk is available by calling (800) 6382423.



MetaLink Corp.
P.O. Box 1329
Chandler, AZ 85244- 1329
(602) 926-0797
Fax: (602) 926-1198

Reader Service #502

PC-BASED SIMULATOR FOR ZILOG Z8

PseudoMax 8Z, a PC-based simulator for the Zilog Z8 family of microprocessors, has been introduced by PseudoCorp. The simulator allows the developer to test and debug Z8 programs even before the hardware exists. By means of machine windows, the developer can watch the program execution as the simulator single steps or free runs through the program code. Each register, the stack, I/O ports, and blocks of memory can be monitored.

Ten user-definable screens enable the designer to customize the simulator. Each screen can contain up to 79 machine-specific windows. Three of the ten screens come predefined. Screen 1, the main screen, contains windows for flags, the stack, interrupts, internal RAM, register pointer, and others. Screen 2 contains the port windows, and Screen 3 contains the working registers and working register pairs.

Other features include unlimited breakpoints, memory mapping, and a trace file feature that gives the user the ability to selectively record the simulator session for later analysis.

The introductory price of the simulator is $100.00. The Z8 Cross- Assembler is $50.00 and a Z8 Disassembler is available for $100.00. A Developer Pack, consisting of all three products is $200.00.

PseudoCorp
716 Thimble Shoals Blvd., Suite E
Newport News, VA 23606
(804) 873- 1947
Fax: (804) 873-2154

Reader Service #503

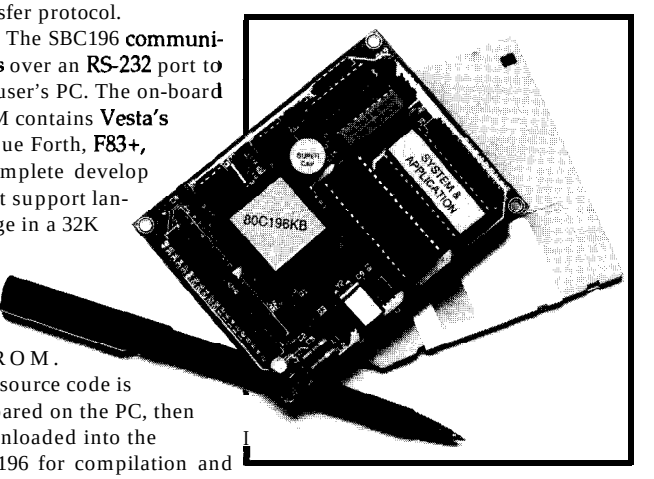
SINGLE-BOARD COMPUTER FOR INSTRUMENT CONTROL

A high-performance 80C196 single-board computer has been announced by Vesta Technology Inc. The SBC196 is a 3" x 4" board designed for low-power applications. It draws only 80 mA while running, 20 mA while idling (most of the time), and less than 1 mA in the sleep mode. A single 5-VDC supply supports all on-board functions. The board is equipped with a "supercap"-backed real-time clock that can awaken the SBC196 from the sleep mode at intervals varying from 100 times per second to once per year. External events can also awaken the SBC196.

The SBC196 features a 12-MHz clock and a more powerful instruction set which runs several times faster than a comparable 8051. Internal resources include timers, high-speed event capture I/O, PWM output, watchdog timer, A/D converter, and a serial channel.

The SBC196 supports eight 10-bit A/D conversion channels. Additionally, an EEPROM (128- to 2K-byte serial device) is on-board to store the calibration factors commonly associated with transducer applications. An on-board-generated ±9-VDC supply is available during run and idle modes, but is turned off during sleep intervals to conserve power. A small peripheral board attached to the I²C bus supports a 4 x 40 alphanumeric LCD display, 8 x 8 keypad, and multitone piezoelectric beeper. Also available is a low-speed, low-power RF modem for remote data acquisition and control applications.

The SBCl96 provides direct access to the 80C196 data bus with a decoded peripheral select ready to connect bus-oriented ICs. The SBC196 also supports two serial bus protocols: Multiwire (NSC) and I²C (Philips). Attachment to either serial bus is simple due to embedded software support and the noise-immune serial data transfer protocol.



The SBC196 communicates over an RS-232 port to the user's PC. The on-board ROM contains Vesta's unique Forth, F83+, a complete development support language in a 32K EPROM. The source code is prepared on the PC, then downloaded into the SBC196 for compilation and testing. Complete debug resources are at the programmer's disposal. The SBC196 takes 80K of usable code and data space from the 80C196 in which the user may write applications as large as 48K (code and data) in addition to the 32K Forth kernel. After the program is running correctly, the compiled code is uploaded to a file and stored on the PC, ready for burning into an autostart EPROM as the finished application. The SBC196 sells for $169 quantity one, $99.00 quantity 100.

Vesta Technology, Inc.
7 100 W. 44th Ave., Suite 101 -Wheatridge, CO 80033
(303) 422-8088 — Fax: (303) 422-9800

Reader Service #504

16-MHZ 80C186 CPU ON STD BUS

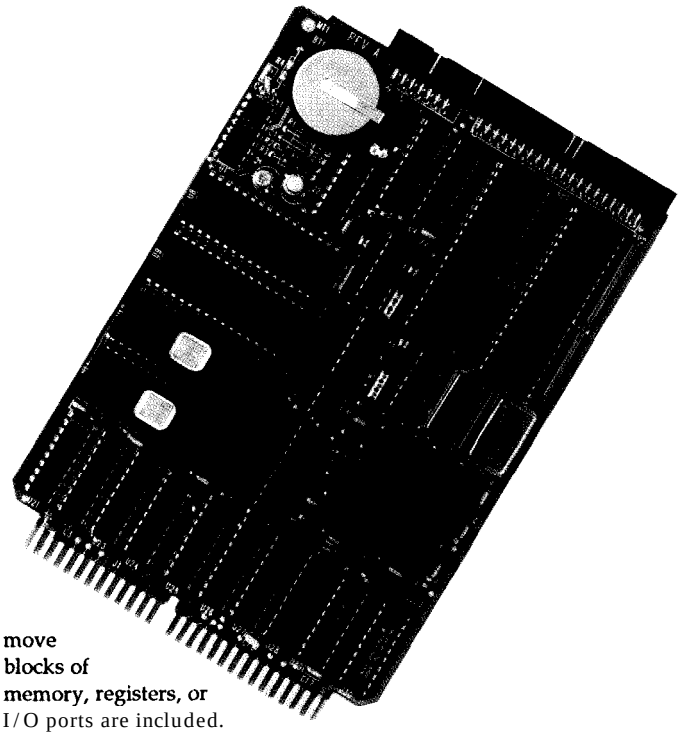
A board-level computer with integrated I/O for STD Bus has been announced by Cubit. The Model 8650 features the Intel 80C186 microprocessor operating at 16 MHz. It is implemented on the 16-bit STD Bus, which is also compatible with 8-bit boards. A development environment based on Borland's Turbo C++ is available.

Two battery-backed 128K x 8 CMOS static RAM chips provide a total of 256K of on-board RAM. Two ROM sockets will accept 1-megabit EPROMs or flash EPROMs for a total of 256K bytes. On-board memory is accessed with full 16-bit operation, while off-board memory and I/O are treated as 8-bit and/or 16-bit, depending on required format.

Two RS-232 serial ports and four 8-bit parallel ports are provided by two 8256 serial/parallel I/O chips. Five 8-bit programmable timer/counters are included in each of the 8256's for a total of ten. Two of these in each chip can be cascaded to form two 16-bit timer/counters. This is in addition to the timer/counters integrated into the CPU chip.

A clock/calendar chip provides the date and time in 12- or 24-hour format. A power fail detect circuit generates a nonmaskable interrupt when the voltage level falls below the minimum operating voltage, permitting the processor to store critical data. Both the RAM and the clock/calendar chip are battery backed with a replaceable lithium battery.

The Model 8650 includes two 27C512 debug firmware EPROMs to assist with program development. It may also be incorporated into the application program to provide initialization of the board and act as a service tool. The monitor links the board to a PC or other RS-232 device. Code can be downloaded from the PC in Intel hex format and exercised in RAM. Standard monitor functions to examine, modify, or move blocks of memory, registers, or I/O ports are included.



The Model 8650 sells for $620.00 in single quantities.

Cubit
340 Pioneer Way — Mountain View, CA 94041-1577
(415) 962-8237 • Fax: (415) 965-9355 Reader Service #505

UNIVERSAL DEVICE PROGRAMMING WORKSTATION

A software-controlled 40-pin universal device programming workstation has been announced by Xeltek. The SUPERPRO interfaces with an IBM PC/XT/AT/386 or compatible to provide reliable, fast programming with Normal, Intelligent, Interactive, and Quick Pulse algorithms. The 10.4" x 6.9" x 1" unit incorporates a high-speed parallel interface, and features a 40-pin ZIF socket that accommodates 0.300" to 0.600" spacing.

The SUPERPRO supports most (E)EPROMs, CMOS EPROMs, and Flash EPROMs of 24/28/32/40 pins, bipolar PROMs, PALs, EPLDs, EEPLDs, and GALs. It supports the Intel 87xx and Signetics 87Cxx families of microcontrollers and serves as an IC tester for TTL and CMOS logic as well as DRAMs and SRAMs.

The unit accepts standard file formats including JEDEC, Intel Extended Hex, Motorola S, Tektronix Hex, and binary. It manages 16- and 32-bit word splits and supports most compilers in JEDEC format. The SUPERPRO features test vector capability and multiarray fuse map editor. User-definable patterns are available for IC testing.

A comprehensive, up datable software library environment is included in the Library Operated Programming System (LOPS). AU programming algorithms and device footprints are contained in the library for different device types. Menu-driven screens and help messages guide users through operation without the need to refer to a manual. Its macro capability allows users to automate any programming procedures by typing a single keystroke.

An optional Library Generator allows the user to update the software library by typing in the necessary data specifications and algorithms. This feature makes the LOPS system software unlimited in terms of programming. The SUPERPRO is available beginning at $795 for the basic unit.

Xeltek
764 San Aleso Ave.
Sunnyvale, CA 94086
(408) 745-7974
Fax: (408) 745- 1401

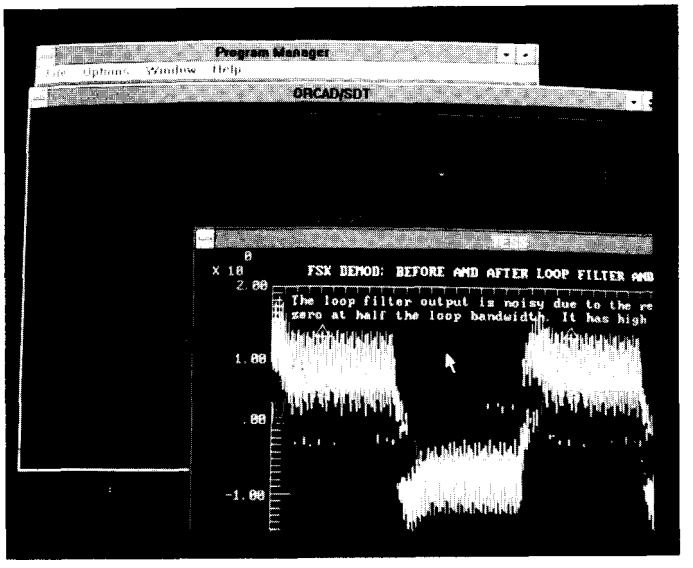
Reader Service #506

VERSATILE ANALOG/DIGITAL SYSTEM SIMULATOR

The TESS (Transient Electronic System Simulator) block diagram simulator from Tesoft simulates mixed analog and digital systems at the block level. Unlike transistor-level simulators, TESS runs at speeds fast enough to model communications systems. It can be used to analyze modems, radios, signal processors, and control systems to observe the effects of nonlinearities, bandlimiting, quantization, and adjacent channels. Loop stability tests, BER measurements, FFT spectrum analysis, filter synthesis, Gaussian noise, and cost/performance tradeoffs can also be performed. Circuits with up to 800 blocks and 1000 nodes can be run, and simulation size is limited only by disk space. Support is included for a math coprocessor as well as macros and subcircuit libraries. Interfaces to OrCAD and P-CAD are included.

The TESS Model library contains over 55 blocks for building communications equipment and other electronic systems. These include filters, mixers, VCO, A/D and D/A convertors, digital logic models, phase meter, and many others. Subcircuits can be used to create new models from existing ones. An optional menu-driven model generator, **MODGEN, can be used to** define a custom model using Microsoft FORTRAN; no special modeling languages or linkage coding is required. Models can be as small as one line or as complex as 100K of code. TESS Version 1.1 adds 24 new filters, high-quality graphics with interfaces to publishing software and plotting devices, memory management to support huge model libraries and run other tools, and enhancement of many existing features.

TESS sells for $695. Options include symbol libraries for OrCAD and P-CAD schematic capture ($195 and $295). There is also a



MODGEN model generator ($495) which lets users add new model blocks to TESS. A demo disk package is also available.

Tesoft
205 Crossing Creek Ct. . Roswell, GA 30076
(404) 751-9785 . Fax: **(404)** 664-5817

Reader Service #507

FEATURE ARTICLES

CCINK DESIGN CONTEST WINNER

A MIDI-Controlled Sampled-Sound Player

This article describes a simple circuit that connects to a MIDI bus and replays sampled sounds when so commanded. The primary application is as a MIDI drum synthesizer for the authors' home studios, and this application drove many of the design decisions. However, digital percussion is only one of the things the unit can be used for. Any audio waveform that can be digitally sampled or otherwise reduced to digital form can be played back by this unit, including other musical instruments, human voices, and sound effects.

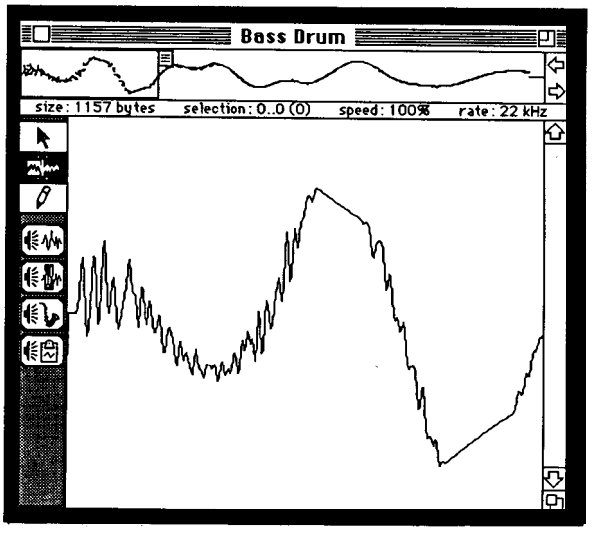
The unit, as described in this article, is capable of storing about six seconds' worth of sound. The six seconds can be arbitrarily partitioned among the notes-that is, each note can last a different amount of time. Each note lasts its own fixed amount of time. Up to four notes can be active at any one time, and each of them may be played at different volumes (using MIDI velocity commands in the range l-127). The sample / playback rate we use is almost exactly 10 kHz, which provides fidelity more than adequate for rock 'n roll.

In our development system, sounds are captured on a Macintosh using the MacRecorder digitizer and sound editing facilities (from Farallon Inc.). This system was selected primarily because we already had the hardware and software when we decided to design the MIDI unit. Sounds are transferred to a PC for linking into the embedded program and downloading to the unit's EPROM.

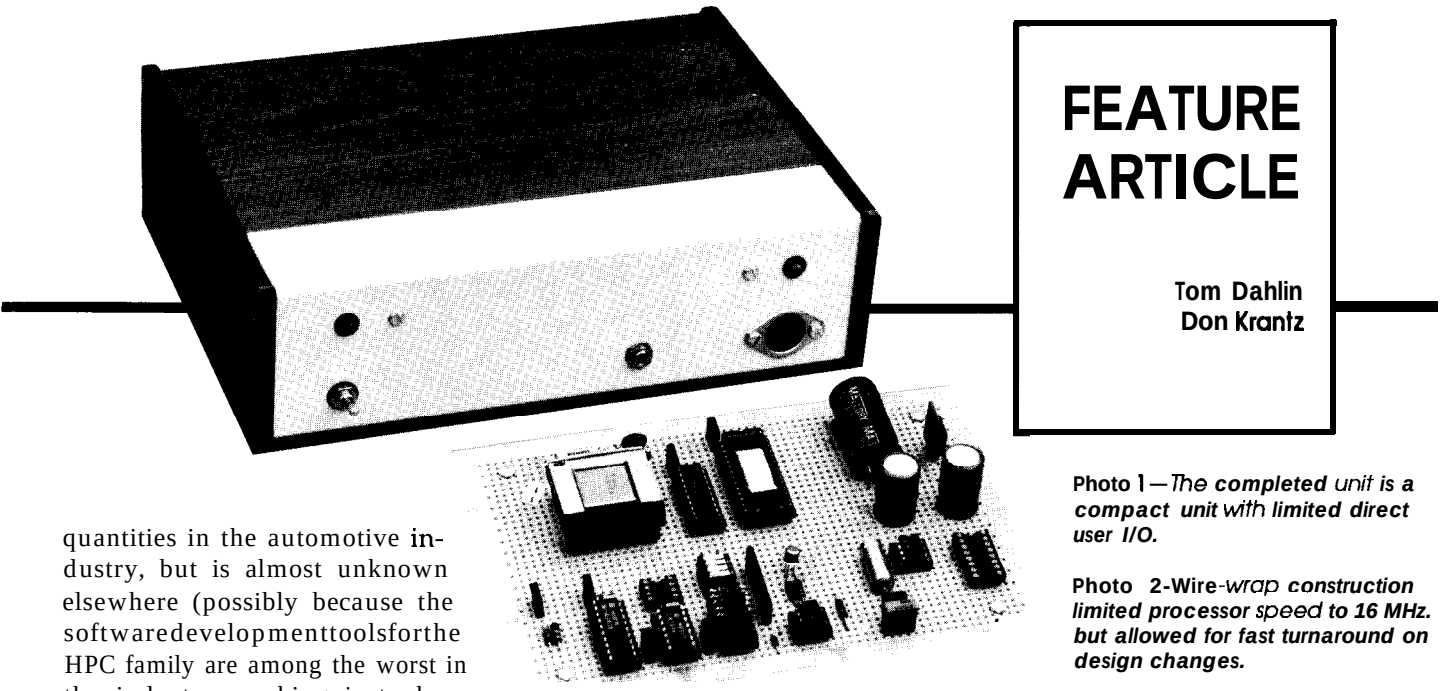
AN UNUSUAL PROCESSOR

Figure1 shows themajorelements of the unit in block **diagram** form while Figure2 isaschematic. As you can see, the system uses very few parts. A major design goal was to design for low cost, both material and labor, and we believe that we reached this goal.

The processor is the National Semiconductor HPC 46003 (see page 18). This processor is used in large



An example of a sampled sound. We used a Macintosh computer and MacRecorder hardware to sample the drum sounds. Thisscreen is from the 'Sound Wave' program which allows you to edit, resample. and filter sampled data. The sampler hardware has a resolution of 8 bits and a maximum sample rate of 22k samples per second.



FEATURE ARTICLE

Tom Dahlin
Don Krantz

Photo 1—*The completed unit is a compact unit with limited direct user I/O.*

Photo 2-Wire-wrap construction limited processor speed to 16 MHz. but allowed for fast turnaround on design changes.

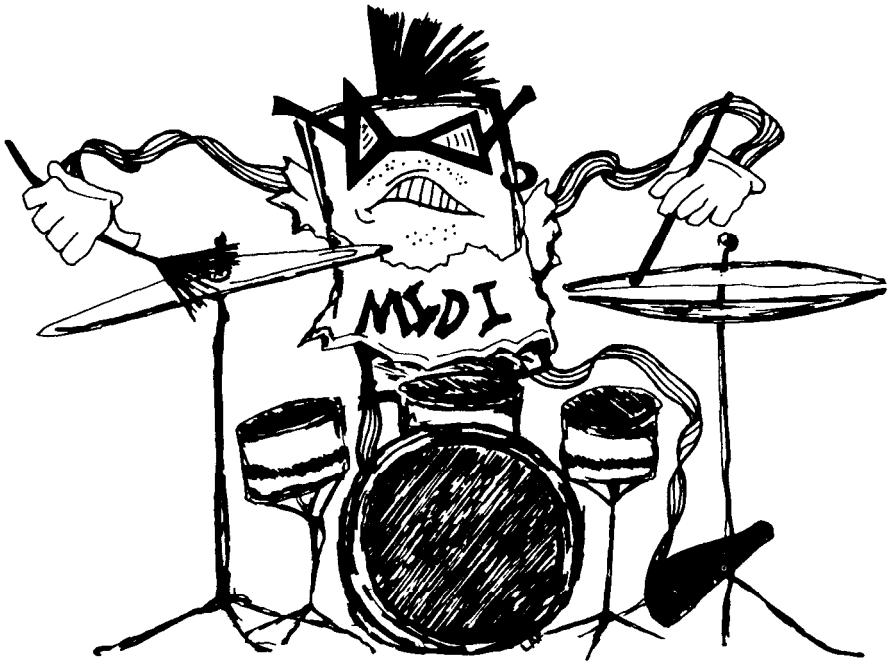
quantities in the automotive industry, but is almost unknown elsewhere (possibly because the softwaredevelopmenttoolsforthe HPC family are among the worst in the industry, ranking just above loading **binary** machine instructions on front-panel switches).

The HPC is a 16-bit machine that can run with either an 8-bit or 16-bit external data bus. We chose to use the 8-bit external bus to save an EPROM and address latch. Because the CPU must double fetch instructions in its 8-bit mode, it runs slower than it would if we had used the 16-bit mode. The early chips in the HPC family (this one included) also cannot fetch 16-bit data items from RAM or ROM when in 8-bit mode; 16-bit operands must be explicitly double fetched and reconstructed in software. Later members of the HPC family are reputed to cure this problem.

The HPC is available in a 30-MHz version, with many instructions executing in about four clock cycles (the clockisone-half the crystal frequency). Al though the MIDI unit was designed to run at 32 MHz (the National field engineer says the 30-MHz part works fine at 32 MHz), we opted to run the prototypeat 16 MHzbecause we wire-wrapped the prototype.

We selected the HPC for this project after the usual rigorous tradeoff study and analysis phase. We had used the HPC in an earlier project where we needed the speed and the input capture registers. Because of the earlier project, we had a primitive in-circuit emulator, an assembler/linker, and a rudimentary C compiler.

The HPC has many good features, but it also has some bad ones. On the positive side, the part is very fast and has a decent instruction set (it is RISC-like and reasonably orthogonal). At 30-MHz it is fast enough to do simple audio digital signal processing. The hardware design is simple and straightforward, very much like the 68*xx* family. On the negative side, the part is not a mainstream processor, and it is unlikely that it will make large inroads in the 8031 or 68HC11 camps. Because of this, third-party tools are nonexistent. The local National field engineers made heroic efforts to support us, but they didn't have all the answers. More than once, we had to hold up the project while the California development system folks figured out what was wrong.



MIDI DATA INTERFACE

The MIDI spec calls for optical isolation on the receiver end of a MIDI connection. We used an HP2730 optoisolator connected directly to the serial input port (pin I6) of the HPC. Our application does not require the ability to transmit on MIDI, but this capability can be easily added by connecting a 74LS driver to the HPC's

serial output pin. The 32-kbps MIDI clock is developed internally to the HPC by dividing its 16-MHz clock.

Our unit is designed to listen to a single MIDI channel (16 are defined by the MIDI spec). A DIP switch selects MIDI channel assignment.

The system has one visual indicator, an LED designed to show activity on the MIDI channel. Originally, this LED was supposed to blink only when traffic was received on the unit's own channel. This was later changed so that it changes state whenever any MIDI traffic is detected. This allows the user to monitor for MIDI-overload conditions.

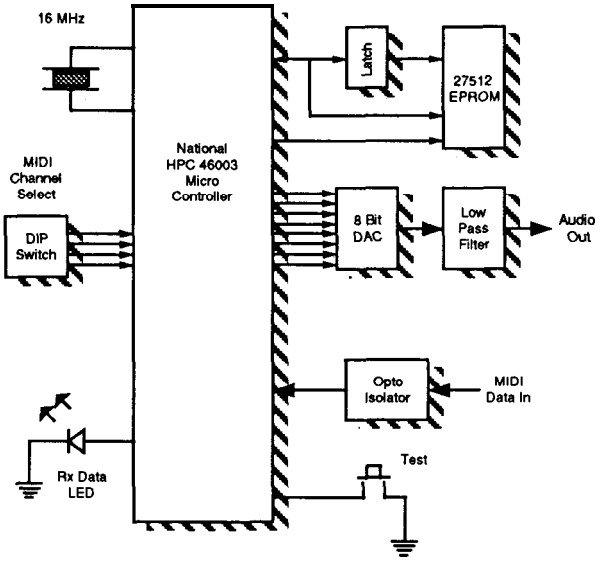


Figure 1 -Bloc&diagram of *the MIDIplayback unit. The system is built around the NS HPC46003 microcontroller. MIDI data is optoisolated and run into the HPC's serial port. Software decodes the incoming MIDI 'note on' command and plays back a sound through the B-bit A/D converter. A lo w-pass filter cuts off high-frequency noise.*

DIGITAL-TO-ANALOG CONVERTER

The front end of the analog audio portion of the unit is an inexpensive 8-bit DAC, the National DAC0830. The DAC is operated in a flow-through mode, where data on its input pins are immediately converted and presented on its output as a differential current. An LM324 op-amp converts this current into voltage. An LM385 is used for a stable voltage reference.

Part of the unit's LM324 is configured as a third-order low-pass filter with a Bessel response. The cutoff frequency is set at 4.9 kHz, which is about the correct Nyquist frequency for a 10-kHz sample rate. R1, R2, R3, Cl, and C2 set the cutoff frequency, and are mounted on a DIP header for easy substitution.

The +5-volt digital power and the positive half of the audio section's power is provided by a simple linear regulator. An Intersil ICL7660 inverts the +5-volt supply to provide -5 volts for the analog section.

INTO THE HEART OF MIDI

The software is the heart of the MIDI unit. It is written mostly in HPC C, witha small portion written in HPC assembly language. The software has three main components: a MIDI data input section, a MIDI data parser, and an analog output driver. In addition to these components, there are a few

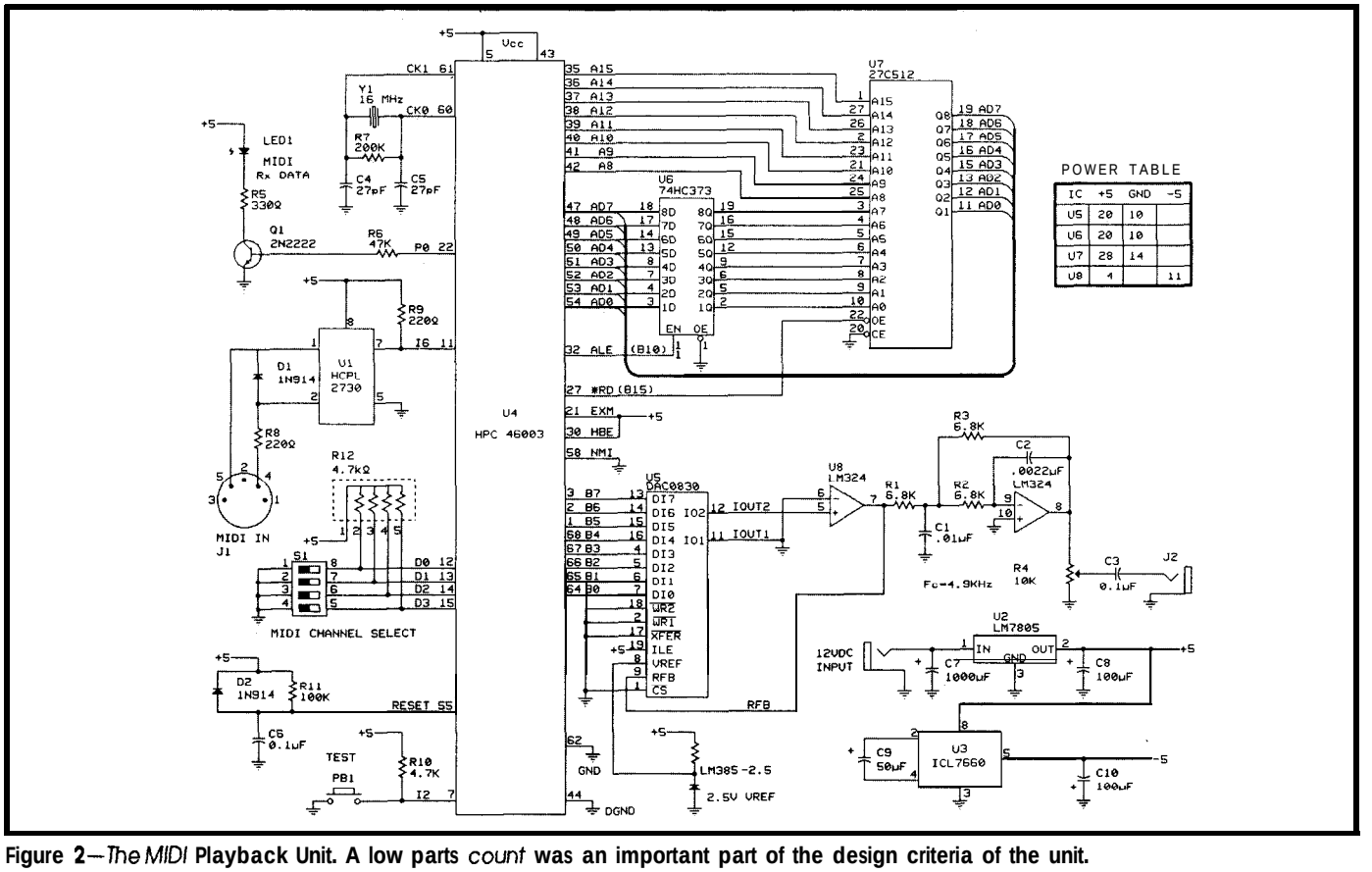


Figure 2—*The MIDI* Playback Unit. A low parts *count* was an important part of the design criteria of the unit.

Note	Octave	Number	Drum
C	-2	36	bass drum
C#	-2	37	rimshot
D	-2	38	snare
D#	-2	39	hand clap
E	-2	40	snare
F	-2	41	low tom
F#	-2	42	closed hi hat
G	-2	43	low tom
G#	-2	44	open hi hat 2
A	-2	45	mid tom
A#	-2	46	open hi hat 1
B	-2	47	mid tom
C	-1	48	high tom
C#	-1	49	crash cymbal
D	-1	50	high tom
D#	-1	51	ride cymbal
F#	-1	54	tambourine
G#	-1	56	cowbell

'Note" corresponds to piano keyboard note name
'Octave" is relative to middle C
'Number" is MIDI note number
'Drum" is the drum assigned to the note
Observe that there are duplicate entries for several drums

Table 1—*The MIDI* **play-***buck unit translates* MIDI *'note " data to drum type* **for playback. This** *is* **common among MIDI** *synthesizers and drum machines.*

```
/*------- USART.C --------
General-purpose UART interface. Transmitter disabled in ISR
 because it's not used in MIDI application. ------*/

/*-------------------------
MACRO to test if transmitter is currently enabled.
-------------------------*/
#define tx_enabled() \
     (UART_INTERRUPT_CLOCK_REGISTER & 0x01)

/*-------------------------
MACRO to disable transmitter interrupt
-------------------------*/
#define disable_tx0 \
     (UART_INTERRUPT_CLOCK_REGISTER &= 0xFE)

/*-------------------------
MACRO to enable transmitter interrupt
-------------------------*/
#define enable_tx() \
     (UART_INTERRUPT_CLOCK_REGISTER |= 0x01)

/*-------------------------
MACRO to test if a received character is available
-------------------------*/
#define rx_buffer_full() \
     (UART_CONTROL_STATUS_REGISTER & 0x02)

/*-------------------------
MACRO to test if the transmitter buffer is empty
-------------------------*/
#define tx_buffer_empty() \
     (UART_CONTROL_STATUS_REGISTER 6 0x01)

/*-------------------------
MACRO to output a string to the UART - construction-makes
placement a bit dicey if a semicolon is placed following the
macro expansion. Does not append a newline; does not expand
newline '\n' to CRLF.
-------------------------*/
#define  puts( s)\
     {   cptr = s; \
         while(*cptr)  putc(*(cptr++)); }

/*-------------------------
MACRO to test if a received character is available.
-------------------------*/
```

(continued)

listing 1 -General-purpose *UART interface module*

NATIONAL SEMICONDUCTOR HPC 46003

The HPC 46003 is a member of National's family of high-performance microcontrollers. There are several members of this family, each with the same 16-bit CPU core but differing in the I/O and memory options provided. The part is unique In that it can operate with either an 8- or 16-bit-wide memory bus. This provides the designer with the option to use an 8-bit bus for low cost or reduced real estate, or a 16-bit bus for maximum speed.

KEY FEATURES

- 30-MHz parts available-the register instructions execute in as little as 134 ns
- CMOS fabrication—low power, tolerant of input voltage variations
- 16-bit architecture-internal and external, with additional 8-bit instructions and addressing modes
- 1 &bit data bus, ALU, and registers
- 64K bytes of direct memory addressing
- Most opcodes are a single byte
- 16- x 16-bit multiply and 32- x 16-bit divide instructions
- 8 vectored interrupts
- 8 timers with up to 8 outputs
- 4 Input capture registers—on-the-fly programmable edge detection and interrupts
- 256 bytes of byte- and word-addressable internal RAM
- Built-in watchdog timer
- Full-duplex UART with programmable transmit and receive data rates
- Microwire serial bus (Microwire is a three-wire serial interconnect bus for communicating with peripheral chips such as A/D and D/A converters)

miscellaneous minor components whose functions should be self-evident. *[Editor's Note: Complete software for this article is available from the Circuit Cellar BBS and Software On Disk #20. See page 91 for downloading and ordering information.]*

The MIDI input section uses the HPC's internal UART in a fully interrupt-driven mode. As data is received from the 32-kbps MIDI bus, it is stored in a circular receiver FIFO. The logic is in the file `UART.C`, shown in Listing 1. This file was taken in whole from a

```
#define rx_rdy() \
     (rx_gozinta  != rx_gozouta)
/*-------------------------
FUNCTION  putc( c)

ABSTRACT: Outputs the character 'c' to the serial UART.
Buffered interrupt-driven output. Not to be called from
interrupt service routines. Note: If tx buffer is full when
called,  this function does a busy wait.

RETURNS: void
-------------------------*/

putc(c)
   char c;
{
     while  (((tx_gozinta - 1) % TX_FIFO) ==
             (tx_gozouta % TX-FIFO))

     disable_interrupt(UART_INTERRUPT);
     if (tx_enabled())
     {
         tx_buffer[tx_gozinta] = c;
         tx_gozinta = (++tx_gozinta) %  TX-FIFO;
     }
     else
     {
         TRANSMIT_CHARACTER_BUFFER = c;
         enable_tx();
     }
     enable_interrupt(UART_INTERRUPT);
}
/*-------------------------
FUNCTION getc()

ABSTRACT:  returns a character from the UART. Does a busy
wait if no character is available.

RETURNS: char
-------------------------*/
NOLOCAL char
getc()

     static char  temp;

     while (!rx_rdy())
         if ((PORT_I_INPUT-REGISTER & 0x04) == 0)
             return(0);
     toggle-port _p_bit(0);
     temp = rx_buffer[rx_gozouta];
     rx_gozouta =  (++rx_gozouta) % RX-FIFO;
     return(temp);

/*-------------------------
FUNCTION uart_int()

ABSTRACT:  interrupt service routine for the UART interrupt.

RETURNS: void
-------------------------

INTERRUPT6 NOLOCAL void
uart_int()
{
#ifdef TRANSMITS_TOO
     if (tx_buffer_empty())
     {
         if (tx gozinta == tx_gozouta)
             disable  tx();
         else
         {
             TRANSMIT_CHARACTER_BUFFER =
                 tx_buffer[tx_gozouta];
             tx_gozouta = (++tx_gozouta) %  TX-FIFO;
         }
     }
     if (rx_buffer_full())
#endif

         rx_buffer[ rx_gozinta]=  RECEIVE_CHARACTER_BUFFER;
         rx_gozinta = (++rx_gozinta) % RX-FIFO;
         if (rx_gozinta == rx_gozouta)
             rx_gozouta = (++rx_gozouta) % RX_FIFO;
     }
```

sting 1 —*continued*

previous application with the exception of the serial transmitter which is not needed for the drum unit.

The MIDI parser is in the program's main loop in the file MAIN. C, shown in Listing 2. It does not implement the entire MIDI standard, but only as much as makes sense for a drum kit. This does include "running status," a data compression scheme used by many MIDI bus masters. Incoming MIDI commands are decoded by the parser, which looks for note-on and note-off commands addressed to the unit's MIDI channel number. Table 1 lists the notes recognized by the unit. As valid commands are received and parsed, they are queued for processing by the output driver.

DRIVING ANALOG OUTPUT

The "guts" of the program is the analog output driver. This is part of the program that actually produces the output waveform. It is rigidly cyclic, being connected directly to the 10-kHz heartbeat interrupt (timer interrupt 5). The code is in the file

```
int i, j;                       /* temporaries */
unsigned char note, velocity;
int my note selected = 0;       /* for running status */
int my_channel = 0;             /* for running status */
int was_note_off = 0;           /* for running status */
NOLOCAL void
main()

    drum1_ctr = 0; /* these are declared in an assembly */
    drum2_ctr = 0; /* language package, so they don't */
    drum3_ctr = 0; /* get initialized by the compiler. */
    drum4_ctr = 0;
    configure_machine(); /* initialize the HPC hardware */
    /* read MIDI channel number */
    my-channel = (PORT-D) & 0x0F;
    /* power-on delay */
    for (i = 0; i < 32000; i++)
        ,
    /* go into overdrive (Set only 1 wait state for ROM) */
    PSW REGISTER = 0x18;
    /* start listening to MIDI port */
    enable_ interrupt( UART_INTERRUPT );
    for ( ; ; )
    {
        j = getc();
        /* is test switch pressed? */
        if ((PORT_I_ INPUT-REGISTER & 0x04) == 0)
            test();
        else if (!(j & 0x80)) /* this isn't a status byte*/
        {
            if (my_note-selected)
            {
                note = j;
                velocity = getc();
                if (was_note_off)
                    note_off(note, velocity);
                else
                     note_on(note, velocity);

        }
        else if ((j & 0xF0) == 0xF0) /* system ex, */
                 /* system common or real-time */
```

(continued)

Listing 2—*The MIDI parser is in the program's main loop shown here.*

```
        {
            switch  (j & 0x0F)
            {
                case 0:     /* system exclusive */
                    while (getc() != 0xF7)
                        ;
                    break;
                case 1:     /* need to eat one data byte */
                case 3:
                    getc();
                    break;
                case 2:     /* need to eat two data bytes */
                    getc();
                    getc();
                    break;
                default:
                    break;
            }
        }
        else if ((j & 0x0F)  !=  my-channel)
        {
            my-note-selected = 0; /* reset running status */
        }
        else  switch (j & 0xF0)
        {
            case 0x80:          /* status is 'note off' */
                my-note_selected = 1;
                was note_off = 1;
                break;
            case 0x90:          /* if status is 'note on' */
                my note selected = 1;
                was note_off = 0;
                break;
            case 0xA0:          /* poly key pressure */
            case 0x30:          /* control change */
            case 0xE0:          /* pitch bend */
            case 0xC0:          /* program change */
            case 0xD0:          /* channel pressure */
                break;
        }
    }
}
```

Listing 2—*continued*

TIMER5.ASM and is shown in Listing 3. It is the only part of the unit's software that is written in assembly language.

Upon receiving a timer interrupt, the driver first loads the DAC with the value calculated during the previous cycle. The one-cycle delay in writing the output prevents jitter caused by the variable processing time in the output calculation. The time variance is attributable to differences in the number of active notes and the processing required for amplitude scaling.

The driver can calculate and mix two or three simultaneous notes using a 16-MHz clock, and four or five notes using a 32-MHz clock. Each note's waveform can be adjusted in amplitude in 127 discrete steps, and each note is temporally independent of the other notes (i.e., it can start and end independently of the others). The multiplication instruction used to compute the volume is a substantial contributor to the total execution time for the output driver routine.

The MIDI parser and the output driver communicate using shared

```
;-------------------------------------------------------------
; TIMER5.ASM
;
; This routine executes at 10 kHz, and constructs the analog
; output waveform.
;-------------------------------------------------------------

; Timer 5 interrupt

      psw = 0xc0:word
      LOCZ = 0x0:word

      .extrn   _dac_value:byte:BASE
      .extrn   _timer:byte:BASE
      .extrn   _timer_temp:byte:BASE

;  ************************************************************
;
; Macro to process each drum
;
      .macro   drum,ptr,ctr,vol,num
      .mloc    endofit

      .sect    bsect,BASE
ptr:  .dsb     2
ctr:  .dsb     2
vol:  .dsb     1
num:  .dsb     1
      .public  num,vol,ctr,ptr
      .endsect

      ifeq     ctr.W,#0x0            ; drum active?
      jp       endofit               ; no,  jump  out

      ld       b,ptr.w               ; load cur signal ptr
      ld       a, [b].b              ; ready to multiply
      ifbit    7,a.b                 ; negative?
      or       a.w,#0xFF00           ; sign extend (if neg)
      mult     a.w,vol.b             ; times volume
      add      _dac_value.w,a.w      ; add to total
      inc      ptr.W                 ; pnt to next signal val
      inc      ctr.w
endofit:
      .endm

;  ************************************************************

      .sect    csect,ROM8
-timer-interrupt:
      .public  _timer_interrupt
      .ipt     5,_timer_interrupt
      push     psw.w
      push     b
      push     a
      push     x
      push     k
      ifbit    5,0x150.B             ; checks for timer int
      jp       cc40
      jmpl     end_of_drums          ; exits if not timer 5

cc40: sbit     7,0x150.B             ; ack timer interrupt 5
      ld       0xe2.b,_dac_value+1.b; write to port B
      ld       _dac_value.w,#0x8000 ; zero out dac value
      add      _timer_temp.w,#0xFFFF; timer temp --;
      ifeq     _timer_temp,#0        ; ticked to zero?
      jp       flick                 ; jumps if a major tick
      jp       dl                    ; exit out if not zero
tick: ld       _timer_temp,#100      ; set for 100 Hz
      add      _timer.w,#1           ; timer++;

; Four instances of macro "drum" defined above

d1:   drum     _drum1_ptr,_drum1_ctr,_drum1_vol,_drum1_num
d2:   drum     _drum2_ptr,_drum2_ctr,_drum2_vol,_drum2_num
d3:   drum     _drum3_ptr,_drum3_ctr,_drum3_vol,_drum3_num
d4:   drum     _drum4_ptr,_drum4_ctr,_drum4_vol,_drum4_num

end_of_drums:
      pop      k
      pop      x
      pop      a
      pop      b
      pop      psw.w
      reti
      .endsect
```

Listing 3—*The analog output driver forms the heart of the program.*

memory. When a note-on command is detected, the parser writes a pointer to the start of the sampled waveform data for that note into shared memory. The parser also writes the number of samples for that note into the same shared memory. The output driver checks shared memory for new data each time it is invoked.

The software was originally written in a version of HPC C that does not support separate compilation. Thus, all of the C code is compiled in a single pass (see the file MIDI.C). Although the C files are broken out separately, they must still be compiled in a single passbecause of data definition visibility. The program compiles quickly enough that there hasn't been an incentive to convert the program to separate compilation (in other words, it ain't broke and we ain't fixin' it).

Additional miscellaneous quick-and-dirty utilities were written to handle and massage the sampled sounds. A Macintosh utility written in ZBasic is used to convert MacRecorder data files to ASCII for transport to the PC. On the PC end, various small programs are used to resample the data files (MacRecorder sampled at 11 kHz; we wanted it at 10 kHz) and to write the output as C source code to be compiled and included in the C program.

REFERENCES

Anderton 86. Craig Anderton. MIDI for Musicians. Amsco Publications. 1986: New York.

Conger 86. Jim Conger. C Program ming for MIDI. M&T Publishing Inc. 1986: Redwood City, CA.

Conger 89. Jim Conger. MIDI Sequencing in C. M&T Publishing Inc. 1989: Redwood City, CA.

Washington 89. Winefred Washington. "A Low-Cost MIDI Sequencer." Circuit Cellar INK. December 1989.

Tom Dahlin is a Software Engineering Specialist at the 3M company in St. Paul, Minn.

Don Krantz is an Engineering Fellow at Alliant Techsystems Inc. (formerly Honeywell Ordnance Division).

IRS

401 Very Useful
402 Moderately Useful
403 Not Useful

CCINK DESIGN CONTEST WINNER

FEATURE ARTICLE

A Portable 8051 -Based DRAM Tester

Designing Maximum Features into Minimum Space

John Wettroth

For a long time I wanted to have a way to check out dynamic RAM chips (DRAMs). Buying $100 worth of surplus "pulled" parts for my PC or a bargain hunting friend is a fairly gut-wrenching experience for a frugal person like myself. Troubleshooting an ailing PC using the "row at a time"technique has always seemed like a time waster, too. At the time I came up with this design (early 1989), DRAM prices were at an all-time high, and though prices have subsided somewhat, the economics of testing DRAMs is still sound.

What I really wanted was a battery-operated portable DRAM tester that I could take to my local electronic swap meet and also have around my lab for general use. I looked around a bit and found that there were sophisticated laboratory instruments with huge price tags and a few unsophisticated, small AC-powered gadgets that weren't particularly cheap. It was also early March and I had been toying with the idea of entering the first CIRCUIT CELLAR INK Design Contest but couldn't think of a good project. I did just enough thinking about this project to convince myself that it would be easy. It turns out that I was wrong-it wasn't easy, but it was fun. I learned a lot about these little devils called DRAMs and it earned me first prize in the cost-effective category!

The final product is housed in a rugged hand-held box; uses an 87C51 microprocessor and a handful of other ICs; and tests 64K, 256K, and 1M DRAMs for data integrity and access time. The user interface is through a 16-character LCD display and a keypad. The hardware and software design is modular to allow changes or improvements to be made easily. It presently doesn't support SIMMs, SIPs, or "by 4"-type devices, but the design does not preclude these. A block diagram is shown in Figure 1.

DRAM BASICS

Before jumping into the design of the tester, it might be helpful to review the operation of a dynamic RAM. What follows is an introduction and should allow you to understand the design of the DRAM tester fully.

The obvious and biggest difference between dynamic RAMs and static RAMs is the dynamic nature of DRAMs, that is, they must be kept active or they will lose their data. This is because their cells are very simple, with the active storage element for a bit of data being a capacitor rather than a flip-flop as in a static RAM. A cell capable of storing one bit in a DRAM requires only one transistor, while a static RAM requires four or often six transistors. This basic density difference has always made DRAMs significantly less expensive

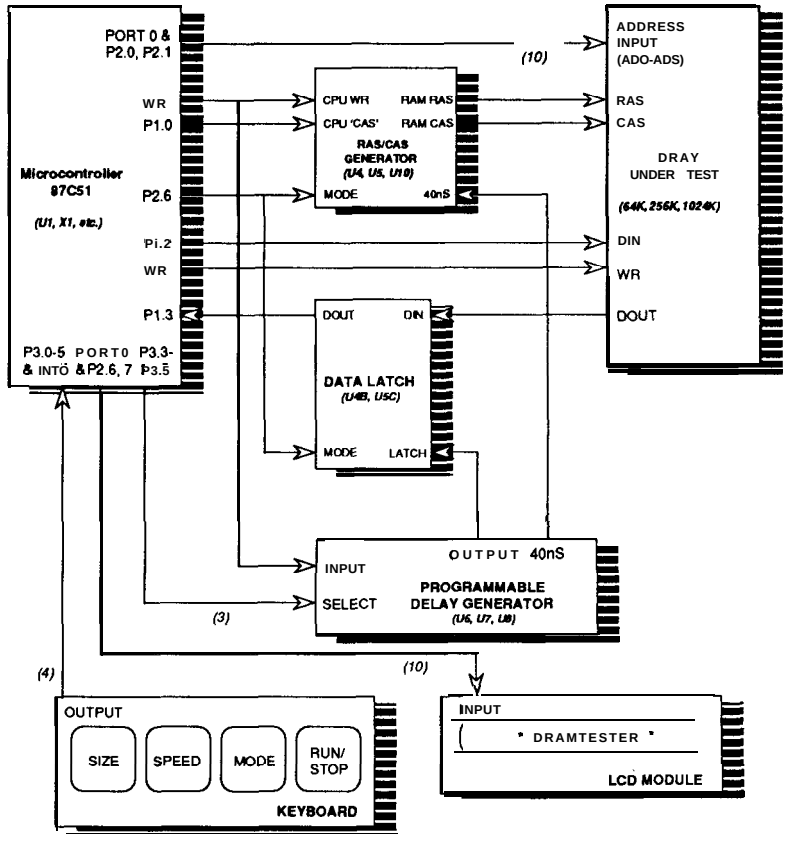
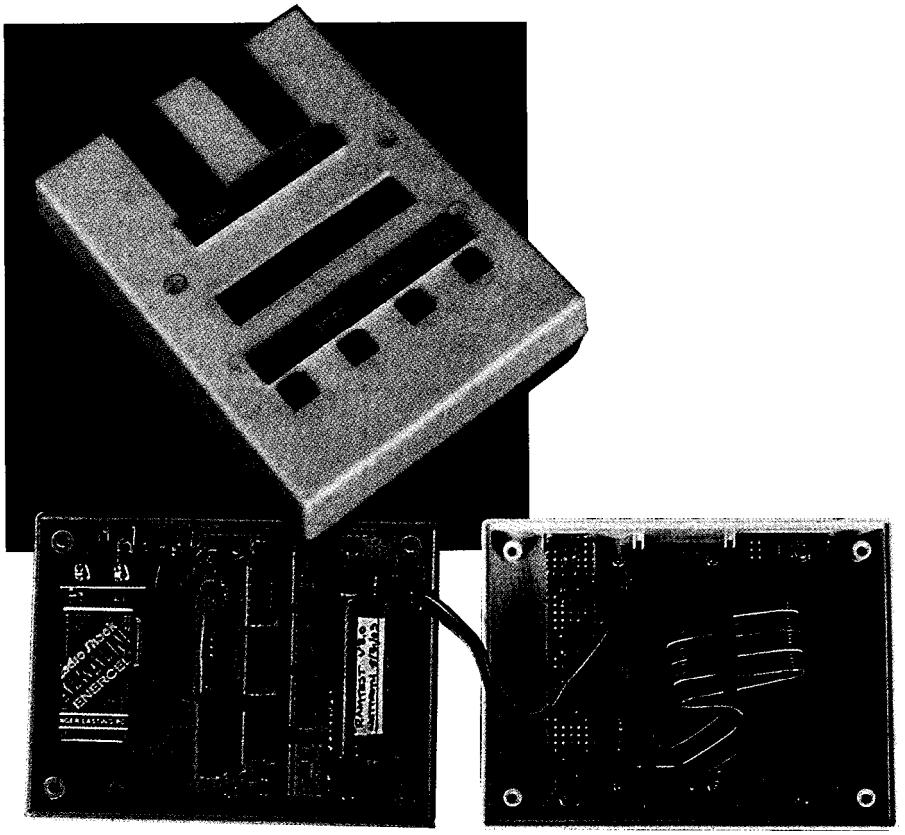


Figure 1 -An *87C5* 1 microcontroller makes up the heart of this cost-effective Design Contest winner.



on a cost-per-bit basis. It's true that there is additional overhead involved in DRAM-based products but the economics for large memory arrays are still overwhelming.

A DRAM-based system has to give up a little CPU overhead or have specialized hardware to go out and "refresh" its DRAMs every few milliseconds or so. This refreshing at its most basic level is a process of reading out and recharging each capacitive storage element. It's apparent that this hassle is worthwhile, witnessed by all the DRAM-based PCs in the world.

There are also several interfacing differences between DRAM and SRAM. Figure 2 is a pinout of a 256K by 1 DRAM; its generic part number would be "41256." DRAMs, like the one shown, are usually configured as single bit slice arrays of a large number such as 64K by 1 or 256K by 1. In order to build an array of 8 bits, you put eight "by 1" devices in parallel, each providing a bit.

Since DRAMs have a large linear address space, and IC pin count is always at a premium, the address bus is usually multiplexed. The multiplexed addressed bus facilitates refreshing as discussed in the next section. The address multiplexing is accomplished by dividing the address space of the DRAM into a row and column address. In this way a 64K part can be accessed using only eight multiplexed address lines (a 64K RAM array can be thought of as a 256-by 256-bit square). This is similar to the multiplexed low-order address and data bus on Intel microprocessors.

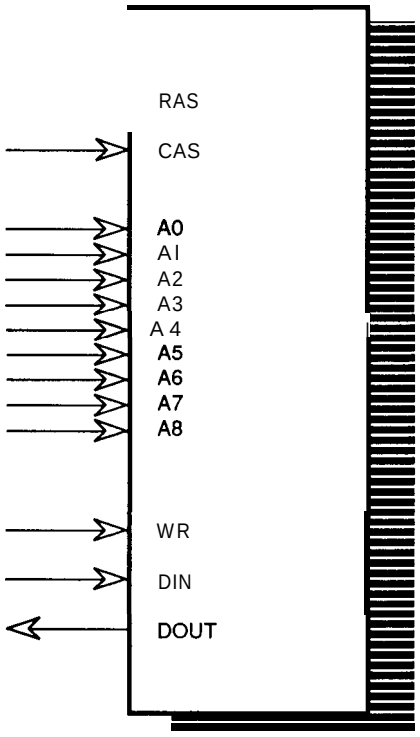


Figure 2-The *DRAM Tester is designed to test parts such as this* 256K *by 1 DRAM. Its generic part number would be '4 1256.'*

In order to complete the multiplex of the row and column address lines, two address strobes are required, one for the row address (RAS) and one for the column address (CAS). The RAS line also doubles as the chip enable pin and the CAS line doubles as the output enable pin. DRAMs also have a data input pin (D), a data output pin (Q), and a write line (W). It's fairly impressive that a 256K DRAM can be packaged in a 16-pin DIP package. The process of addressing a DRAM is to apply the row address to the address inputs, drop the RAS line to strobe it in, apply the column address, and drop the CAS line to completetheaddressselection. Write,Data In, and Data Out are handled appropriately, similar to static RAM.

DRAM timing diagrams can look very confusing, mainly due to the multiplexed address system and refreshing requirements. If you don't get too hung up on the details, it's fairly simple. Figure 3 is a simplified timing diagram for a DRAM read cycle. Accessing a cell in the DRAM consists of placing the row address on the address input and bringing RAS low to strobe in the row address (dotted line 1 in Figure 3). Since the RAS line also acts as the chip enable, it is kept low through the remainder of the cycle. The column address is next placed on the address inputs and the CAS line is brought low (dotted line 2). The addressing is now complete and Q is enabled by CAS (remember that CAS doubles as the output enable pin). Data is picked up off Q (dotted line 3) and CAS and RAS are brought high, ending the cycle (dotted line 4). This process can be repeated after a short "precharge" time (lines 5 and 6).

Writingisaccomplishedasshown in Figure 4, but W and D are set up before CAS falls. This is a simple write called an "early write" since write is low before CAS is asserted. This allows Q and D to be connected together, making for simpler interfacing using a single bidirectional data

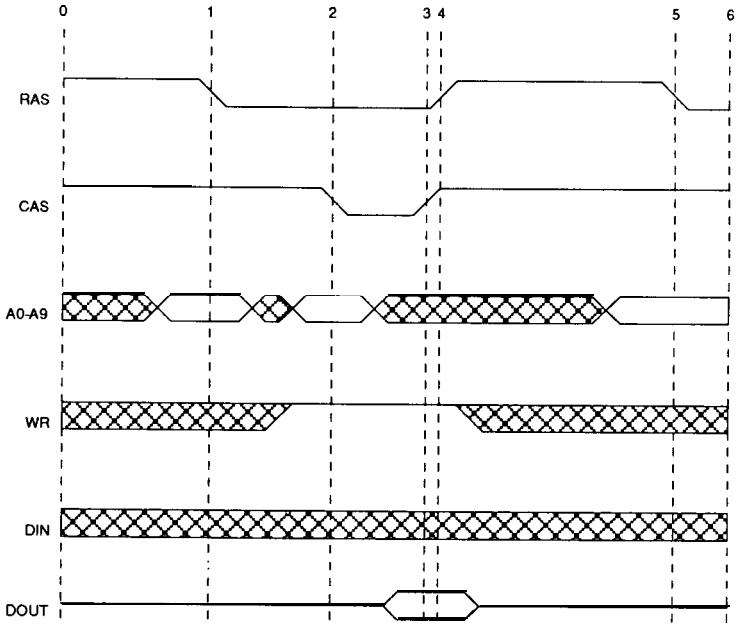


Figure *3-A simplified timing diagram for a DRAM read cycle. A typical DRAM data sheet is fairly* simple *if you don't get lost in the details.*

bus. There is also a delayed write in which the write line is brought low after CAS is low. This is slightly faster, but since the DRAM will be in a read state (since CAS is the output enable pin) with its outputs active until W falls, it is not possible to tie data input to data output directly. If you look at a DRAM data sheet, this simple sequence of events is buried in setup and hold times between each event in the cycle. All these times are important, but they certainly mask what is fundamentally a simple process.

These are the basics of the hardware interface, though there a few timing parameters important to the tester that should be discussed: refresh timing and access time.

Refreshing a DRAM requires that each row be accessed within the refresh period, which is nominally 2 ms for a 64K part, 4 ms for a 256K part, and 8 ms for a 1M part. The refresh

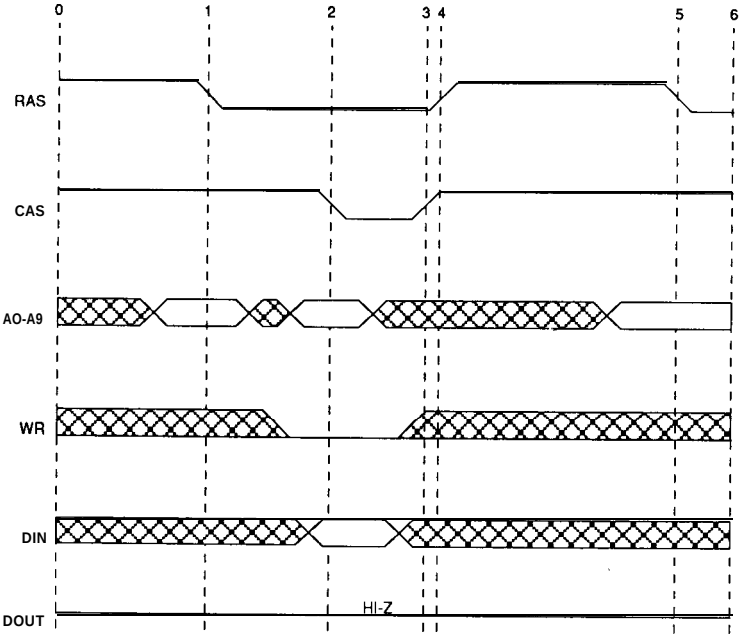


Figure **4**—*The* actual *write cycle for a DRAM is similar to this illustration, but the write line, "W," and data input line, "D," are set up before CAS falls.*

takes place a row at a time internally each time that row is accessed. During a refresh, the DRAM reads the value of each bit in the row and rewrites it, putting a fresh, full charge on the tiny capacitor storing the data. There are two details that need to be added to this simple idea: Modem DRAMsonly require half their rows to be accessed for refreshing. The lower half and the upper half of the part are refreshed simultaneously due to the internal organization of these parts. (The actual organization is two half-sized arrays.) This means that a 64K DRAM only requires that 128 rows be accessed rather than the full 256. The other detail is that a full memory access cycle is not required for refreshing. The minimum that is required is a row address input and a brief RAS strobe as shown in Figure 5. This is called RAS-only refresh and is the simplest refresh method. Each time a row is accessed, when the RAS line is returned high, all capacitors in that row are recharged. This is called the "precharge" timeand typically takes about the same amount of time as the row access time (to be discussed later).

```
on entry:  low row address in ACC
           high row address in DPH
           low col address on PO
           high col address on P2

entry setup for writing-
     CLR   RAMWE      ; write
     SETB  RAMDIN     ; write a 1
or-  CLR   RAMDIN     ; write a 0

entry setup for reading-
     SETB  RAMWF.     ; read

do memory operation,  putting Acc out on PO
and DPH on P2-
     MOVX  @DPTR,A  ;  generate  RAS

8051 WE line is latched by hardware that now holds
RAS line low,  row address has been strobed in

8051 write cycle is now over and ports 0 and 2 have
column address,  generate CAS now-
     SETB  RAMCAS     ; end cycle

RAS hardware reset by rising edge of CAS-
end
```

Listing 1 -*Addressing for a read or write for any RAM location in the Data test modes uses a code segment similar to that shown here.*

Refreshing at a system level is accomplished in a variety of clever ways. IBM PCs perform a periodic dummy DMA cycle which reads a large block of memory. One of the many things that made the Z80 micro-

processor so successful was that it contains internal hardware for refreshing a whopping 64K of DRAM directly. A memory hungry friend of mine designed an 8031 system which used hardware counting circuits clocked by PSEN to perform refresh on external DRAM. The DRAM Tester presented here depends on continually accessing the part for the duration of the test, so no explicit refresh is required. The only requirement for refresh is that it be done within its maximum time and that no accesses are done during that period.

The final groundwork that we must lay before we get into the design of the tester is how DRAM access time works. Access time is the amount of time it takes to get the data once you've met the conditions to access it. In contrast with a static RAM in which there is only one access time, there are two access time numbers for a DRAM: row and column. In a properly designed system, row access time will be the limiting factor and is the access time used to specify DRAMs. Row access time is the minimum guaranteed time required between RAS going low and data being valid on the output, assuming some maximum delay between the RAS and CAS (the time between dotted lines 1 and 3 in Figure 3). This is similar to access time for a static RAM. The column access time is an important number to a designer and is the minimum guaranteed time required between CAS going low and data being valid on the output. The reason the column access time is important is that the row access time can only be met if a proper delay is inserted between RAS going low and CAS going low. If the delay is too short, the address multiplexers in the DRAMs will not work properly. If the delay is too long, the access time will be dominated by the column access time and the guaranteed row access time is meaningless. The DRAM tester uses a fixed delay of 40 ns between RAS and CAS when testing for access time. This is a good delay for all but the very fastest parts (<60 ns).

This is all you have to know about DRAMs to understand the Tester. Let's now move on to the design.

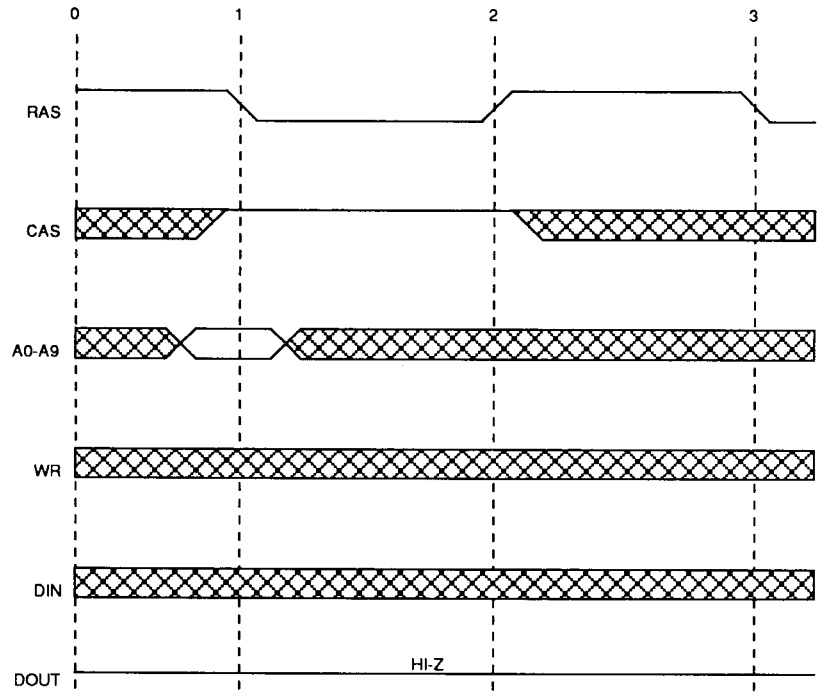


Figure 5—*All DRAMs must be refreshed on a regular basis. The simplest type of refresh is the RAS-only refresh shown here.*

SPECIFYING THE DESIGN

The first step in any design is to think it through thoroughly and decide what you want the widget to do. I started a little notebook of ideas and thoughts for this design a few weeks before I actually began to get serious about this project. The original major goals I set and a little discussion of each are listed below. I was able to stick with these with a few exceptions.

Truly portable operation—Battery operated with a "reasonable" battery life, hand held, and rugged.

This was very important since my main need was testing unknown parts at the local swap meet. I was willing to, and did actually, make a few performance compromises in the design to keep the device hand held. The "reasonable" battery life goal meant I could use the system for a heavy day of testing. "Rugged" is being able to throw it in the glove compartment of my car and forget about it until the next swap meet.

Test access time and data integrity of all common DRAMs—Access time from 80 to 200 ns; data test of 64K × 1, 256K × 1 , 64K × 4, and 1024K × 1 parts.

The box absolutely had to exhaustively test each cell for data integrity, but I hemmed and hawed over the access time requirement. I especially wavered on this when I got further into the design. The access time test looked too difficult to implement within the size constraints. I included it mainly because a lot of surplus parts are bought on faith concerning access time. The 80–200-ns timing came from the parts I generally buy. I later deleted the requirement for testing 64K × 4 parts since the pinouts were very different and would have required additional hardware. I've since considered adding this capability to a more advanced design. (It's never over...)

"Smart" user interface —A keyboard and alphanumeric LCD display interface rather than rotary switches, lights, and so on.

This one was more for fun than anything else. I like little alphanumeric prompts and keyboards better than lights and switches though I'm sure either could get the job done. I wanted something that would catch the judges' eyes in the design contest, of course!

These simple specifications went a long way toward defining the design and keeping me on track. I wanted to think things through a little better than usual for the design contest.

THE DESIGN

The first part of the design was to try to figure out how to interface the RAM in a reasonable way and to make some basic design decisions. It was nearly a given that the design would be based on an 8051-series device. The main reason for this was the size constraint and that I have development tools for the 8051. I knew the 8051 was capable if I could come up with the right design. Other basic design decisions were a size constraint of 1"×3"× 5", operation from a 9-V battery (also sets the power budget), and a one-line 16-character LCD display. These were all somewhat arbitrary but allowed me ground rules around which to make other decisions and tradeoffs.

DATA TEST HARDWARE INTERFACE

Once the basic design constraints were set I began trying to figure out how I could use an 8051 to address the DRAM and generate RAS and CAS for the data testing. I realized early that, since the data test had to test a lc of cells in a part, the time could get out of hand if a cell test took longer than even a few microseconds. The speed really becomes a problem when you start looking at a 1M part. The timing goes something like this: The tester needs to, at the very minimum, write and read a zero and a one to and from each location. It would also be nice if the read didn't immediately follow a write of the same location since this is, after all, a dynamic RAM and short-term amnesia is a significant failure mode. The result is at least four million access cycles for a 1M part.

I started by somewhat arbitrarily hanging the DRAM address lines on 8051 port pins and trying to figure out how the code might work on paper. Accessing the part involved moving a row address to the part, writing a low to RAS, moving a column address to the part, writing a low to CAS, then, in the case of a read, checking to see if the proper output is present, returning CAS high, and finally returning RAS high. This turned into a 17-cycle write loop and a 20-cycle read loop using a very simple-minded interface and

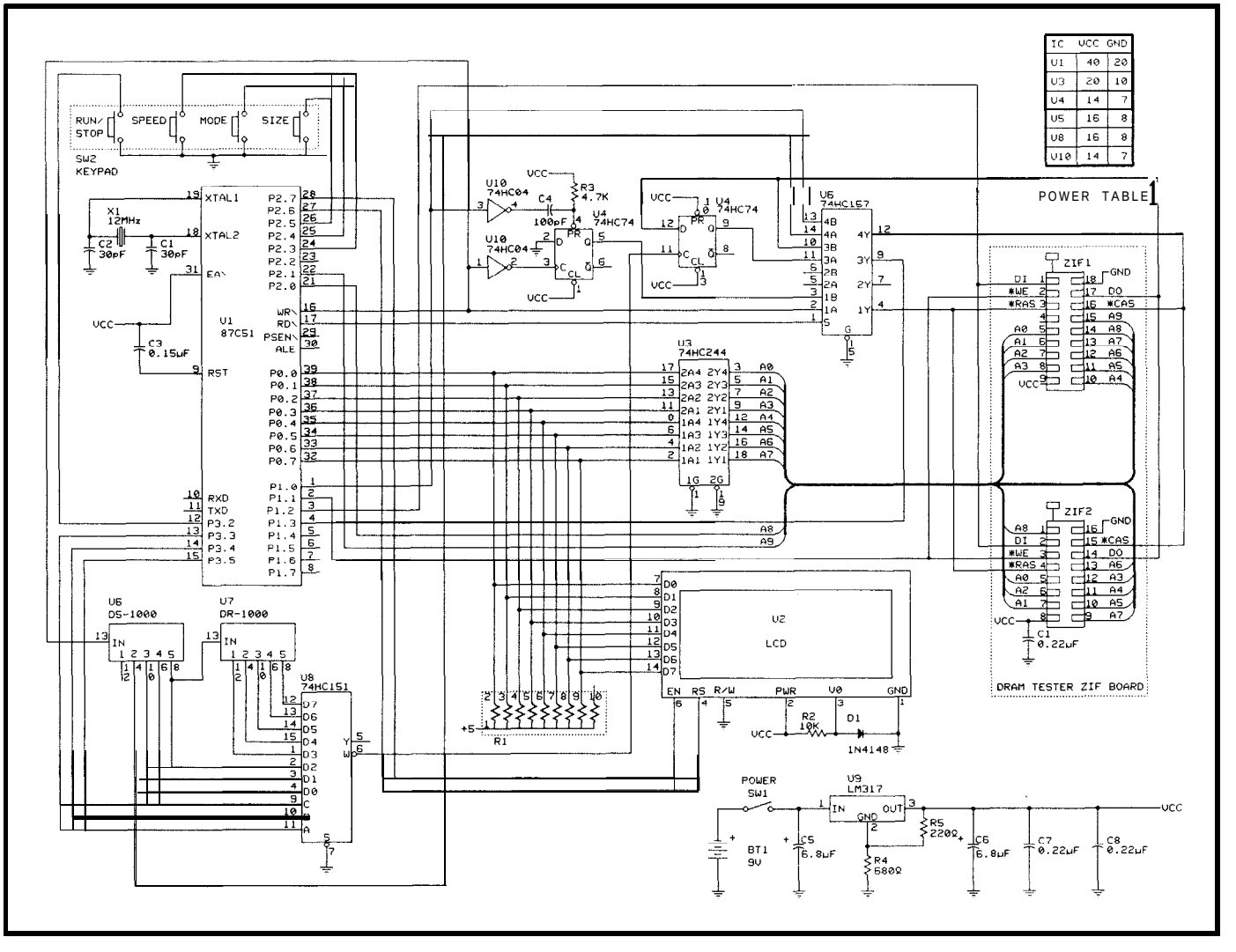
straightforward code. With a 12-MHz 8051 thisresultsina timeof 74seconds for a 1M part. This is a long time and seemed excessive. I spent a some time looking at simple ways to shorten this time. I thought a lo-second total test time would be a good goal.

I looked at a special test mode that 1M DRAMs have just for the purpose of speeding up testing at the factory. Internally, 1MDRAMsareconfigured as four 256K arrays. When a 10.5-V "supervoltage" (a DRAM vendor's jargon-not mine) is applied to a special test pin, any data on the input goes into all four arrays simultaneously, and any data output is thelogical AND of these four arrays. As a result, the part can be tested as a 256K part, taking a quarter of the time. I pursued this to the point of looking at step-up voltage converters to generate the 10.5 volts, but abandoned the approach since I figured out an addressing method thatgotthe1M test timedown considerably. The time was above my lo-second goal but I decided to accept it in the name of simplicity.

The actual interface makes use of some of the 8051's built-in memory-access hardware and a minimal amount of external hardware. This interface is covered in detail in the hardware description section that follows, but basically takes advantage of the fact that the port pins on an 8751-type device are also used for external memory accesses. By placing the column address on the port pins (PO and P2) and the row address in the data pointer register (DPTR), external memory operations making use of DPTR rapidly switch port 0 and port 2 from column addresses to row addresses, simplifying both the hardware and software. An external flip-flop is used to make RAS the proper length to serve itschip select function. RAS is cleared on the rising edge of CAS which is generated by the 8751's write enable line. Using this interface brought the test time for the data test to 32 seconds for a 1M part, only eight seconds for a 256K part, and a blazing two seconds for a 64K part. I decided this was acceptable and very workable from a parts count standpoint.

ACCESS TIME TEST HARDWARE INTERFACE

This article gives the impression that the design was a serial process—first I did this, I finished that, then I thought about this... That's not really the way any design goes. While I was developing the data test interface I was also looking at what impact decisions would have on the access time interface. A lot of things that looked reasonable for the data test made the access time measurement impossible. In the data test, the part is simply exercised and timing isn't really an issue. In the access time test, you need



to perform operations in the tens of nanoseconds and measure times in the hundreds of nanoseconds with decent resolution. This kind of timing is not something any processor can do very easily, much less a minimal 87C51 system.

Several approaches to the this problem were considered. One approach was a brute force method of just measuring the access time using a high-speed clock. A 100-MHz clock could be gated into a counter when RAS fell, a short counter would generate CAS four clocks (40 ns) later, and finally, when data was available from the RAM, theclockinput to the counter would be disabled. The contents of the counter would be the row access time in tens of nanoseconds. While this is possible, it isn't very practical at any reasonable power level. Crystal-controlled 100-MHz oscillators themselves are not very low power circuits much less counters capable of being clocked at 100 MHz. A design approach bites the dust.

I started looking at delay lines and quickly realized they were a good fit. Delay lines are used routinely in DRAM circuits to generate the special timing required. They produce delays that are accurate to a few percent, have TTL-compatible inputs and outputs, and are reasonable on power. The series of devices that seems the most complete is made by Dallas Semiconductor. They makedelay lines with a single input and multiple output "taps" at fixed delays. The device that looked best was one with five 20-ns taps for an end-to-end delay of 100 ns. The DRAM tester uses two of these devicesend to end inconjunction with an 8-to-1 multiplexer to produce a programmabledelayof 60-200 ns with a resolution of 20 ns and an accuracy of ±5%. Five percent accuracy is ±10 ns for a 200-ns part, which is half the time resolution (20 ns) and very reasonable in my book. Access time is actually measured by clocking a D flip-flop with a delayed RAS signal and bringing the RAM's data input into the D input. If the D input is stable in time for the clock, then the access time passes. If not, Q will not reflect D and the access time fails.

One of the tough problems that had a very simple solution was how to jam a row and then a column address onto the address lines and strobe them in time to see the minimumrow access time. The delay between RAS and CAS has to be on the order of *40* ns to meet the row access time. This means that a row and then a column address have to change very quickly. The simple solution to this problem was to not change the column address after strobing in the row address. This only allows access time testing on "diagonal" addresses in which the row and column address are the same, but after all, access time is independent of address and the hardware savings are significant. The CAS signal in the access time test mode is generated by taking a fixed 40-ns delay out of the first delay line whose input is RAS.

TEST ALGORITHMS

The test algorithms are really the key to the DRAM tester. I wanted to make the hardwareand software flexible and modular enough so that new

algorithms and test modes could be implemented as I learned more about DRAMs and thought more about the testingrequirements.Testingisalways a tradeoff: speedy testsare convenient and sometimes will catch a failure, exhaustive tests take longer but usually don't show up any failures that wouldn't be found in a quick test. The algorithms I developed are adequate for my needs and probably most general testing needs. They are on the speedy side of the tradeoff balance but don't compromise too much. All of the tests can be made harder, longer, or "better" if you'd like, generally by adjusting constants or making small changes in the firmware. The source code is fairly modular and should be commented well enough for you to see where it could use a tweak for your improvements. *[Editor's Note: Software* for *this article is available from the Circuit Cellar BBS and on* Software On *Disk #20. See page 91 for downloading and ordering information.]*

The three test modes that the DRAM tester presently supports are called Data Test Mode, Long Test Mode, and T_{acc} Test Mode. The Data and Long Test Modes are practically the same. The Data Test first does some pin tests to check for "stuck" address lines, then reads and writes a zero and a one to each memory location in the device. The **Long** Test Mode repeatedly runs the Data Test until either a failure occurs or the test is manually aborted. This should be useful for finding suspected intermittent failures. The T_{acc} Test measures the access time of the device with a 20-ns resolution from 70 to 210 ns. The user can select either an auto speed mode in which the access time is displayed or a manual mode where a speed is entered by the user and the tester just displays a pass/fail result.

THE GRITTY DETAILS

Power for the system is provided by a single 9-V battery which is regulated to 5 V using an LM317 monolithic regulator. An LM317 regulator was used rather than a fixed 7805-type so power to the RAM chip could be adjusted under microprocessor control if margin testing is required at some point in the future. Power supply bypassing is provided by C6–C8. The 87C51 is set up conventionally with C3 providing a nominal 5-ms power-up reset, and crystal Xl providing a 12-MHz clock source with capacitors Cl and C2. The keyboard, SW2, is interfaced to port pins P2.3, P2.4, P2.5, and interrupt 0. Interrupt 0 is used for the Run/Stop key so that tests can be aborted while running without having the overhead of polling the keyboard. When not testing, all the keys are polled. The LCD display module is connected to port 0 for data, P2.6 for RS, and P2.7 for EN. Resistor network R1 provides 4.7k pull-ups on all the port 0 lines to allow it tooperateasanoutput port. Writing to the display consists of moving the data to port 0, putting a 1 or a 0 on P2.6 for RS (commands or data respectively), and pulsing the EN pin (P2.7) high for 1 µs.

The display voltage is set by the forward drop across diode D1. Although very simple, this setup works well. The 74HC244 octal bus driver (U3) is used to isolate the RAM chip under test from the display data pins. Thisensures that shorts in theaddress lines of the RAM won't interfere with writing messages to the display. The HC244 is always enabled as a straight through buffer. The remainder of the hardware is used for the actual testing of devices. A basic description of its function is described here but its use will become clearer when the test details are discussed in the software section. The function of this hardware is significantly different depending on whether Data tests or Tacc tests are in progress. A quad two-input data selector, U5, is controlled by the processor to select whether a Data or a T_{acc} test is to be performed.

The microprocessor begins the Data test by writing a 1 to P3.7 which instructs the three sections of U5 to route their B inputs to their outputs. U5a selects the source of the RAS signal, either directly from P2. 6 (also write enable) in position A or from the Q output of flip-flop U4a in the B position. U5b is not used. U5c is used to select the data source from the RAM

data output in to the 8051 as being either direct (in position B) or latched by flip-flop U4b (in position Al. U5d is used to select whether CAS is generated directly by the processor (in position B) or generated by delaying RAS by 40 ns through U6, the first silicon delay line (in position A).

The 8-to-1 data selector, U8, is used to select which of eight delay times from the two silicon delay lines, U6 and U7, are selected to strobe flip-flop U4b, which latches the data coming from the RAM in the access time measurement mode only. The selected outputs from the delay lines are from 60 to 200 ns which relate to access times from 70 to 210 ns when other device timing and propagation delays are included. Flip-flop U4a is used to generate the RAS pulse from the8051 write enable pulse in the Data test mode. This method of using the 8051's internal write timing and tristatedriverscontibutes to thelowparts count in this instrument, saving two 18-pin DIP drivers and a couple of gates. It is a solid, though nonstandard, approach which probably deserves some explanation.

To set up for a memory operation in the Data test mode, the low row address is placed in the accumulator and the low column address is placed on port 0. The high column address is placed on P2.0 and P2.1 (A8 and A9) and the high row address is placed in the DPH register. A MOVX @DPTR,A instruction begins the cycle by doing an 8051 write cycle. Only the end of the write cycle is used in the tester. At the end of write cycle, write enable goes low while the accumulator (low row address) is on the data bus. The write enable line clocks flip-flop U4a, which has its D input tied low. This forces its output low, generating the RAS signal to the RAM and strobing the contents of the accumulator and DPH (the high address is present on port 2 during a write) to the row address of the RAM. The write cycle ends in 400 ns, at which time port 0 and port 2 regain their normal functions of outputting their port pin values (the low column and high column address, respectively). The CAS signal is generated by bringing P1.O low which strobes in the column address now appearing on ports 0 and 2. Finally, the cycle ends when P1.0 is returned high and the half-monostable circuit created by R3 and C4 generates a 0.5-µs pulse that presets flip-flop U4a, returning RAS to a high state.

SMALL LINKABLE MODULES

The RAM Tester firmware is written in 8051 assembly language. It is partitioned functionally into four linked modules called RAMTEST, CHKDTA, TACC, and RTUTILS. RAMTEST is the main module and contains all of the user interface code, power up and interrupt vectors, and the data structures for constant and variable storage. CHKDTA contains the code for the Data and Long test modes. TACC contains the code to perform the access time tests. RTUTILS contains global use subroutines and utilities for such things as writing data to the display, generating delays, and so on. The structure of having small linkable modules makes the software easier to modify and improve. This ease of improvementwasamajordesigngoal. The highlights of these modules are described in the sections that follow.

RAMTEST MODULE

RAMTEST is the main software moduleandbeginsexecutionatpower up. It first initializes the LCD display then sets up system variables with their default values, sets the stack pointer, and falls through to wait for a key press. The RAMTEST module primarily handles the keyboard and keeps track of the user-selected test options. When the Run/Stop key is pressed, a small code segment determines which test is requested and branches to that test routine. The test options are stored in several tables in ROM. These data structures consist of a five-character ASCII string to be displayed at the right side of the display for that option followed by a one-byte value which is used by the appropriate test routine.

There are three variable counters—spcnt, s zcnt, and mdcnt-that keep track of which

option in each of the three tables is being used, and three variable values that store the current values from each table. Each table is terminated by a 0. This use of tabular data structures makes the software very flexible.

The way these value and count variables operate on the tables is similar, so a description of one should suffice. spcnt stores the selected number of the entry in the speed table (sptbl). A spcnt value of 0 would select auto speed, a value of 1 would be 210 ns, and so forth. When a key is pressed, register R7 is first checked to see if an old key or new key is being pressed. If a new key is being pressed, the user wants to view the current value of that key and spcnt is not changed. When the same key is pressed again, spcnt is incremented and the ptnext subroutine (in the RTUTILS module) is called. The ptnext subroutine gives the offset from sptbl to the selected ASCII string in the accumulator and leaves R0 with the "value" variable—the byte that appears after the ASCII string in the table. The "value" is loaded into spval and will be used by the TACC code later. The show1 display subroutine (in the RTUTILS module) is called to display the current ASCII string at the right of the display. both ptnext and show1 aredescribedmore fully below.

TWO TESTS IN A MODULE

The CHKDTA software module is responsible for both the Data and Long test modes. After the complete Data test is finished, the variable mdcnt (mode count) is checked and if it is zero (Data test) the pass/fail result is displayed. If it is nonzero (Long test) the test is run again (and again).

One of the unique features of this tester is the way in which it generates the row and column addresses for the RAM. Addressing for a read or write for any RAM location in the Data test modes uses a code segment similar to the pseudocode shown in Listing 1.

This segment is at the micro level of the Data test. At the macro level, there are three stages. The first is the size check which works by first writing 0 to every byte in the RAM that is on a 64K boundary. It then writes a 1 with all the address lines high. The software then searches for this 1 in 64K steps until found. If it is found at FFFF, then the part is a 64K part since it did not see any of the higher order address bits. If it is found at 3FFFF, then the part is a 256K part, and finally if it is found at FFFFF, then the part is a 1M part. This value is displayed if auto size was requested and the unit is not doing a Long test. Otherwise, it is compared to the manual size entered and if different, a size test fail prompt is generated and the test stops. If the size test passes, the test continues to the pin test.

The pin test checks for open pins on the device. It operates by first clearing location 0000 of the RAM. It then writes a 1 to each locationin the device that has an address with a single address line high (e.g., 0001,0002,0004, etc.). If a pin is open, then this single address line will not be seen and the

Reader Service #106

write will overwrite location 0000. After all locations are written, location 0000 is tested for a 0 and a pin test fail prompt is generated if it is 1.

If the size and pin tests pass, then the actual Data test code is run. This codewritesand thenreadslsand Osto each location in 64K blocks. Writing and reading small blocks allows failures to be determined faster on larger parts. The 64K blocks are actually written and read by four subroutines in the CHKDTA module called WR1BLK, RD1BLK, WR0BLK, and RD0BLK. If a failure is found in any of the reads, a Data test fail prompt is generated and the test terminates and returns to the main module. Otherwise the Data test pass prompt is displayed and the same return is made.

SIMPLE ACCESS TIME SOFTWARE

The TACC software module is responsible for access time measurements and is very simple since most of the test is done in hardware. Because of the speed required to do the access time measurement, the row address and column addresses are kept the same for all the measurements. Keeping the row and column addresses the same eliminates the time overhead of quickly (very quickly at 70 ns) removing the row address, applying the column address, and strobingit. What this means is that only "diagonal" addresses can be used for access time tests-a minor design tradeoff since access time should be independent of address. Because the access time measurement is primarily hardware based, a description of the software amounts to a description of the hardware in the context of the test.

The access time measurement begins by writing 0s to all of the "diagonal" addresses in the lower 64K block of the part. It then writes a 3-bit address to the 8-to-1 multiplexer'to select which tap from the silicon delay line it will use. This 3-bit value is usually the value that was stored in spval by the keyboard routine before the test was run. RAS is generated by bringing P2.6 low. This strobes the row address into the DRAM and is also the input to the delay line. CAS is generated 40 ns later by a tap out of the delay line and strobes the same address into the column address of the DRAM. The DRAM's data output pin is pulled up by a 10k resistor so that either a tristated output or a 1 will produce a high output. This output is tied to the D input of a flip-flop which is clocked by the output of the 8-to-1 multiplexer: the programmable delay. Since the RAM was filled with Os, the processor looks at the Q output of the flip-flop after it is clocked and expects to see a low if the access time measurement was successful. The software subroutine in the TACC module that performs this test on 256 locations is called TACSUB. In theauto speed mode, the access time measurement is started atitsslowest speedandisdecremented by 20 ns per step until a failure speed is reached. It is then incremented by one to the last known good condition and displayed. In the manual mode, the TACSUB subroutine is called *256* times for a more extensive test and a pass/fail result is displayed.

DATA TEST

The Data test is the default test mode at power up. This mode tests the basic read/write operation of each cell of the DRAM. It is selected by pressing the Mode key until "Test Mode Data" is displayed. If the user wishes to enter a size for the part, the Size key is used to select it. The default size is Auto in which case the tester will determine and display the RAM size at the beginning of the test. RAM size is selected by pressing the Size key repeatedly until the desired size is reached. Speed has no effect during the data test so can be set to anything.

The test is run by pressing the Run/Stop key once and verifying the "Test Ready..." prompt, then pressing it once more to run the test. The test can be aborted when running by pressing the Run/Stop key again in which case the display will read "Test Aborted..." The possible test results for the Data test are:

1. Data Test Pass-All portions of the data test passed. This indicates that all cells in the part are functioning (good part).

2. Data Test Fail—Indicates that at least one cell in the part is not capable of storing a 1 or a 0.

3. Pin Test Fail—Indicates that one or more address or output lines are shorted, tied high, tied low, or are open. Make sure the part is installed correctly in the socket before you assume the part is bad.

4. Size Test Fail-This display indicates one of two conditions depending on the setting of RAM size. If RAM size is set to auto, it indicates that no valid size for the part could be found. If RAM size is set manually, it means that the size determined is different from that set. This failure could occur, for example, if a 64K part was being tested with the RAM size set to 256K.

TACC TEST

The T_{acc} mode is used to measure the access time of the part. It isentered by pressing the Mode key until "Test Mode Tacc" is displayed. The speed maybe set automatically or manually, with Auto being the power-up default. The Auto mode is more useful for most applications and will simply determine the access time and display it as "Auto Spd = xx ns." The Manual speed mode is useful for sorting parts by speed and will display "Tacc Test Pass" or "Tacc Test Fail" depending on whether the speed is less than or greater than the speed selected. The Auto mode can also display "Tacc Test Fail" if the measurement cannot be made or if the access time is greater than 210 ns. Once the speed is selected, the test is run by pressing the Run/Stop key twice.

LONG TEST

The Long test is a variation on the Data test that just runs the Data test repeatedly until either a failure occurs or the test is aborted by pressing the Run/Stop key. The Long test mode is entered by pressing the Mode key until "Test Mode Long" is displayed and then pressing the Run/Stop key twice. "Long Testing.. ." will be displayed. The same failures as the Data test are displayed. There is no explicit pass in this open-ended test, only a nonfailure at the time the test was aborted.

CONSTRUCTION

The prototype that I submitted to the contest is pictured in the opening photo and opened in Photo 1. There is nothing critical about the packaging, though if you want something that you can really use, it needs to be solid. Since this unit is a hand-built prototype that included parts I had on hand, the best I can do is to tell you what I did, though your packaging may differ somewhat.

The plastic enclosure I used is made by Unibox (LMB makesone that is similar) and is 3.9" x 1.5" x 5.25". I machined the box with an 1/8" end mill which fits the radius of the Textool ZIF socket nicely with no filing. I roughed out the keyboard holes on the mill and finished broaching the corners with a square needle file. If you don't have access to a machine shop, the whole job could be done carefully with a Dremel and files. A local sign shop made me engraved plasticlabels for the keyboard and ZIF socket for about ten dollars. I wanted to engrave the box directly and fill the lettering with paint but the shop said that the box wasn't "stable" enough. The slide switch hole in the side was made with a drill, a sharp knife, and lots of filing. It is tough to make good-looking, rugged, one-of-a-kind packaging; if you come up with a better way to do it, I'd like to hear from you.

In the end, I built three boards: a keyboard, a ZIF socket board, and a main electronics board. There is also an LCD module that wires to the main board. The main board mounts on bosses provided in the bottom of the box. The keyboard and ZIF board mount on bosses in the top of the box. The LCD mounts with through holes to the top of the box. If I did it over, I would glue little bosses in the top for the LCD so the screws wouldn't be visible.

I'm sorry to say that I can't offer you a PCB layout. I use perf board with Scotchflex IDC socketsand hardware (made by 3M) for all my prototyping projects. If you've never heard

of these sockets, I highly recommend themforalmostanythingbutthemost critical circuits. The technique is similar to wire-wrap in some ways but usesinsulationdisplacementpinsand makes low-profile boards.

MAIN ELECTRONICS BOARD

The overall board size is 3.1" x 3.6" with 0.4" notches in two comers for the bosses in the box. The board is mounted to the box with #4 x 1/4" self-tapping hardware on a 3.2" x 1.5" pattern to match the box bosses. The keyboard wiring hangsoff this smaller board, the ZIF socket board connects with a 20-pin dual row IDC ribbon connector at the top, and the LCD display is connected with a 14-pin DIP headerand ribbonneartheside.Leave good service loops for this wiring but make sure you can close the box.

KEYBOARD

Thekeyboardisalittlestrip board about 0.6" wide and 3.6" long. I used stiff solder-typeperfboard forit so the key switches could be soldered down and would feel solid. The keys I used are 0.3" on a side, are available from good electronics wholesalers, and are made by Calectro. The board alignment is critical for smooth operation and aesthetics. I point-to-point soldered the wiring on the board and ran a ribbon over to the main board.

ZIF SOCKET BOARD

The ZIF socket board uses similar construction techniques as the keyboard. I used point-to-point wiring with 30-gauge wire-wrap wire and solder. I taped over the back of it with electrical tape since it is a bit delicate. It has about a 2" ribbon cable and dual row 0.100" center IDC header hanging off it that mates with the top of the main electronics board. The clearance of this ribbon cable with the battery is a bit tight.

LCD DISPLAY

The LCD display is a standard Japanese display module. They are available on thesurplusmarket at very low prices; I buy them whenever I see a good deal. Though I wasn't able to find one, a bottom view display is preferred since this is almost always the orientation of the display. The display is connected to the main board with a 14-pin DIP header and ribbon cable about 3" long. I generally wire the 14-pin header to agree with the 14 pins of the display. Though this involves a little bit of scramble wiring of the ribbon cable, it's easier to troubleshoot and I've standardized on it for all my junk box displays. The display is mounted with #2-56 machine screws at its four comers. If you use the machine screws, use spacers or be careful tightening them since the LCD boards are fairly fragile.

A STARTING POINT

This article has described the design and construction of a portable DRAM tester. My hope is that this article will be a starting point for the reader to embellish and improve. Some ideas for improvements might be the inclusion of "by 4"-type parts or support for SIPs and SIMMs, both of which are considerably more common now than when I developed this unit. Another improvement might be automatic power shut-off, a necessity in a portable instrument I think. I'm sure improvements could be made in the test algorithms and software, too. I also deliberately left the processor's UART available so that a serial operation mode could be included. This might be useful for logging tests to a printer or getting sort data. Anyway, have fun and let me know what you come up with by leaving me some mail on the Circuit Cellar BBS. I'd really enjoy hearing from you.

John Wettroth is the chief engineer at SAIC Mil Products in San Diego, Calif. He also owns Travtech of San Diego, which produces an HP 48 calculator I/O interface.

IRS

404 Very Useful
405 Moderately Useful
406 Not Useful

Using the TI74 for Data Acquisition

FEATURE ARTICLE

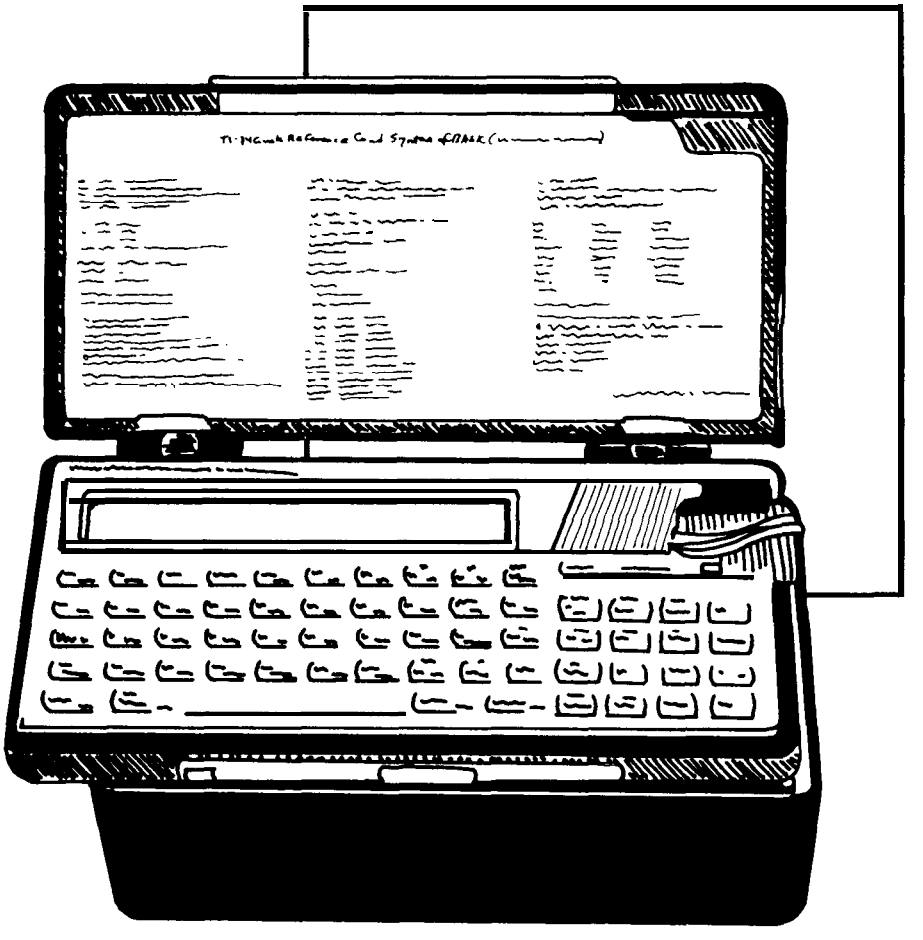
Ed Vogel

Low-Cost BASIC I/O

I have often looked at my calculator and asked, "If this thing is so smart, why do I need a multimeter and counter/timers to take data from experiments?" It is, after all, a microprocessor-based instrument; all it needs is an I/O port and some programmability. Hand-held computers are a little better. Most have a serial interface which requires more programming than I care to do.

Enter the T174. It has thecomplete address, data, and control buses plus a chip select on an edge card connector for memory expansion, greatly simplifying theinterfaceand programming task. The T174 is a reasonably priced ($100) hand-held computer/scientific calculator fromTexasInstruments. When programmed by an IBM PC or compatible (adapter cost is$60), it is capable of many modest data acquisition tasks via the I/O adapter I'll describe in a bit. Most of the parts needed to wire-wrap and package the project are available at Radio Shack. The edge card connector, T174, and ICs are attainable from a number of other sources, including those listed at the end of the article. A solderless breadboard and power supply are good to have on hand since 24 digital I/O lines are only so useful by themselves.

This project consists of three main assemblies: T174, enclosure, and I/O board as shown in Photo 1. The enclosure is not absolutely necessary, though it does provide strain relief and protect the edge card connection. I don't recommend adding more circuitry to the I/O board since I don't have firm numbers regarding the T174 power supply capability. In its present incarnation, this unit is bulky. Those of you who have considerable experience in electronics will no doubt find clever ways to make it smaller and power supply independent. This articleisfor those witha tight budget—my friends who teach high school physics, for instance.



Most hand-held computers have a serial interface which requires more programming than I care to do.

THE I/O BOARD

Before putting any parts on the I/O board, drill four 3/16" holes near

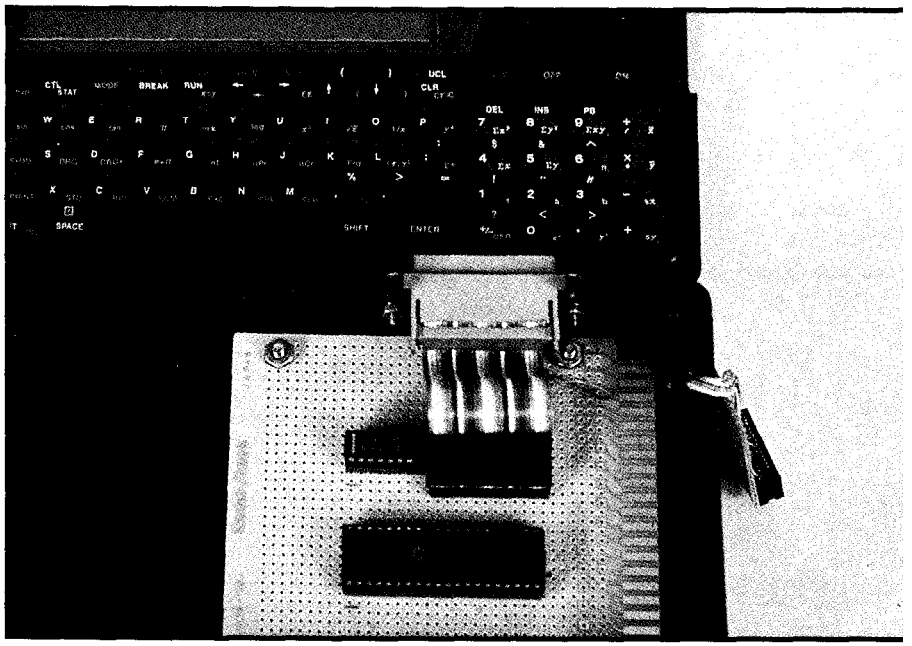


Photo 1 -*the 1174 data acquisition unit consists of three main assemblies: the T/74, the enclosure, and the I/O board.*

the corners of the prototyping board. Mark the hole pattern onto a separate piece of paper for use later in the construction of the enclosure. Solder the wire-wrap sockets, Cl, C2, and R1 into the PC board. Mark pin 1 of each socket on the bottom of the board for reference. Using the schematic shown in Figure 1, start making wraps, marking off each connection as it is completed.

Attach a DB-25 connector to one end of the ribbon cable. Split out wire number 13 and install the 24-pin IDC socket on the other end. Solder the wire that was split out to circuit ground and install the IDC socket in the 24-pin wire-wrap socket on the I/O board.

THE EDGE CARD CONNECTOR

This is the most difficult part of the project: Work methodically and take your time. Set the edge card connector in a table vise and situate the I/O board pins up nearby. Cut a twelve-inch piece of wire-wrap wire and strip one end for wrapping. Strip the other end back one eighth of an inch and form a tight hook. Tin the hook with solder. Using the TI connector illustrated in Figure 1, solder the pretinned wire to pin 14. Wrap the other end of the wire to pin 34 of U1.

Cut out a piece of perfboard to the dimensions in Figure 1 and attach this piece to the edge card connector.

THIS IS ONLY A TEST

The experienced may wish to stop here and do their own custom application. The project is now at a semi-functional level so testing is in order regardless of additional plans. I used the following items to perform the test: T174, IBM adapter, I/O board, multimeter, and oscilloscope or logic probe. I am not going to do a detailed

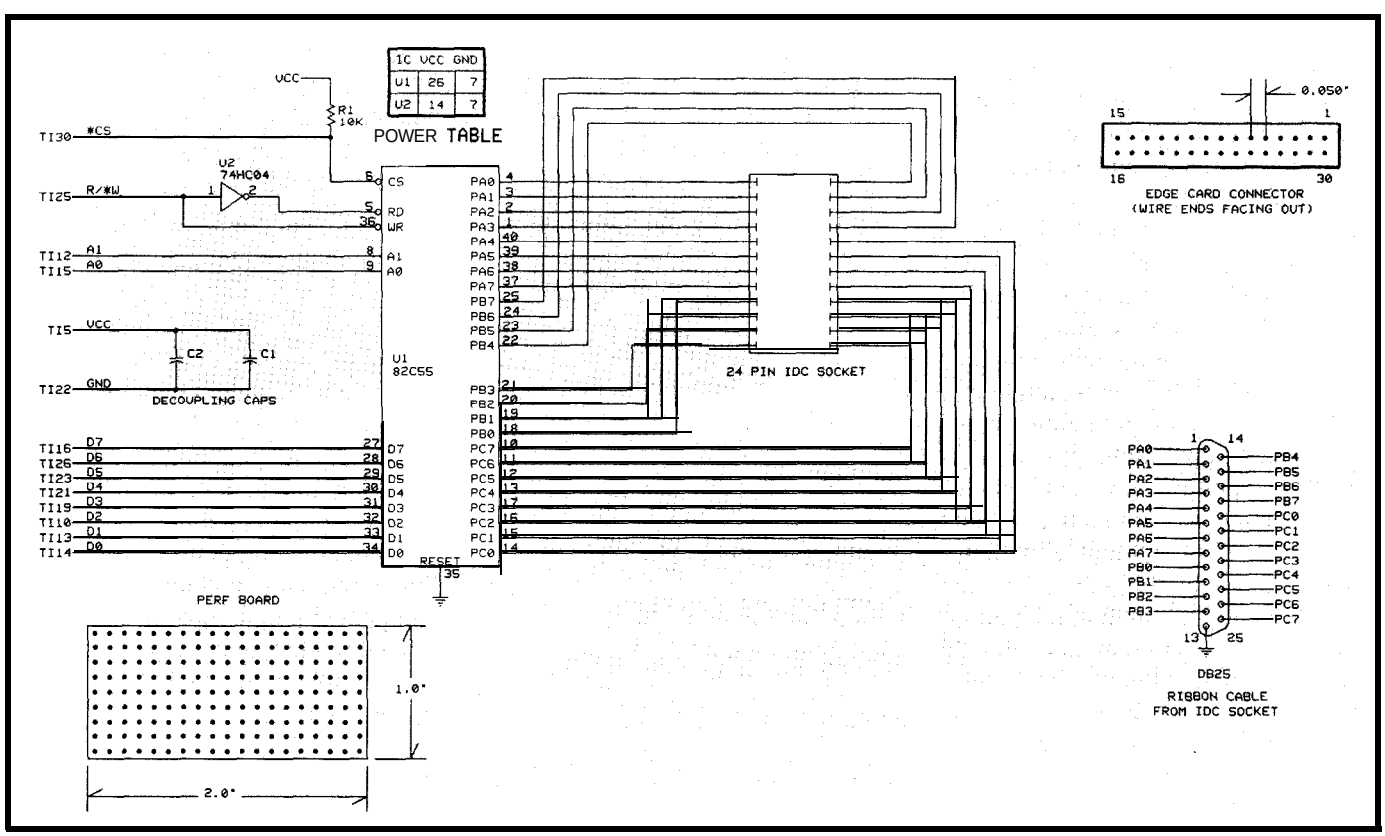


Figure 1—*The heart of the I/O board is an 82C55 PPI. Connections from the board to the outside world are through a 25-pin D-type connector. A ribbon cable from a* **24-pin** *D/P header to the 25-pin D connector completes the connection.*

```
10 CALL POKE(16387,128)   ;SETS DATA DIR FOR ALL PORTS AS OUTPUT
20 CALL POKE(16384,0)     ;OUTPUTS 0 ON PORT A
30 CALL POKE(16384,255)   ;OUTPUTS 255 ON PORT A
40 GOTO 20                ;LOOP
```

listing 1 - *This short TI74 BASIC routine may be used to check the interface*

explanation of the IBM adapter operation since TI's documentation covers it adequately. I will offer two important hints: use complete DOS paths with the delimiters specified by TI, and execute the BASIC NEW command if the T174 returns an error message when you CALL PEEK or CALL POKE.

The two files to load from the IBM are PEEK.SUB and POKE.SUB, which are in the PC interface system disk directory UTIL74. Connect the T174 to the I/O board. The edge card connector and perf board should fit snugly in the cartridge port. Turn on the T174. A blinking cursor should appear on the display's far left. If it doesn't or if there is an error message, turn the T174 off. Sometimes the bus can hang so thoroughly that the off switch will not function. If this happens remove the batteries and unplug the edge card connector. Inspect the wiring for shorts, especially at the connector, and ring them out with a multimeter. Assuming the wiring problems are fixed reconnect the I/O board, power up, and type in the program in Listing 1.

Consfrucfion of fhe enclosure is straightforward and requires no measuring. The only fools needed are a drill, a hacksaw, and a file.

Run the program and use a logic probe or oscilloscope to check for activity at U1 port pins PAO-PA7. They should be toggling at a rate of about twenty times a second. If this is not working, power down the T174 and start looking for wiring errors.

THE ENCLOSURE

Construction of the enclosure is straightforward and requires no measuring. The only tools needed are a drill, a hacksaw, and a file. Use the piece of paper on which the PC board hole pattern was marked earlier to lay out the bottom of the enclosure for drilling the mounting holes. Drill the holes and install the standoffs. Place the PC board on the standoffs, put the DB-25 connector on the edge of the box, and outline the portion inside the mounting ears. Cut a 3/4" deep slot here. The DB-25 connector should drop in and not interfere with the top cover. Match the alignment marks of the mounting ears to those in the enclosure, drill, and install the mounting

hardware. Put rubber feet on the bottom of the box. Install the PC board on its standoffs, pulling the edge card connector and cable up and out of the box. Slide the cover onto the box just up to the cable and mark it.

File a slot on the cover where marked and smooth with sandpaper. I used electrical tape to protect the cable where it emerges from the cutout. Screw the lid onto the box. Apply several layers of sticky-backed tape to the bottom of the T174 plastic carrying case. Position it carefully on the enclosure cover, sitting in such a way that the edge card connector and cable are not hanging out over the enclosure edge. Press down to set the tape, install the T174 in its **case,** and plug in the edge card connector.

MAKING IT WORK

To simplify interfacing I recommend making up another DB-25-to-24-pin IDC cable. It will plug nicely into a solderless breadboard and will assist in keeping track of wiring connections. Remember to split out wire 13 for ground if you do this. Any wall socket DC adapter regulated through a 7805 will do nicely for +5 V to the breadboard.

The first step in any interfacing is to determine how many inputs and outputs are required since this determines which control word gets written into the data direction register. Read the comments in the test program and the data sheet for the 82C55 for more details. I have interfaced a KAD 0820BCN A/D converter available from Digi-Key, a 74HC574 latch, and a solid-state relay switching AC power on and off to a instrument. The highest sampling or switching rate you can expect is about 20 times a second, which is about the same as that used by multimeters. There is a host of other chips that can be interfaced to this system to do many data acquisition and control tasks.

Ed Vogel works in product development for Science Applications International Corporation and is a part *time science teacher.*

SOURCES

Most parts may be found at Radio Shack. The remainder may be obtained from Allied Electronics. The 15 x 2 edge connector (#235-21-030DS-20) may be obtained from Methode Electronics,

Radio Shack: located across the U.S. and in many foreign countries

Educalc
27953 Cabot Rd.
Laguna Niguel, CA 92677
(800) 677-7001
(7 14) 582-2637

Allied Electronics
7410 Pebble Dr.
Fort Worth, TX 76118
(8 17) 595-3500

Bell Industries
contact:
Methode Electronics
c/o David Norling
(818) 991-4238

IRS

407 Very Useful
408 Moderately Useful
409 Not Useful

The Mystery of Intel Hex Format

Exorcising the Hex Demon

FEATURE ARTICLE

Ed Nisley

There comes a time in every project when your heart stops, your eyes cloud, and you realize that you haven't the foggiest idea what to do next. In embedded systems programming, one such moment occurs when you desperately need data in Intel hex format and all you've got is binary... or vice versa.

Fortunately, Intel hex format is easy to generate or parse, at least once you have the key. After reading this article, you can restart your heart, uncross your eyes, and get on with your other problems.

THE BACKGROUND

Your programs start as eyeball-readable ASCII text in a disk file and, after a trip through a compiler or assembler, end up as binary data in another disk file. The EPROM holding your program contains binary data, so what's the problem? Just stuff the binary data in the binary EPROM, right?

Step back in time two decades, when "hand calculators" had fingers, an "electronic desk calculator" was a suitcase full of TTL, and a "personal computer" got salary and benefits. Perhaps you were one of the lucky ones with a Teletype terminal and an EPROM programmer in your office.

If you were really lucky, that TTY had a paper tape reader/punch, as did the EPROM burner. How would you get data from one to the other, with some assurance that a bit or two didn't fall off along the way?

The Intel hex file is a survivor of the era that replaced TTYs with CRTs, paper tape with floppy disks, and nearly everything else with a personal computer. Although there are other formats around, if you can get your data into Intel hex format, the message will get through.

The fields in each line are

:ccaaaattdd...ddss<cr><lf>

Field	Contents
	Each line must begin with a colon
cc	Number of data bytes in the line
aaaa	Address of first data byte
tt	Record type
dd	Data bytes
ss	Checksum
<cr>	Carriage return character
<lf>	Line feed character

A record that puts the data bytes 12 34 at address 5678:

:025678001234EA

The last record in the file is usually:

:00000001FF

Figure 1 — *The Intel hex format divides each line into specific fields. The first four fields have fixed lengths, while the size of the data field is specified by the 'data length' field. All of the characters are upper case, printable ASCII.*

THE FORMAT

The most essential feature of the Intel hex format is that it uses only upper case, printable ASCII characters. You can print that file, massage it with a text editor, send it over the phone without a fancy binary protocol, or even type it directly into a recalcitrant gizmo by hand (it has been done!).

Figure 1 shows the format of each line, or "record" as it's called in the trade, within an Intel hex file. The format's name and traditional file extension come from the fact that everything is hexadecimal. Every line begins with a colon character and ends with a carriage return and (in most cases) a line feed. Some devices are smart enough to ignore lines that don't start with a colon, although many will grump about an invalid record format, and some don't need the line feed.

The first byte (two characters) gives the number of data bytes in the record. Although a single record can have up to 255 bytes, most devices gag after only 16 or 32. Unfortunately, these upper limitsare generally undocumented and determined by experimentation after a last-minute catastrophe. A zero length byte means there are no data bytes in the record; this usually occurs only in the last record in the file.

The next two bytes (four characters) are the **address** of the first data byte in the line. Despite what you may think from years of experience writing code for Intel 8086 processors, the high-order address byte comes first! Because there are only two addressbytes, an Intel hex file can hold only 64K bytes of data. While there are variant formats for bigger

EPROMs, 64K is enough for reasonably sized projects. *[Editor's Note: "Extended" Intel hex format allows for 20-bit address fields.]*

It is worth noting that successive recordsdo not have to be in ascending address order and they need not be contiguous. Most devices don't care one way or the other, but if your device complains about your file, you may need to sort the records in ascending order of the address field. A standard ASCII sort will work if you skip the data length field.

The next byte (two characters) is the record type. There are only two useful values: 01 occurs on the last record in the file, while 00 marks all other data records. There are a few other "standard" values, but you'd best consult your gizmo's manuals if 00 and 01 don't suffice.

The data bytes come next. There must be exactly as many bytes as specified in the count field, so there will be twice that many ASCII characters in the data field. If you use32 data bytes per record, the resulting ASCII text fits neatly into the 80-column straitjacket of your screen and printer. The number of bytes on each line may vary, but should not exceed 32 bytes for compatibility with many devices.

The checksum is the final byte (two characters) before the carriage return. To compute this value, add the binary values of all the other hex data together, take the two's complement, and thelowbyteis thechecksumvalue. Because the checksum is the last byte in the line, your programs can accumulate it on the fly while the rest of the line goes by. A few examples may clarify this process:

The record to put 00 hex at address 0000 is ":0100000000FF."

The record for 01 hex at address 0203 is ":0102030001F9."

And putting FA CE at address BABE requires the line ":02BABE00FACEBE."

Although upper case and lower caselettersmeanprettymuchthesame thing to our cerebral neural nets, most devices insist on upper case ASCII text. For those of you writing in C, use "%2.2X" format rather than the "%2.2x" that you may think is more readable.

Finally, the last record in nearly every Intel hex file looks like ":00000001FF."

Some **compilers** put the **program's** starting point in the address field, blithely ignoring the fact that record type 03 is set aside for that very purpose. As most microcontrollers start at address 0000, and you'll have gone to great trouble to provide special code for the ones that don't, this feature is generally of little use. Unless your gizmo absolutely requires a special end record, you can hard-code (gasp!) the value shown above.

Address	Contents
8000	ASCII character **used with** BASIC-52 PROG command PROG4 produces 34 hex ("4")
8001	RCAP2H bit rate setting
8002	RCAP2L
8003	MTOP high byte
8004	MTOP low byte
8010	Program start marker, 55 hex
8011	First byte of BASIC program

GETBASIC simulates the PROG4 command by saving the current serial port bit rate and MTOP values. The interpreter will restore these values and start the BASIC program automatically. Other PROG commands can be simulated by changing the character at address 8000.

Hex file addresses typically start at 0000 because the 8052 circuitry decodes the EPROM so the first byte is at address 8000. Your system may require different addresses.

Figure 2—The BASIC-52 **interpreter examines storage** during the power-on *reset* routine to find out if there is a BASIC program stored in EPROM. This table shows what information is required to autostart a BASIC program.

```
REM This program will send tokenized BASIC-52 data from RAM.
REM It stops when it encounters its own line number...
REM Use GOTO 64000 to start uploading to your PC's terminal
REM emulator. You must edit the captured data to remove
REM extraneous lines! EPROM file format matches PROG4, with
REM current bit rate & MTOP.

64000 rem -- tokenized BASIC program hex file uploader
64002 string 35,16 : $(0)="0123456789ABCDEF"
64004 l9=20h : rem maximum data bytes/line
64006 b1=00000h : rem starting address for HEX file
64008 l8=64000 : rem first line number of this program

64050 b0=dby(13h)*256+dby(14h) : rem BASIC program start addres

64100 print "Tokenized BASIC-52 program EPROM header starts on
        next line"

64110 c0=0 : rem initialize checksum
64112 print ":",
64114 d0=10h : gosub 65000 : rem 16-byte header block
64116 d0=b1/256 : gosub 65000
64118 d0=b1.and.0ffh : gosub 65000
64120 d0=0 : gosub 65000
64122 d0=34h: gosub 65000 : rem PROG type number
64124 d0=rcap2/256 : gosub 65000 :rem bit rate
64126 d0=rcap2.and.0ffh : gosub 65000
64128 d0=mtop/256: gosub 65000 : rem MTOP value
64130 d0=mtop.and.0ffh : gosub 65000
64132 for i=1 to 11 : rem fill remainder of line
64134 d0=0: gosub 65000
64136 next i
64138 gosub 65020
64140 b1=b1+10h

64150 c0=0 : rem initialize checksum
64152 print ":",
64154 d0=1: gosub 65000 :rem BASIC marker byte
64156 d0=b1/256: gosub 65000
64158 d0=b1.and.0ffh: gosub 65000
64160 d0=0 : gosub 65000
64162 d0=55h : gosub 65000 : rem the marker!
64164 gosub 65020
64166 b1=b1+01h

64200 rem -- start sending a line
64202 if 01h=xby(b0) goto 64400 : rem last byte?
64204 if b1=(xby*256+xby(b0+2)) goto 64400 : rem our first
        Line

64300 rem -- it's a standard line
64302 gosub 65030 : rem extract line info
64310 gosub 65050 : rem begin hex line
64312 gosub 65080 : rem send data section
64314 gosub 65020 :rem send checksum
64316 if l2<l9 then l0=l2 else l0=l9 :rem set up next chunk
64318 if l2>0 goto 64310
64320 goto 64200

64400 rem -- do end of file
64412 c0=0 : rem initialize checksum
64414 print ":",
64416 d0=1: gosub 65000 : rem line with one byte
64418 d0=b1/256: gosub 65000
64420 d0=b1.and.0ffh: gosub 65000
64422 d0=0 : gosub 65000
64424 d0=01h : gosub 65000
64426 gosub 65020
64428 print ":00000001FF"
64430 print "BASIC-52 program ends on previous line"
64432 stop

65000 rem -- show d0 & update checksum
65002 c0=(c0+d0).and.0ffh
65004 print chr$(asc($(0),(d0/16)+1)),
          chr$(asc($(0),(d0.and.0fh)+1)),
65006 return

65020 rem -- show checksum and finish line
65022 d0=(100h-c0).and.0ffh: gosub 65000
```

(continued)

listing 1 —*This program extracts a tokenized BASIC program from an 8052 system and dumps if in Intel hex format. Lines 64 110 through 64 166 add the header information used by BASIC-52 to identify BASIC programs in EPROM, Lines 64400 through 64428 add the 0 1 byte that marks the end of the program and the standard last line of the hex file.*

FOR EXAMPLE...

Although the canonical sample program for this topic converts a binary disk file to an Intel hex disk file, I'll pick a problem closer to the heart of embedded systems: creating an autostarting BASIC-52 EPROM for your 8052 microcontroller. Even if this isn't your current problem, you can work through the code (everybody speaks BASIC!) to see how to produce an Intel hex file from raw binary data.

GETBASIC.BAS, shown in Listing 1, is a complete program that you download "atop" your own BASIC-52 program. Because it uses line numbers 64000 through 65098, your program cannot use those lines. Use GOTO *64000* rather than RUN to start GETBASIC, so that the interpreter doesn't run your own code. GETBASIC stops when it encounters its own first line in the tokenized file, so the Intel hex output contains only data from your program.

The BASIC-52 interpreter stores BASIC programs in a compressed format, replacing the BASIC keywords with single-byte tokens and converting the line numbers to binary. This conversion happens as you type text into the 8052's serial port and is undone when you list the program. As a result, you can't just burn ASCII text from your disk file into an EPROM and expect it to work (it'sbeen tried!).

The interpreter examines the contents of several specific addresses during the power-on reset routine to see if there's a BASIC program ready to run. If so, it starts execution of the (tokenized) program. It is your responsibility to install an EPROM with the right data at the right addresses before turning the system on.

There are three issues: getting the data out of the 8052's external RAM, creating the appropriate BASIC-52 EPROM program information, and putting it all into Intel hex format for your EPROM burner. The program shown in Listing 1 does all three in one shot, dumping the formatted data out the serial port for you to capture in a disk file. I'll presume you can run your own terminal emulator and EPROM burner.

```
65024 print
65026 return

65030 rem -- set up info on current BASIC line
65032 l1=xby(b0) : rem BASIC line length
65036 l2=l1 : rem remaining line length
65038 if l1<19 then l0=l1 else l0=19 : rem length of this fine
65040 return

65050 rem -- send overhead part of hex line
65052 print ":",
65054 c0=0 : rem initialize checksum
65056 d0=l0 : gosub 65000 : rem line length
65058 d0=b1/256:gosub 65000 : rem address high
65060 d0=b1.and.0ffh :gosub65000:rem address low
65062 d0=0 : gosub 65000 : rem record type
65064 return

65080 rem -- send data part of hex line 65082 do
65084 d0=xby(b0) : gosub 65000
65092 b0=b0+1 : b1=b1+1 : l2=l2-1
65094 l0=l0-1
65096 until l0=0
65098 return
```

Listing 1 -continued

The BASIC interpreter stores the starting address of the program's first line at internal RAM locations 13 and 14 hex, so GETBAS IC **starts dumping at that point. However, the hex file addresses start at 0000 because that's the first byte in the EPROM when it's in the EPROM programmer. To add to the confusion, the EPROM is addressed at 8000 in the 8052 system. If you are using a large EPROM that has address 8000 in the "middle" of its decoding range, or if your system has a custom startup routine that looks elsewhere for the EPROM, you must modify** GETBAS IC **accordingly by changing variable** B 1 **in line 64006.**

Each BASIC-52 program line consists of a binary length byte, the line number expressed as a two-byte

Test program source code:

```
10 rem this is a GETBASIC test program
20 rem download it to your 8052 first, then download GETBASIC.BAS
30 rem start GETBASIC using GOTO 64000
40 rem and capture the results using your terminal emulator
50 rem ---
60 for i=1 to 100
70 print "Current I is",i,"and I squared is",i*i
80 next i
90 stop
```

GETBASIC output:

```
>goto 64000
Tokenized BASIC-52 program EPROM header starts on next line
:1000000034FFDC17FF000000000000000000000000CB
:01001000559A
:2000110025000A9620746869732069732061204745544241534943207465737420707 26F02
:050031006772616D0D16
:200036004000149620646F776E6C6F616420697420746F20796F7572203830353220666 97B
:200056007273742C20746865 6E20646F776E6C6F6164204745544241534943 2E4241530D4B
:2000760025001E9620737461727420474554424153494320757369 6E6720474F544F2036EC
:0500960034303030 0D94
:20009B003A00289620616E64206361707475726520746852072657375 6C747320757369E2
:1A00BB006E6720796F7572207465726D696E616C20656D756C61746F720D55
:0900D5000900329620 2D2D2D0D9D
:0C00DE000C003CA049EA31A63130300D86
:2000EA002C004689224375727265 6E74204920697322 2C492C22616E642049207371756136
:0C010A0072656420697322 2C49E2490DE3
:0601160006005097490DA0
:05011C0005005A900DE2
:0101210001DC
:00000001FF
BASIC-52 program ends on previous line
STOP - IN LINE 65000
READY
>
```

Listing P-Sample *GETBASIC test program. Notice that the hex output does not contain any of the GETBASIC code!*

binary value, the tokenized program text, and a carriage return character. The length byte includes all those fields, so you can find the starting address of the next line by adding the length byte to the current address. The last line of the program is followed by a single 01 hex byte.

The first line number in GETBASIC's code is stored in variable L8 so that GETBASIC can stop producing output when it encounters its own beginning, rather than when it hits the true end-of-program marker. If you don't have another program in RAM when you start GETBASIC, the hex output will consist of the header information and a 01 byte marking the end of the program, with nothing between!

GETBASIC starts a new hex line for each new BASIC program line, so you can edit the hex file to remove program lines.. .but this procedure is not for the faint of heart. It also starts a new line after emitting the amount of data specified by variable L9 (set in line 640041, so you can control the maximum line length to match your gizmo's restrictions.

GETBASIC adds text marker lines before and after the hex data so that the starting and ending points are more obvious in the captured disk file. You must edit the file to remove those lines, as well as any other extraneous data, before feeding the file into your EPROM programmer (unless your programmer ignores lines that don't start with a colon!).

With all that as prologue, GETBASIC's actual workings are fairly straightforward. Listing 2 shows a sample program and the resulting hex output so you can trace through the code.

Examining the first hex line:

```
:1000000034FFDC17FF00
              00000000000000000000CB
```

shows 34 at address 0000, so the BASIC-52 EPROM header simulates a program created using PROG4. The FFDC at address 0001 will set 9600 bits/second with an 11.059-MHz crystal. Finally, the 17FF reveals that MTOP was reduced 2K from the 1FFF normally found with an 8K RAM.

The next line:

```
:01001000559A
```

contains a single byte, the 55 marker at address 0010.

The remaining lines are the tokenized text corresponding to GETTEST.BAS. I'll leave it as an exercise for the reader to match the binary line numbers with the BASIC text. The BASIC-52 manual has a table matching keywords and tokens, so you can check to see that the rest of the program is correct.

The final step is to burn the data into an EPROM, stick it into your 8052 system, and see if it works. Try it!

THINGS TO DO

There are several utility programs that belong on every embedded systems programmer's disk. Given the information in this article, you should be able to whip off a pair of programs that convert hex files into binary files and binary files back into hex. *[Editor's Note: There are numerous binary-to-hex and hex-to-binary conversion programs for the IBM PC posted on the Circuit Cellar BBS.]* A hex sorting utility might be handy, and, of course, It Would Be Nice to download a hex file to your 8052 once in a while.. .

You may be tempted to skip the checksum when you're reading hex files from a PC disk file. After all, what can possibly go wrong with a disk file that wouldn't be caught by DOS and BIOS error checking? As it turns out, every so often my OS/2 terminal emulator would drop a byte. The hex file checksum was the **only** proof I had that something was broken.. .because the checksum was created on a different system!

Long live the Intel hex file!

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

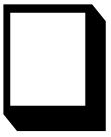
IRS

4 10 Very Useful
411 Moderately Useful
4 12 Not Useful

DEPARTMENTS

page 50

Firmware Furnace



page 57

Silicon Update

page 63

Practical Algorithms

page 86

ConnecTime

You Can't Do That!

A Look at Porting Code From OS/2 to DOS

Perhaps the least interesting part of C programming is deducing which pointer you misused from the observation that your program dies in strange and mysterious ways. The process is rarely boring, because that abused pointer can give your whole machine a serious case of the twitching never-get-overs by mutilating the operating system's code or data.

Try as you might, you can't write C code without at least one such blunder (unless, of course, you use no pointers at all). So you get used to Big Red Switch debugging, even though you know deep down in your heart that the next glitch may scrub your hard disk right down to the platter.

What if you could find pointer errors before they caused any damage? Would that be worth anything to you?

What if your debugger reacted to a wild pointer by popping up a window saying "Protection Violation" rather than freezing in its tracks? What if you could find pointer errors before they caused any damage? Would that be worth anything to you?

As it turns out, I developed the code for an upcoming project entirely under OS/2, using Microsoft C 6.00 and OS/2 CodeView, specifically to get that level of debugging support. When the program was working correctly, I added a few DOS interface routines and recompiled the code. A short session on an AT with DOS-mode CodeView produced, in effect, an OS/2 program ported to DOS!

Although this column usually concentrates on gritty firmware details, tricks for developing the programs merit some attention, too. I'll explore the differences between the OS/2 and DOS versions of my software, then make a few observations on program development by porting from OS/2 to DOS.

THE BIG PICTURE

The whole purpose of an operating system is to separate your program from the actual system hardware. In principle, your program should "talk" only to the operating system and ignore the hardware details. In practice, essentially all PC-DOS programs bypass the operating system to get direct video controller access, handle interrupt-driven serial I/O, and perhaps even snag scan codes right from the keyboard hardware.

OS/2, in contrast, provides a mind-numbingcollectionof functions (called an Applications Program Interface, or API, in the currently fashionable technobabble), enough that

FIRMWARE FURNACE

Ed Nisley

many programs won't need direct hardware access. By default OS/2 will terminate an unauthorized program that attempts to use an `IN AL, DX` instruction; you can't even get access to the I/O ports, much less misuse them!

If your code really needs direct I/O, a `CONFIG.SYS` setting can bestow I/O authorization. A variation will give all user programs I/O authorization, so OS/2's protection isn't quite as strict as you might imagine. And DOS-mode programs running in the DOS box can wreak their usual hardware havoc. So it goes.

The process of porting an OS/2 program to DOS involves nothing more than providing a set of DOS-mode functions to mimic whatever part of the OS/2 API your program uses. Obviously if your program uses many OS/IL-specific API functions, you have a formidable task. But if your intentions are to debug DOS-mode programs under OS/2, the conversion need not be difficult.

USER INTERFACING

An important part of any program is the user interface, because that is the only part of the program "visible" to the outside world. In fact, many programs are mostly user interfaces with a small computational section. There is always a tradeoff between adding convenient features to the user interface and adding vital features to the actual program!

Rather than write Yet Another User Interface, I used the Oakland Group's C-scape Interface Management System. C-scape provides a wealth of functions to implement a

```
/*---------------------------------------*/
/* Initialize the serial handlers        */

int SerInitialize(void) {

#ifdef OAK_OS2
USHORT DosRC;
USHORT Action:
LINECONTROL LineCtrl = {8,0,0,0};
/* 8 data, 1 stop, no parity      */
BYTE DevCmd;

#else
unsigned RateSel;
unsigned Temp;
#endif

  ScrShowStatus("Serial port setup...");

#ifdef OAK_OS2
  DosRC = DosOpen(PortName,&hSerDevice,&Action,0L,
        FILE_NORMAL,FILE_OPEN
        OPEN_ACCESS_READWRITE |
          OPEN_SHARE_DENYREADWRITE |
          OPEN_FLAGS_WRITE_THROUGH,
        0L);
  if (DosRC|| (Action != FILE-EXISTED)) {
    sprintf(ErrMsg,"Can't open serial port: %s\n",PortName);
    CscShowErrMsg(ErrMsg,ERR_FATAL);
  }

  if (DosDevIOCtl(0L,&SerBitRate,
        ASYNC_SETBAUDRATE,
        IOCTL_ASYNC,
        hSerDevice)) {
    sprintf(ErrMsg,"Can't set bit rate to %i\n",SerBitRate);
    CscShowErrMsg(ErrMsg,ERR_FATAL);
  }

  if (DosDevIOCtl(0L,&LineCtrl,ASYNC_SETLINECTRL,
        IOCTL_ASYNC,
        hSerDevice)) {
    CscShowErrMsg("Can't set serial data format",ERR_FATAL);
  }
  DevCmd = 0;
  if (DosDevIOCtl(0L,&DevCmd,
        DEV_FLUSHINPUT,
        IOCTL_GENERAL,
        hSerDevice)) {
    CscShowErrMsg("Can't flush serial input queue",ERR_FATAL);
  }

#else
  if (NULL == (pRecRingBuffer = malloc(RECRINGSIZE))) {
    CscShowErrMsg("Not enough RAM for serial buffer",ERR_FATAL);
  }
```

(continued)

listing 1 —*Serial port initialization. The C preprocessor variable OAK_OS2 is defined when the source file is compiled to create an OS/2 program. The code between the "#ifdef OAK_OS2" and '#else' statements is present for OS/2 compiles, while the code between the '#else' and "#endif" statements applies to DOS compilation.*

```
  pRecRingHead = pRecRingTail = pRecRingBuffer;
  pRecRingEnd  = pRecRingBuffer t RECRINGSIZE - 1;
  RecRingLevel = 0;
  RecRingMax = 0;

  switch (SerBitRate){
    case 1200 : RateSel = _COM_1200; break;
    case 2400 : RateSel = _COM_2400; break;
    case 4800 : RateSel = _COM_4800; break;
    case 9600 : RateSel = _COM_9600; break;
  default :
    sprintf(ErrMsg,"Bad bit rate: %i\n",SerBitRate);
    CscShowErrMsg(ErrMsg,ERR_FATAL);
  }

    _bios_serialcom(_COM_INIT,SerPortNumber-1,
        _COM_CHR8 | _COM_STOP1| _COM_NOPARITY | RateSel)

  if  (!SerIRQNumber){
    SerIRQNumber = SerInterrupts[SerPortNumber-1];
  }
  SerPortAddr = SerPorts[SerPortNumber-1];

  pOldSerHandler = _dos_getvect(8+SerIRQNumber); /* save vect */

  inp(SerPortAddr);             /* clear pending receiver flag */
  inp(LSR);                       /* clear status flags */
  inp(MSR);                       /* clear change flags */
  inp(IIR);                       /* clear THRE flag */

  _dos_setvect(8+SerIRQNumber,SerHandler); /* install handler */
  outp(MCR,0x0B);       /* RTS & DTR active, enable card ints */
  outp(IER,0x01);                    /* enable rec interrupt */

  SerIRQMask = 0x01 << SerIRQNumber;  /* int mask based on IRQ */
  _disable();
  Temp = inp(I8259MASK);
  outp(I8259MASK,Temp & ~SerIRQMask); /* zero bit is enabled */
  _enable();

#endif

  Paused = 0;

  SerSendChar(CHAR_CTRL_C);         /* cancels file transfers &c *
  SerDelayMS(2*SerTimeOutDelay); /* wait for it to settle down    *
  SerFlush();                        /* discard all leading junk *

  return 0;

}
```

listing 1 -*continued*

variety of windows, menus, data entry forms, and so on. Mouse support is integrated into the screen functions and automatically enabled if a Microsoft-compatible mouse is present.

C-scape runs equally well under DOS, in an OS/Z text window, or as a full-screen OS/2 session. It is not compatible with the OS/2 Presentation Manager, but that was not relevant for my purposes. While there were some glitches in C-scape's OS/2 support, and the source code was not compatible with Microsoft C 6.00 (despite Oakland's ads), the library worked quite well and had remarkably few bugs for such a complex system.

With the user interface bottled up in C-scape routines, the program had only two hardware interfaces: the serial and disk I/O routines. By isolating those in two source files, the rest of the code did not need to know which operating system is in control.

Within those two files I used C preprocessor statements to select source code for either OS/2 or DOS. C-scape required an identifier (OAK_OS2) to indicate when the code is compiled for OS/2, so I used OAK_OS2 in my code, too. The preprocessor statements resemble:

```
#ifdef  OAK_OS2
   Os2Stuff();
#else
   DosStuff();
#endif
```

My MAKE script defines OAK_OS2 when compiling for OS/2 and leaves it undefined for DOS compiles, referring to an environment variable that I set from the command line. Based on OAK_OS2's definition, the C preprocessor automatically configures the source code. Rebuilding my program for either operating system is just a matter of setting the environment variable and running MAKE.

Rebuilding this particular program from scratch takes about 20 minutes, so the MAKE script saves the OBJ files in a pair of ZIP archives. I must manually unpack the appropriate ZIP file before starting the first compile, but, after that, MAKE handles the details; including updating the ZIP file with the new routines.

DISK DIFFERENCES

The changes to the disk I/O routines were utterly trivial and can be summed up in one paragraph. My program displays a list of acceptable files whenever it expects a file name, so I used the OS/2 DosFindFirst() and DosFindNext() functions. These calls map directly into the Microsoft C library functions _dos_findfirst() and _dos_findnext() functions, which are valid only under DOS. The OS/2 functions are slightly more versatile, but, as the software I wrote doesn't take advantage of those features, the source changes amounted to two dozen lines of code.

C-scape converts the list of file names into a pop-up list menu. You can use arrow keys or a mouse to scroll through the list, selecting one file with a double click or keystroke. The whole pop-up box is resizable and movable with the mouse. All these functions come from a single line of code that invokes a C-scape routine; you hand it names and it does the rest!

SERIAL SETUP

My project software (which you'll see later this year) communicates with the DDT-51 controller over an RS-232 serial line. OS/2 supports fully buffered, interrupt driven serial ports with a set of API calls, so there was no difficulty finding the features I needed. In contrast, because PC-DOS provides essentially no useful serial port services, every DOS program (mine included) must reinvent the wheel.

An alternative to rolling your own wheel is to use a serial library, similar to C-scape, to handle the serial port under either DOS or OS/2. While there are several such libraries available for DOS, I haven't seen any that provide the same functions for OS/2. I suppose that reflects the relative completeness of the DOS and OS/2 APIs, as well as the perceived payback on an OS/2 version. In any event, OS/2 allows you to roll a much smaller wheel.

Listing 1 shows the key sections of the serial port initialization routine; some preliminary error-checking and status display code is omitted to save space. Even if you're not familiar with the OS/2 API, the OS/2 version (following the `#ifdef OAK_OS2` preprocessor statements) should be much easier to understand than the DOS code (between #else and `#endif`).

The OS/2 serial setup in this example uses just four OS/2 API calls: open the serial port, set the data rate, set the data format, and flush any pending characters. The Microsoft C 6.00 compiler supports the OS/2 API interface with a set of library routines and most of the API's parameters are set with manifest constants. Given the rather bulky names, it's easy to figure out what each function call does: `DosDevIOCtl` sends control information to an I/O device governed by parameters such as `ASYNC_SETBAUDRATE`.

The PC-DOS setup code is significantly more complex, because the required functions are scattered among DOS, BIOS, and C library calls, as well as a few direct hardware I/O instructions. In addition, the code must allocate RAM for a ring buffer, install a serial interrupt handler, and monkey with the interrupt controller chip. All those functions are handled by OS/2 while processing the `DosOpen()` call.

The differences would be even more striking if my software had a truly complex serial interface. As it is, the program sends characters using polling and expects to receive characters only in response to an outbound command. The setup code and serial interrupt handler (there is no transmitter interrupt!) reflect this simplicity; a more complex program would need more hocus-pocus.

```
/*-----------------------------------------------------------------*/
/* Serial port interrupt handler                                   */
/* This is'a hardware interrupt handler, so we don't change        */
/*  any registers. Perforce, it's useful only for DOS programs...*/

#ifndef OAK_OS2

void _interrupt _far SerHandler(void){
char RecChar;

  RecChar = (char)inp(SerPortAddr);     /* fetch the character */

  if (RecRingLevel < RECRINGSIZE){    /* if room for new char */
    *(pRecRingHead++) = RecChar;      /* ... insert it       */
    if (pRecRingHead > pRecRingEnd){
      pRecRingHead = pRecRingBuffer;
    }
    RecRingLevel++;
    RecRingMax = max(RecRingMax,RecRingLevel);
  }

  outp(I8259,EOI);                  /* acknowledge the interrupt */
  return;
}

#endif
```

listing 2—*DOS serial* **interrupt** *handler. This function is needed on/y* **under** *DOS. Characters are transmitted by polling, so there is no transmitter interrupt handler.*

Listing 2 shows the complete DOS serial interrupt handler. It's written entirely in C, so the operation of the ring buffer should be reasonably obvious. Note that this entire function simply goes away under OS/2...

THE OBJECTIONS

The PC magazines are engaged in serious Windows hype; the current dogma is that Windows is what OS/2 really should have been all along. To put this in perspective, roughly a year ago OS/2 was claimed to be what

Windows really should have been all along. What might the coming year hold?

For a while, the biggest and most serious objection to OS/2 was that you needed a bigger and most serious machine with lots of RAM. That's less true today and, better yet, bigger and more serious machines are a lot cheaper. For example, a name-brand-clone 16-MHz 386/SX clone with 2 MB of RAM, a big hard disk, VGA, and all the stuffings costsabout $1500. DRAM is down to $50 per megabyte, even in SIMMs, so a few more megabytes is no longer much of an issue.

Frankly, the next machine you buy will run OS/2 with little or no additional hardwareinvestment. Although nobody bundles OS/2 with their machines (earlier Windowsgiveaways having defined the concept of a "shelf-ware operating system"), the incremental cost of OS/2 is perhaps a few hundred bucks.

Incidentally, there is a difference between "minimum amount of RAM needed to boot" and "desirable amount of RAM to do useful work." Just as you can never be too smart, too thin, or too fast, your system can never have enough RAM. Buy about twice as much RAM as you think you'll need (even for DOS!) and you'll come out about right for starters.

Contrary to popular opinion, you don't need the *n*-kilobuck Microsoft OS/2 Software Development Kit to write OS/2 programs. The Microsoft C 6.00 compiler works just fine, as do a variety of other compilers from other vendors. There are even alternative debuggers available if you don't like CodeView,although the highly touted Multiscope debugger turns out to be incompatible with MSC 6.00 despite advertising claims to the contrary (this should be fixed by the time you read this, I'm told).

Commercial function libraries for OS/2 are still scarce, although Oakland's C-scape does pretty nearly all the tricks you'd need to build a character-mode user interface. All my C-scape customization code worked fine under both DOS and OS/2, without even any preprocessor trickery. Apart from some documentation

problems and the lack of MSC 6.00 source compatibility, C-scape is a very clean design with lots of hooks and capability.

What about all those wonderful DOS programs with no OS/2 equivalents? Well, most DOS programs are quite happy in the DOS box. For those few that aren't, you can use either OS/2's dual-boot option or keep a DOS floppy boot disk handy. The only critical program that I don't have is an OS/2 disk backup routine; I do daily backups from the DOS box and boot DOS about once a month for the Grand Slam backup.

THE PAYOFFS

Part of debugging any program that uses serial communication is figuring out who said what to whom about what. Normally, the program I've been writing about talks to the DDT-51 controller over a three-wire (send, receive, common) cable; I built an octopus connector that routed the two signals to COM3 and COM4 on my PS/2. Because the PS/2 Micro Channel architecture allows multiple serial ports to share a common interrupt line, all three of those ports operate concurrently.

Monitoring the serial exchanges was a matter of starting two REXXTERM sessions in PM windows: COM3 shows the application-to-controller messages, COM4 shows the reverse direction. Meanwhile, the software under test is using COM1 from another PM window. All the results show up in real time!

Relatively little of my time is spent compiling 8031 code in the DOS box. PM supports multiple text windows, so I can edit source code using KEDIT in a PM window, compile it with MSC in another window or the Avocet compiler in the DOS box, then test it using CodeView or download a HEX file to the EPROM emulator through another REXXTERM window.

Mostly as a result of Microsoft's big Windows 3.0 sales extravaganza, the OS/2 versions of essentially all PC-DOS word processing, spreadsheet, and database programs are now on Pause while Windows versions are on Fast Forward. But, as a result of Microsoft's previous (OS/2) sales extravaganza, I have OS/2 versions of WordPerfect, Excel, Paradox, and a large collection of utilities.

THE BOTTOM LINE

OS/2 provides an excellent development environment: once you experience multitasking, background compiles, and crash-proof debugging you'll never want to see a DOS prompt again. Even if you develop PC-DOS programs, using OS/2 will produce a working version faster and easier.

Windows or Desqview may give you many of the same advantages, but I don't have any first-hand experience. If you have tried DOS program development using Windows or any of the other DOS extenders, sign onto the Circuit Cellar BBS and tell us how well it works and what you think of the whole process?

Oh, yes, best of luck with those wild pointers!

IRS

4 13 Very Useful
414 Moderately Useful
4 15 Not Useful

SILICON UPDATE

Pixie Power

A Switch + LCD Combo for Intelligent I/O

Tom Cantrell

Havingcovered big, fancy LCDs in a recent article, let's now turn to a tiny sibling-the Pixie from Industrial Electronic Engineers Inc. (IEE). The most interesting thing about this thumbnail-sized display is that it is built into a push-button switch (Figure 1). Thus, the Pixie qualifies as a complete "I/O" device in less than a cubic inch! Though the Pixie won't replace the CRT and keyboard on our desktop PCs, it could be useful in your next embedded application.

The previous article discussed LCD basics and certainly there is not much to say about the Pixie's role as a momentary switch. The real issues for Pixie are a) how to put it to work and b) what the heck is it good for? As for the latter question, IEE points out the obvious Pixie potential in applications like vending machines, industrial control gear, automotive dashboards, and so on. Certainly, broad acceptance depends on the price and the current quote from IEE-$37.75 @ 500 pieces ($49.95 singles&--seems a little high.

Nevertheless, working under the assumption that electronic stuff always costs less eventually, the Pixie should be able to find a home in an ever growing base of applications. If yours is one, read on to see how to put the Pixie to work. As you'll see, there are three choices along the make-versus-buy curve. Choose whichever is best depending on the state of your technical requirements and wallet.

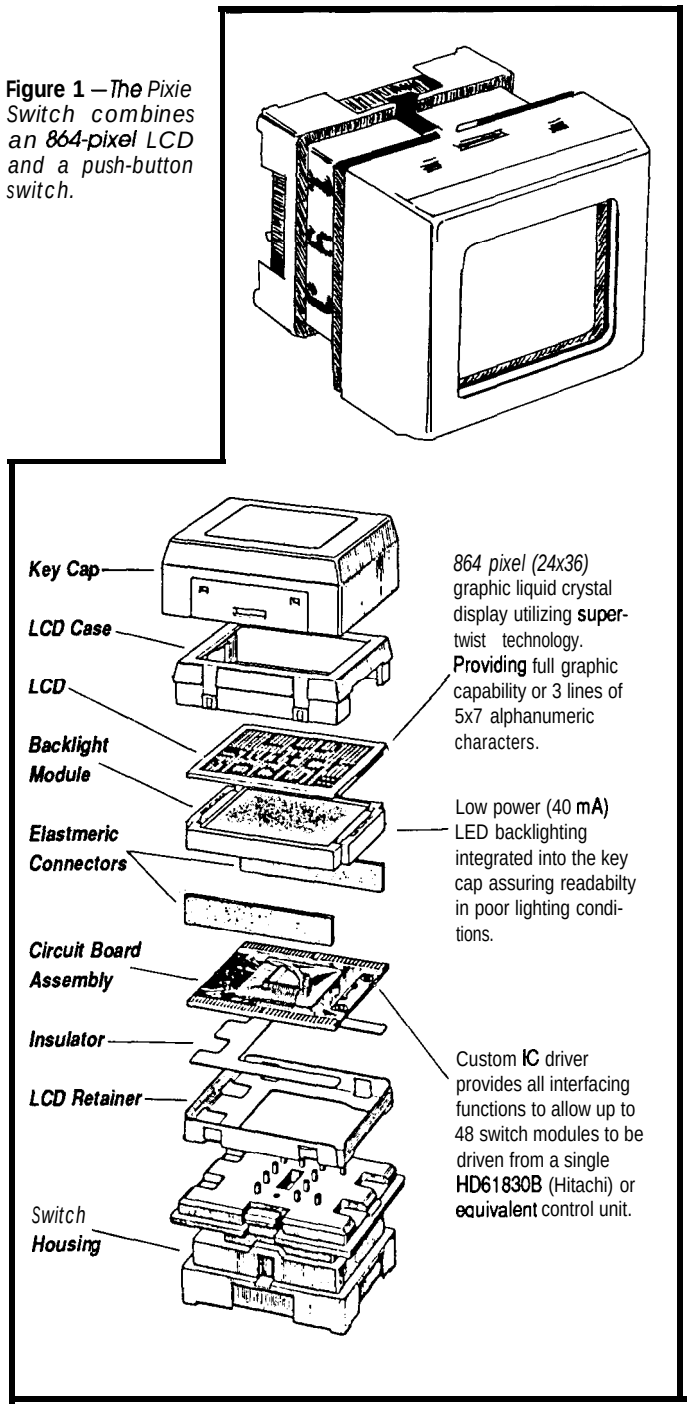


Figure 1 —*The Pixie Switch combines an 864-pixel LCD and a push-button switch.*

ROLL YOUR OWN

If your design is high volume enough to justify extra development costs which reduce unit costs, designing your own direct connection to the Pixie switch may be appropriate.

To get started, let's take a look at the Pixie pinout (Figure 2). The first thing you'll notice is the Pixie's thirteen pins arranged in a distinctly nonsocketable arrangement. This is a shame since the Pixie form factor could, in principle, support a standard 14-pin DIP pinout (I imagine this would cause some grief for the Pixie mechanical designers-otherwise, wouldn't they have made this choice in the first place?). As it stands, the initial challenge (whichever interface choice is made) is the physical connection to the switch. A direct wiring scheme won't work since a PCB

No.	Pin	Function	Connection
1	SW1	switch	user defined
2	SW2	switch	user defined
3	V_{DD}	supply voltage for logic +5V	power supply
4	D_{IN}	data input	D_{OUT} or controller
5	LP	latch pulse	controller
6	FLM	first line marker	controller
7	VLC	supply voltage for LCD	power supply
6	GND	ground (±0V)	power supply
9	D_{OUT}	data output	D_{IN}
10	SCP	serial clock pulse	controller
11	RST	reset signal	controller
12	LED A	LED anode	power supply
13	LED K	LED cathode	power supply

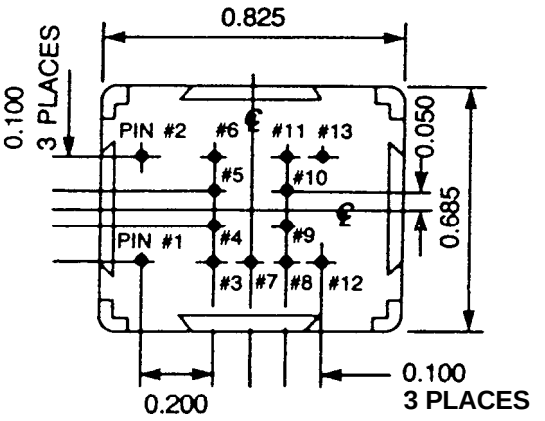


Figure 2— *The Pixie's pin arrangement does not allow for a socketed connection.*

connection of some kind is required to mechanically secure the Pixie. According to IEE, they are working on a solution to ease the Pixie connection-contact them for the latest information.

Reviewing the pin description, VDD and GND supply 5 V to the Pixie logic. More troublesome, VLC is the LCD drive voltage which not only is typically negative (e.g., -3 V) but also needs to be adjustable since it determines the "contrast" (actually viewing angle) of the LCD. Fortunately these days, negative voltages are much easier to come by thanks to monolithic converter chips from outfits like ICL and Maxim. The low-power virtues of LCD technology are illustrated by tiny power requirements (VCC: 0.5 mA, VLC :300 µA typical).

The corresponding weakness of LCDs—the need for ambient light to "reflect"-is overcome with a built-in LED backlight. Unfortunately, as is usually the case, the backlight power consumption dwarfs that of the LCD. The Pixie calls for 5 V at 20 mA on the LED A input (GND on LED K) to brighten things up. Remember to include a 50-ohm or so current-limiting resistor on LED A.

Of course, the Pixie is a switch just like any other and SW1/SW2 are the normally open contacts. Key specs are 12-V/50-mA contact rating and 5-ms switch bounce. Thus,

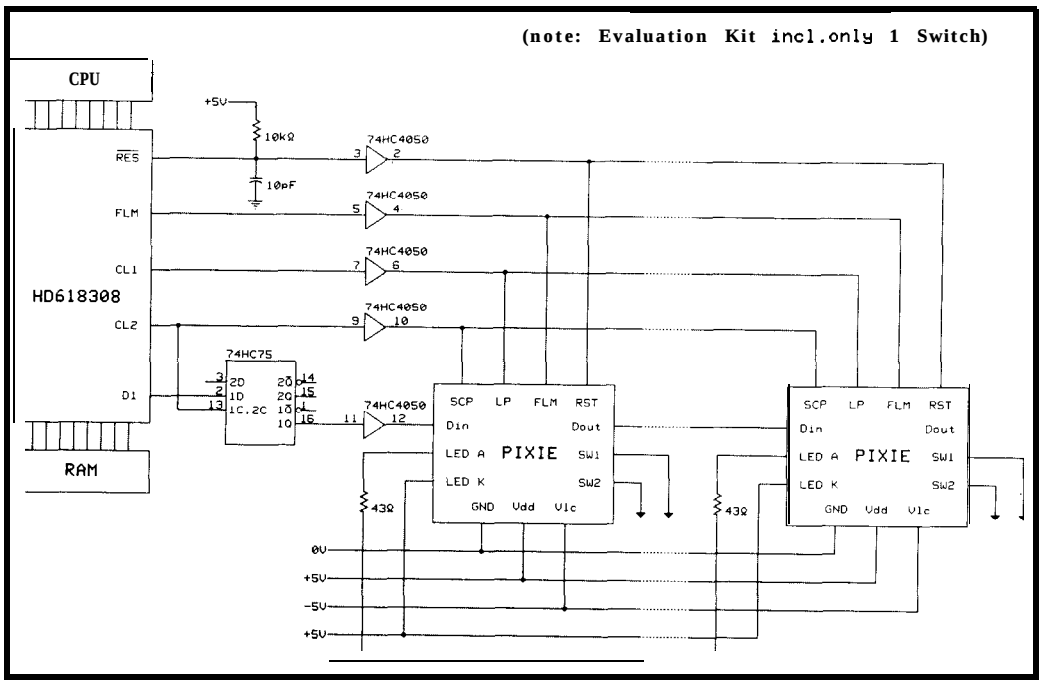


Figure 3—*The Pixie Evaluation Kit needs only SRAM to* **make a working prototype** *circuit.*

you can use your favorite keypad encoder or debouncing algorithms to interface the switch.

RST is simple-just give it a 2-ms or so pulse whenever power is applied to the Pixie.

Now we get to the meat of the Pixie LCD interface in the remaining five signals: SCP, LP, FLM, D_{in}, and D_{out}. By the way, note that RST and these five signalsall require CMOS-level inputs so buffers will berequired to drive the Pixie with the typical micro's TTL-level I/O lines.

Inside the Pixie, LCD driver chips offer a CRT-like interface to the outside world, though the actual LCD display mechanism is completely different from that of a CRT.

The Pixie is organized as 24 rows and 36 columns, thus the Pixie "frame buffer" is 864 bits. SCP (Serial Clock Pulse) corresponds to a CRT dot clock and is used to shift each of a row's 36 bits into the Pixie D_{in} lead. At the end of each row, LP (Latch Pulse) latches the shifted bits into the row (similar to a CRT HSYNC). Finally, once per "frame" (that is, every 24 rows) FLM (First Line Marker) is pulsed. This is like a CRT VSYNC, though in the Pixie it provides AC modulation for the LCD power.

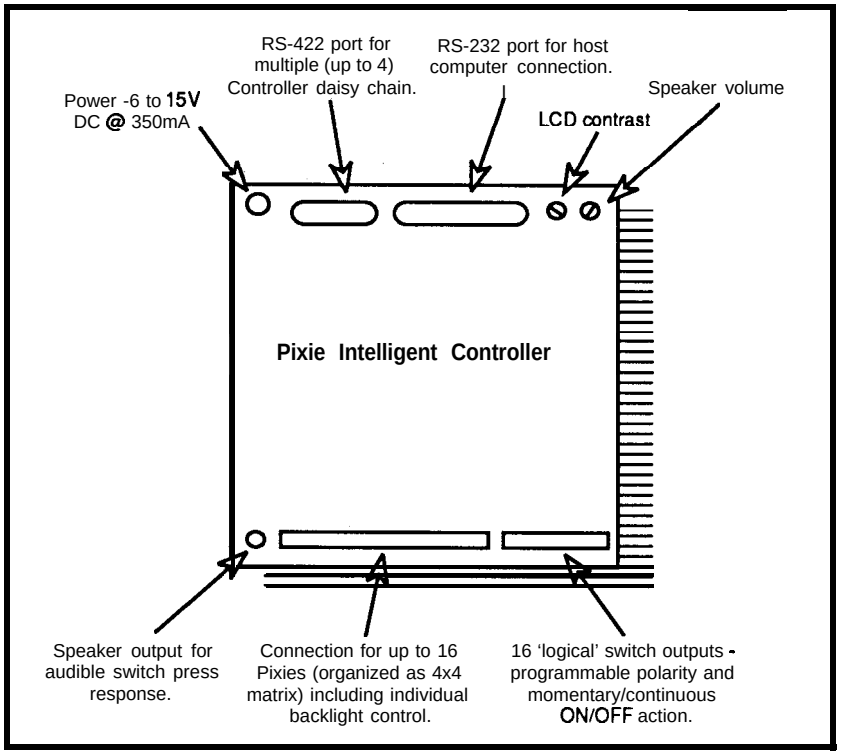


Figure 4—*The Pixie Intelligent Controller takes away electron/c design questions, at a cost of $595.*

Connecting a bunch of Pixies is made easy by the D_{out} pin. Each switch's D_{in} and D_{out} pins are connected in a daisy-chain fashion. Conceptually, each additional Pixie extends the frame buffer another 36 bits in the horizontal direction. Thus, a four-switch setup would appear as a 144 x 24 bitmap. For each row, the bits shifted first will appear on the "farthest" Pixie and those shifted last on the "closest" Pixie.

Like a CRT, the Pixie LCD needs to be refreshed 30–40 times or so per second to stave off annoying flicker. A little math shows that (like a CRT) a little resolution can turn into a surprisingly high bandwidth/processing requirement. A single switch calls for a bit every 30 µs or so which doesn't sound too bad. However, adding switches directly cuts the time between bits—sixteen switches need to be fed bits in less than 2 µs which is rather a challenge.

A bandwidth saving trick which you might consider is to abandon the Pixie D_{in}/D_{out} daisy-chain scheme and dedicate parallel I/O lines to the D_{in} line of each switch. In this case, the refresh burden drops back to that of a single switch. Note that this scheme calls for "packing" all switch's bitmaps together (so a single memory access can refresh multiple switches). Thus, the tradeoff for making refresh easy is that initializing/changing the bitmaps is an interesting exercise-a single switchbitmapis"scattered"acrosstheentire multiswitch buffer.

KISS

The Pixie connection can take advantage of interface chips traditionally used with larger panels, in particular the Hitachi HD61830B. This chip will take care of all the low-level details (SCP, LP, FLM, D_{in} timing, etc.) of the refresh operation totally relieving the host micro of that bothersome task. Hooking up the Pixie to the HD61830B is a snap (note that the CMOS level shifters are still required). If you elect to go this route, order the IEE "LCD Switch Evaluation Kit" which, at $49.95 (the same price as the Pixie switchalone), includesone Pixie, HD61830B, and related components as shown in Figure 3. All you have to add is an easily connected SRAM.

Since the HD61830B handles the refresh, it needs a frame buffer itself and provides an address/data bus for a bytewide RAM memory connection. A 6116 2K-byte static RAM is a good match, providing enough storage for more than 16 (16 x 864 bits = 1728 bytes) switches.

The HD61830B host micro interface is a simple peripheral-type interface of the 68*xx*-type relying on an 8-bit data bus, a R/W (Read/Write) direction line, an RS (Register Select, typically A0) line, and an E (Enable) line which strobes the data transfer. The chip is easily managed in software using parallel I/O lines or, for the more ambitious, directly interfaced to a micro's high-speed bus. Just

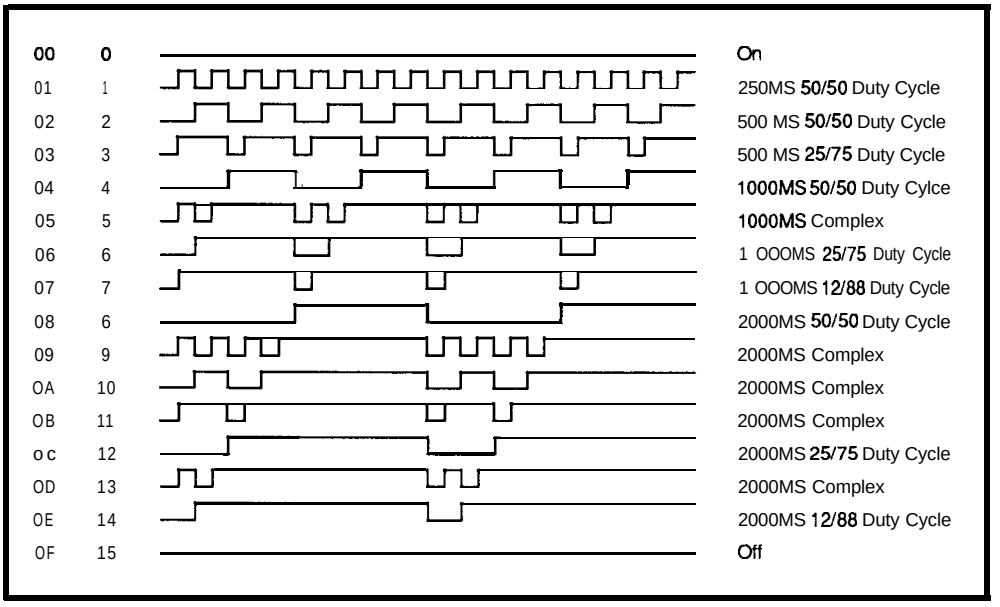


Figure 5—*The Pixie offers a number of options for both LCD backlight color and blink duty cycles for maximum flexibility in display types.*

watch out for the relatively slow interface timing specs of the HD61830B which may call for a wait state or two depending on the speed of your micro. Ultimately, after all is connected, the micro can issue commands/data via the HD61830B to turn individual Pixie pixels on and off.

THE EASY WAY OUT

Don't have time to fiddle the bits? IEE has the answer (assuming you've got $595) in the form of a small (6" x 6" x 2.5") box called the Pixie Intelligent Controller.

The Controller (shown in Figure 4) combines the aforementioned HD61830B with an HD64180 MPU subsystem and all the ancillary circuits: CMOS-level drivers, VLC negative voltage converter with "contrast" trim pot, even a speaker output jack for adding an audible "click" when a Pixie switch is pushed.

The box connects to your micro via RS-232 which, despite its foibles (how many person-centuries have been spent wrestling with a stubborn RS-232 port or cable?), is far easier than either of the previously described schemes.

A 50-pin header offers connections for up to 16 switches (you still have to fabricatea Pixie wiring/mounting mechanism). This "Pixie Bus" features individual controls for each of the 16 switches' backlights and assumes the switches are arranged in a 4 x 4 matrix (the Controller handles the scanning, debouncing, etc.).

Besides handling all the low-level details of the Pixie interface, the Controller offers higher level LCD and switch functions thereby off-loading the host computer.

The Controller can store 256 Pixie bitmaps: 128 predefined and 128 user-defined (note that the user-defined bitmap RAM is battery backed). Once stored in the Controller, the host can assign any legend to the switches simply by sending (via RS-232) a 16-byte "Legend Attribute Block" in which each byte specifies a legend code for a switch.

Similarly, the switch backlights are handled with a 16-byte "LED Attribute Block" in which the lower four bits of each byte specify the LED status for an individual switch (remember, each switch backlight is individually controllable). The 16 choices include ON, OFF, and 14 different blinkrate/pattem combinations (Figure 5). This feature is undoubtedly designed to enhance the user-interface (e.g., an important or "enabled" switch may blink). According to IEE a forthcoming version of the Pixie will feature tricolor (red, green, and red+green [i.e., yellow]) backlighting. I recommend moderation when exploiting the blinking backlights lest your switch array end up looking like a Christmas tree.

The switch closures themselves are handled in two useful ways. First, notification of a switch closure is sent to the host via RS-232. Alternatively, the Controller includes a connector with 16 lines-one for each switch. A nice feature is that the **Controller** performs a mapping between the normally open momentary Pixie switches and the 16 output lines. For each switch, the line output polarity and momentary/continuous action can be specified. Using

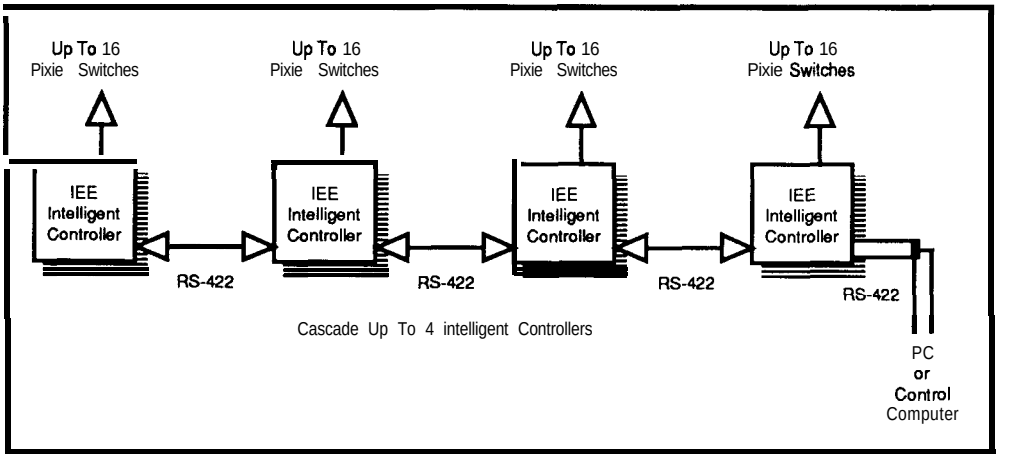


Figure 6—*Up to four Intelligent Controller units can be daisy-chained from a PC or control computer, allowing as many as 64 Pixie switches to be centrally controlled.*

these signals and the mapping feature can allow direct action by the Controller in response to Pixie switch presses without host computer intervention.

If that wasn't enough, the Controller also includes a dedicated 4-wire RS-422 link which allows up to four units to be daisy-chained thereby increasing switch capacity to sixty-four (see Figure 6).

Right now, such a setup is rather expensive. Nevertheless, I suggest you keep your eye (not just your finger) on the Pixie. ✣

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 involved in chip, board, and system design and marketing.

Contact

Industrial Electronic Engineers, Inc.
Component Products Division
7740 Lemona Ave.
Van Nuys, CA 91409-9234
(818) 787-0311
Attn: Louis V. Hronek (x383)

IRS

416 Very Useful
4 17 Moderately Useful
4 18 Not Useful

Adjusting Standard Deviation to Sample Size

Increased Reliability for Basic Statistics

PRACTICAL ALGORITHMS

Charles P. Boegli

Taking the average of a number of values as an indicator of "central tendency" is common practice; so common, in fact, that merely whispering to an engineer the (easily demonstrated) fact that it may not be, often visibly discomfits him. But if the distribution of measurements is unbounded, unskewed, and meets a few other conditions, the mean, which minimizes the "variance," is usually the best indicator of central tendency.

Its calculation is usually followed by that for the standard deviation, which shows how closely the various measurements group. The squares of the differences between the mean and each measurement are added. The sum is divided by some quantity, and the square root of the resulting number is considered the standard deviation σ. The quantity by which the sum of the squares is divided is ordinarily the number of measurements n:

$$\sigma = \sqrt{\frac{\sum (x - X)^2}{n}} \qquad (1)$$

in which x is an individual measurement, X is the mean, and the summation is done over n measurements.

Usually, the standard deviation of the lot from which the samples were drawn is of far more interest than that of the samples themselves. Some texts recommend, in certain cases, dividing by $n-1$ instead of n, using the symbol s to designate the resulting quantity. Given a large number of measurements, the difference is almost academic. But often the accuracy to which a quantity has been found must be inferred from rather few measurements. Then the standard deviation found by either method will be in error.

That this is the case can be demonstrated by *reductio ad absurdum* in the limit. If a single measurement was made, expression (1) indicates it is absolutely precise. The mean and the measurement being identical, the standard deviation is then zero. On the other hand, dividing by $n-1$ yields, for a single measurement, an infinite standard deviation, suggesting that no reliance at all can be placed on the result, which is equally silly.

When the standard deviation (σ′) of a parent popula-

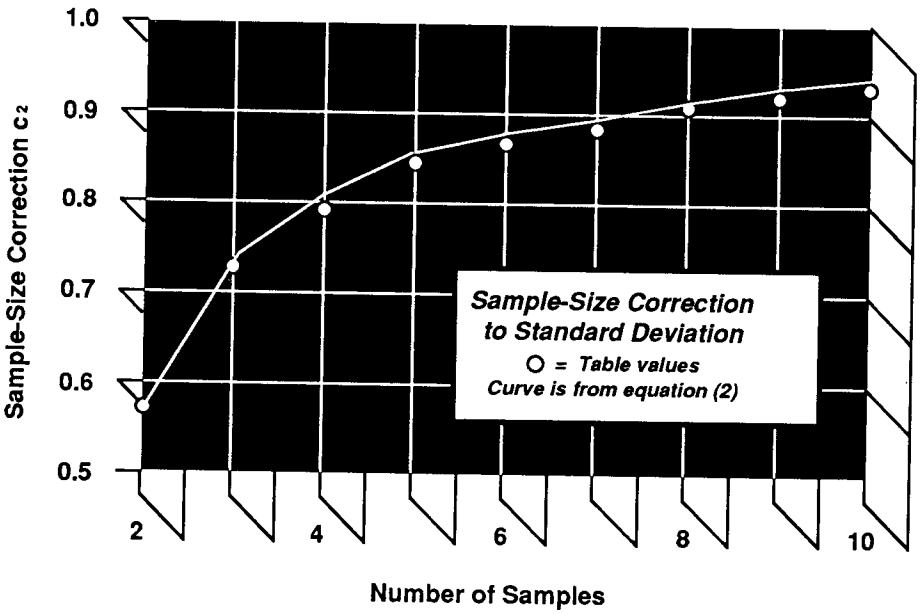


Figure 1—*As the number of samples increases, the necessity to compensate for a small number of samples decreases, hence c_2 approaches 1.*

tion or "universe" is to be estimated from a few measurements, that found by expression (1) may be divided by a factor c_2 that depends on the sample size. A table of these factors appears in many books on statistics, such as the "Statistical Quality Control Handbook" (AT&T, Indianapolis, Indiana) on page 131:

Sample Size	c_2
2	0.5642
3	0.7236
4	0.7979
5	0.8407
6	0.8686
7	0.8882
8	0.9027
9	0.9139
10	0.9227

In Figure 1, the value c_2 is plotted against sample size. As expected, c_2 rapidly approaches 1.0 as the number of samples is increased. The figure also shows a curve of the function

$$c_2 = \sqrt{1 - \frac{1.382}{n}} \qquad (2)$$

which, considering the restriction on the fit, is remarkably close to the values in the table (the restriction was that it must use the square root).

Expressions (1) and (2) can easily be combined, yielding

$$\sigma = \sqrt{\frac{\sum (x - X)^2}{n - 1.382}} \qquad (3)$$

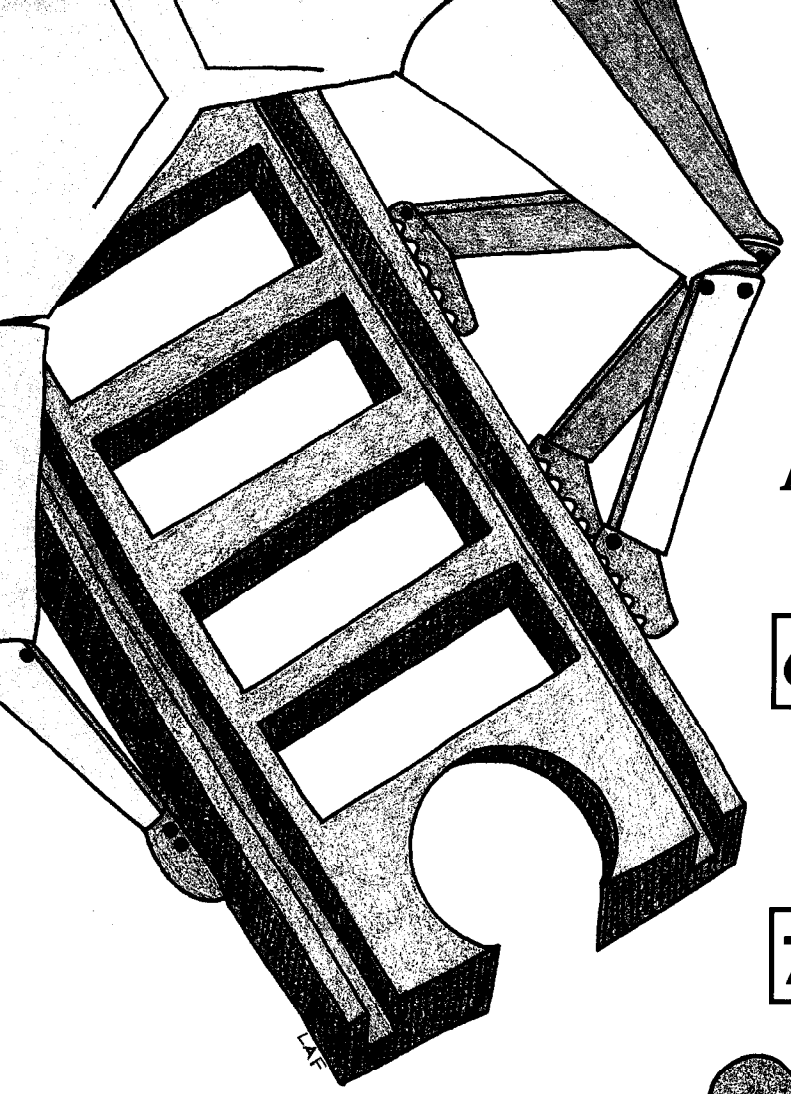
The standard deviation corrected for sample size by expression (3) differs from that based on the table values by less than 1.5%.

Equation (3) correctly indicates that no conclusion can be reached from a single sample, since the result becomes imaginary (i.e., has no "real" meaning). Since the expression is easy to use and remember, it recommends itself to general use in finding standard deviations.

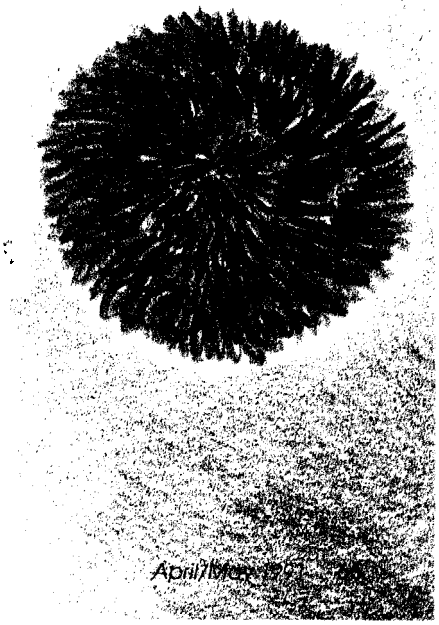
***Charles Boegliispresident** of **Randen Corporation in Blanchester,** Ohio. **Randen** is a small consulting/engineering company that specializes in interfacing computers to test and monitoringequipment,and **in analog** circuit design.*

IRS

4 19 Very Useful
420 Moderately Useful
421 Not Useful



Autorouters



FROM THE BENCH

Jeff Bachiochi

Working with an Autorouter

Integrating a New Tool into an Established Engineering Routine

Until only a few years ago, unless you had pencil and paper handy or owned a typewriter you couldn't document a thing. Word processors were nonexistent. Today, laptops are smaller than the average Royal and provide the perfect platform for word processing. Documentation is even easier with a host of features including spell and grammar checkers. I equate this kind of technology leap with the discovery of America or setting foot on the moon-accomplishments which were once only dreams. The power of the personal computer is weaving its way into the fabric of today's generation.

In the late '70s I bought my first word processor, "Electric Pencil," for the TRS-80. At the time, I was employed by Electronic Music Laboratories, a maker of music synthesizers. We produced our own single-sided (yup, real high-tech) circuit boards. There were no autorouting, PCB layout, or even schematic capture tools. Circuits were drawn on paper, laid out with Bishop Graphics' stick-on transfers and tape, and reduced 2:1 in the darkroom. Rubylith film was exposed to the artwork, developed, and applied to silk stretched tightly around a frame. Copper-clad fiberglass boards were screen printed with the artwork pattern and the unprinted copper etched off the board. Component holes weredrilled and theboards were sheared and prepped to protect the copper pads and traces. Many man hours went into each design.

When I started my present job, things hadn't progressed, except for one important fact. Specialty houses were springing up to handle particular aspects of the design job. Our designs were still drawn on paper (or napkins, place mats, or anything else handy at the time). But now we could pass the design off to a PC board design house. They would hand-tape the board and even produce the films necessary for the next phase: fabrication. This specialty house would "fab" the boards, which were now double-sided with plated through holes. Each step of the process was becoming more specialized. With this specializing came expertise, which meant a better product at a lesser cost.

COMPUTER AIDED DESIGN

The first piece of CAD software 1 tried was Tango PCB. (Keep in mind that this was prior to the availability of any schematic-capture software I was aware of.) As the project I was working on approached its deadline, we needed a PC layout quickly. I decided to give Tango a whirl (no pun intended). The component connections (traces) had to be laid in manually after partscreation and placement was finished. I was impressed with the fact that I could pick two pads on the board and the program would lay in a route automatically connecting them. These weresimpleroutes takingplace on only one side of the board at a time. With a bit of persistence you could strategically place vias and route back and forth between layers.

The disappointment came when I tried to create a DB-25 footprint. Unlike many standard parts, the DB-25 has unusual spacing: 0.108" between pins, 0.112" between rows, with one row staggered from the second. Well, IO-mil spacing was the smallest available,so thepadscouldn'tbeplaccdon the correct centers. I got around the problem by rou ting to a location where the pad should be and plotting without the DB-25 footprints. I added the DB-25 footprint to the films using the good old Bishop Graphics transfers. This was my first CAD-produced artwork and, incidentally, the last time I used Tango. (Newer versionsof Tango PCBdo handlegridslessthan 10mils.)

Ataboutthistimeourcircuitboard house decided to go for broke. After mortgaging every possession, they took delivery of a PC board CAD workstation. When I say they went out on a limb, I'm not exaggerating. I don't think they had ever seen a computeruntil thisequipment started to arrive. Support was outstanding, fortunately, and they were up and running inside a month!

It was not unusual for me to spend several days each month there with our normal design schedule. This gave me the opportunity to look over shoulders and see what the excitement was all about. I learned that the CAD station could accept input from a number of schematic-capture programs. This one bit of news thrust me permanently into the CAD age.

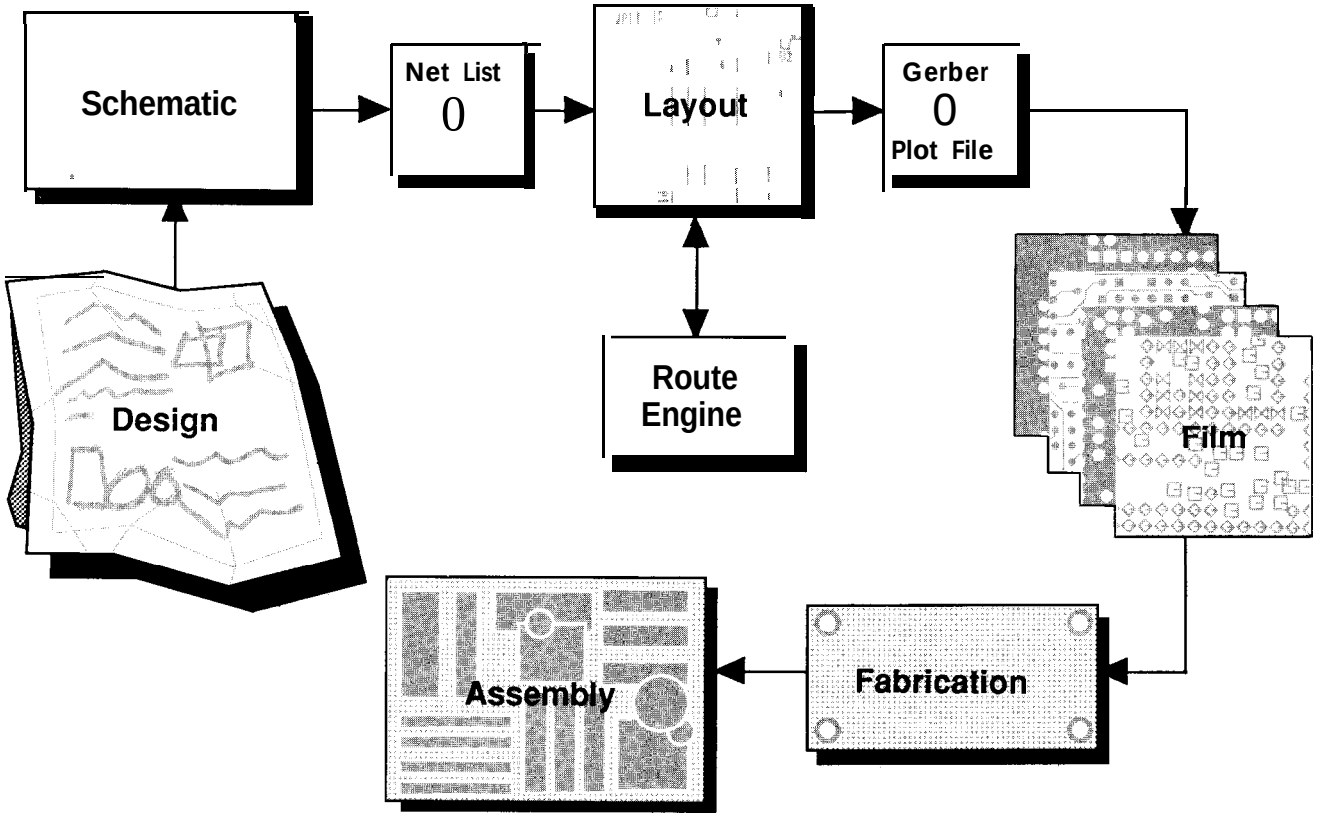
Back at the office I frantically made telephone calls, trying to find out where I could get my hands on a schematic-capture program. Within the week demo programs flooded my mail slot. One package stood out, due mostly to its ability to scroll smoothly within a drawing page. Many of you will recognize from this fact alone that I am talking about Schema from Omation.

JACK OF ALL TRADES vs. MASTER OF ONE

Our design group stays pretty busy with the design aspects of a product. This includes not only the circuitry but often the packaging of the product. In this age of specialty, it makes the most sense to let the specialists do what they do best. Many of our more complicated/compact designs require additional layers above and beyond the double-sided boards typically in use. This reinforces the need for specialty houses.

BUDDY CAN YOU SPARE A DIME?

So you say there is more to life than justdrawingpictures-you want total creative control? Well, the same machine that lets me document schematics now allows me to lay out PC boards.Two thousand dollarscanbuy you the minimum hardware, but don't be fooled; it's only the start. A bigger monitor, like the NEC MultiSync 5D, will show in the final design, so make the marriage work or switch designers or design houses.



At this time1 can't conceive trying tofabricateourownmultilayerboards. I generally don't get involved with selecting and qualifying a fab house because, once final films are made, the project is handed over to the production department. Prior to final films, however, a close bond between circuit designerandlayoutdesignerisamust. As circuit designer, you must convey a sense of flow, as well as nonobvious pit-falls in the circuit design. Parts placement is critical, affecting both form and function. If you aren't meshing well with the layout designer, it will cost twice that alone. Expanded/extended memory, printer/plotter, mouse, and a UPS will quickly drain your bank account. I haven't even mentioned the cost of software.. .

Yes, you can get your feet wet with a minimum amount of investment. But it won't be long until the size of the board being designed reaches the 640K DOS limit. At this point you pause in disbelief and start stripping out all of the memory-resident stuff that automatically loads in on power-up! You may have saved this design, but sooner or later 640K is going to be a problem. It's like buying a new pair of pants: you always seem to need the next larger size.

Let's step back for a moment and take a look at how schematic capture and layout/routing share information. Schematics are pictorial representationsofindividualgates,discreteparts, connectors, and so forth. Lines interconnecting the pictures represent electrical paths. The schematic program's output file, called a net list, is a description of each part and each net. The part is defined by name (e.g., resistor) and its reference designator (e.g., R1). A net is a list of all the component pins which are connected together. Net connections are listed by the reference designator followed by the pin number (e.g., IC1.6 [pin 6]).

The schematic capture's output net list file is the layout/router's inpu t file. The information is the same, however in the layout package the parts are displayed in their physical form as opposed to their symbolic form (as they are in the schematic). The nets are temporarily displayed as simple"shortestdistance"conncctions (often called a "rat's nest" display).

FOUR ON THE FLOOR

The first part of the layout design is the physical mechanics of the circuit board, including board outline and mounting holes. I save the empty board with a file name such as BOARD1. This allows me to use it again in another design without having to redraw it. At this point the net list can be read in. Parts not in the PCB library are flagged. These should be created before proceeding.

I don't find autoplacing of components very useful. This seems to cause more problems in a design then the time it saves. I prefer placing the ICs on a board in some logical order by using the rat's nest as a visual indication of signal flow. Connections are made daisy-chain style, in the order in which they fall in the net list. For instance,anet consistingof IC1.l (pin 1), IC2.2 (pin 2), and IC3.3 (pin 3) are all electrically connected. If ICI is moved, pin 1 will stay connected to IC2.2 even if it's physically closer to IC3.3. As parts are moved around the connectionscrisscrossina jumbled mess.The worse the rat's nest, the more difficult the routing will be.

The best connection pattern is often the shortest distance and not necessarily the order in which the list was read in. Here is where the computer can speed things up. Each net is analyzed to find the shortest connection path to all points within the net. This is not routing, merely a minimizing of overall net lengths, whichredistributes the connections in the daisy chain.

A constant arranging and rearranging of parts is necessary until all parts are placed on the board in such a way as to minimize rat's nesting as much as possible. Each crossing of a net will most likely require a via. Since vias take up space, result in a potential weak point, and increase board costs, I keep them to a minimum.

I spend about 25% of the total PCB design time in layout. It is a crucial step in the outcome of the final product. When satisfied with the parts placement, I save the file again, under a new name (BOARD2). If the route does not go well, I can get back to a clean slate easily.

WHICH ROUTE TO TAKE?

Multilayer boards (those with more than two sides), usually have internal power and ground planes, so routing power and ground is unnecessary. On double-sided boards, though, they must be routed, and I prefer to route them first. In general, I use 30-50 mil traces. This means a trace will not fit between ICs or other components with 0.1" lead spacing, but they will fit nicely through the channel beneath ICs. Therefore, keeping the ICs in nice even rows makes for cleaner routes. The ability to select which nets are routed is helpful because there are different algorithms for each type of circuitry. Power/ground, memory, digital, and analog circuitry are all handled differently.

The manual router, which is part of the Schema PCB package, will allow you to hand route your board.

The autorouting option is a cost-effective addition. I find most boards will route from 80% to 100% completion depending on board density. Board density is the total board area divided by the equivalent number of 14-pin ICs. To do this, all the parts on the board are converted into an equivalent number of 14-pin ICs. A 14-pin IC, which measures 0.4" x 0.8" is 0.32 square inches or about one-third of a square inch. If three of these (or their equivalent) were on a l-square-inch board, the density would be 1 divided by 3 or 0.33, which is pretty dense. On the other hand, if only one 14-pin IC was on that square inch, the density would be 1 divided by 1 or 1. I find this router good for densitiesof 1 or greater, and OK down to about 0.5.

Of course you can't go by density alone; the number of nets and complexity of rat's nesting will alter the equation. Percentage completion can be a real nasty number. On a 100-net board, 90% completion means having to hand route the remaining 10 nets. On a 1000-net board, 90% completion means 100 nets are left unrouted. If you could lay in one manual route every six minutes, that's 10 hours for 100 routes-and that's a very optimistic estimate. When you get down to the routes the machine can't place, you're talking "bottom of the barrel." I've had routes which took multiple hours to place by hand (which does include interruptions). This is where a good push-and-shove interactive router could aid in the hand routing. Unfortunately, I don't find the Schema push-and-shove router useful at this stage of the layout where things are really getting tight.

If you want to spend "thouand of dollar$," you can purchase a route engine. Route engines differ a bit from the routers associated with layout packages in that they have no layout capabilities. A route engine will do nothing but route the board. If it gets stuck, it will remove routes and try alternate possibilities. Because most route engines use a "rip-up-and-retry" algorithm, percentage completion is much higher, but also the time required to try all its possibilities can be-lengthy. You must weigh your needs to determine if your designs are large enough in size and high enough in density to warrant the bucks. Once the route engine finishes, any routes left uncompleted will be extremely difficult to hand route. You might find the routeengine has simply used every available channel.

It is not unusual for output produced by one package to be incompatible with another. Some conversions are standard, others optional, and still others unavailable. Buying a completely integrated package will keep frustration levels down to a minimum.

Once1 have totally routed aboard, I spend some time cleaning up the routes. Sometimes routers do some strange things and you need to check the routing for manufacturability. Traces should leave pads perpendicular to the wave used to solder the boards. Routes should be centered or evenly spaced between component pads with a minimum of stair-stepping.

WYSIWYG-SEEING IS BELIEVING

I used to plot the artwork at 2:1 on a dot-matrix printer, then started using an X-Y plotter. Films were produced by reducing the plots down to actual size. Now, I generate Gerber files and use a software package called GerberJet to print them on an HP LaserJet for review: I just want one final confidence check before committing to film. If all is OK, I send the files by modem to the design house for plotting on their laser plotter.

CHECK THE SPECS

The first thing to look at in a CAD package is: What is the minimum hardware needed to make good use of the product? What higher end hardware does it support? This includes input devices (mouse, keyboard, tablet, light pen, etc.) and output devices (display board, monitor, printer, plotter, etc.). What size design will fit into the standard 640K memory? Is there support for extended or expanded memory and a coprocessor?

With regards to autorouters, the most important point is that of compatibility. Make sure you can go back and forth between layout and routing without(oratleastaminimumamount of) file conversion. If you are interested in multilayer designs, determine themaximum number of layers which can be simultaneously routed. Make sure that you don't have to play tricks to get good power and ground plots with thermal relief pads.

Review the routing grids, pad, track and via sizes supported. For those unusual components, look for "off-grid" support and the ability to route SMT (surface mount) components. Other featuresconsist of "keep-outs," 45" or radius cornering, copper sharing, ground planing, and route gluing. One of the most flexible, yet most complex aspects of routing are the strategy parameters. User-configurable parameters such as the depth of a rip up and retry tree search, obstacle hugging, high-density avoidance, route direction weight, and via cost versus trace length, will each have an effect on the routing algorithms.

It is hard to resist dabbling in the PCB layout design function. It does require that a good deal of time be invested before a well-designed board can be **produced** with any proficiency. Therefore, I suggest a design service bureau be used whenever possible. This allows capital equipment costs to stay low and also allows you to keep your energies channeled in the direction which will be most profitable to your organization. On the other hand, if you must keep the total design in-house...

Before you buy, compare specs, get demo diskettes, and if possible, ask vendors for si tes in your area where their software packages can be seen in use. A picture is worth a thousand words.+

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

Curtis Franklin. Jr.

Bringing in the Pros

Working with a Board Design Firm

I don't care how powerful the system is—if Yo" don't have adesignerthatknows how to make the system work and knows how to design a board as if relates to production and manufacturing, fhe system will do you no good. When you get down to it, the designer is the power ible

Personal computer autorouters are becoming more powerful and competent every day, but many design engineers still turn to a professional board design firm for final board layout and film production. CIRCUIT CELLAR INK's Curtis Franklin went to Custom Photo and Design and talked with Raymond Long, the firm's president. Mr. Long discussed some of the services offered by a modern design firm, and talked about the process of working witha professional board designer.

When you're workingwith an average client, what do you expect from them and what do you deliver?

When we design a board for a customer, he comes in with a schematic and a physical board outline. At this point we will give him a quote for the job. Based on the customer's requirementsand needs, we will either do the schematic capture and net list generation ourselves or begin the board design based on work he's already done.

So would you perform [schematic capture1 if they eithergave you a handdrawn schematic or something that came off of a system that's incompatwith your equipment?

We have a lot of customers out there who generate schematics on Autocad. Forget any electrical continuity checks or design rules, but it gives them a nice pretty schematic. Autocad schematics are not compatible with our system and obviously hand-drawn schematics are not compatible. We have the ability in-house to generate a schematic on Schema, Orcad, or the Cadnetix system. Once we've done that we submit the penplotted schematic) to the customer for line checking. That's for his peace of mind and to check us: After all, we're human and we make errors too. From the schematic we generate a net list and continue with the board.

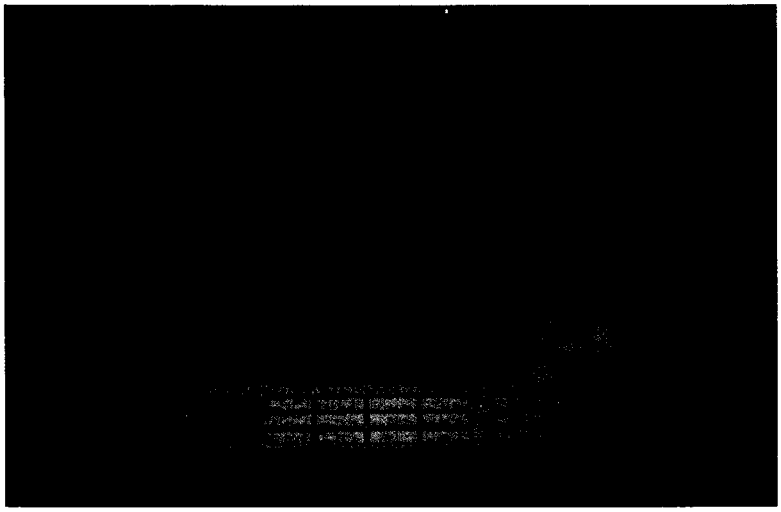
In a service bureau environment, one of the toughest things for us to get is a spec sheet-we don't buy components. We request, from the customer, spec sheets or samples of the parts that he is using on the board. Primarily, we need specs for capacitors, switches, connectors, or pots. We don't need the standard ICs, resistors, and diodes. We have pretty much all the cross-reference books on those, but we prefer to have something on capacitors because they can run a wide range of sizes.

Next, while the schematic is being drawn and reviewed, the designer is building the parts, the board outline, and the pad stacks to the technology that the customer wants. For example, he may need a high-production autoinsert board, or a mil-spec board. (To be honest with you, we design all of our boards to a mil-spec requirement at no extra cost because it's easier to design to a high standard than it is to vary back and forth. People think it's going to cost extra because you're designing from a mil-standard, but it isn't, especially if everybody is geared to the mil-standard.)

Once the board outline is built, the shapes and the components are built, schematics are drawn, and the net list is generated. Then we merge the net list to the design file and do a parts placement.

The placement, for many boards, is very, very critical. At this point we either request that the engineer on the

job sit down with the designer or go through critical placement using the **placement** aid tools. Using the schematic and input from the engineer we will come up with a final placement. We use no auto placement software on our systems-it just doesn't lend itself to a good design. There are some fairly slick placement packages out there but the human intervention in parts placement [is vital].



An Intergraph System is used to control the final loser plotting of film. It is linked to the design and routing systems by a high-speed local area network.

More and more people are using surface mount technology. Are you finding that it is having a big effect on parts placement?

The problem with surface mount is conversion from standard through-hole technology to surface mount components. At one time there was a problem with the footprint for the surface mount components, but the IPC (Institute for Interconnecting and Packaging Electronic Circuits) has stepped in and is starting to regulate the surface mount footprint pads.

In the early days of surface mount, Hitachi would design an IC and would have their own surface mount pattern. In order to lock the customer into that part they'd have a custom pad. Even the difference between infrared and vapor-phase soldering determined which part you used, how you designed a board, and what pad or footprint you put down for the part. Now, the IPC has stepped in and most companies around the world have standardized so that you have a universal pad for any given part. Most companies are submitting their footprints to IPC before they put out their recommendations and standards.

We are the only service bureau in the country that is directly associated with the IPC. One of the employees here is on the IPC committee to rewrite Mil Standard 275. In fact, he is in California now because the IPC is releasing the IPC D-275 standard and he has to be there for that release. The government is getting out of specification writing and they are turning it all over to the IPC. Every designer in here is up to the IPC standards-the new standards for designing boards.

After the placement phase, what happens?

After we do placement we then manually route in any super-critical lines, restricted areas, or unusual features. We determine at this point whether the board is going to be controlled impedance. Impedance control is becoming more and more common and has to be designed into the board because a combination of design technology **and** board fabrication helps you maintain your impedance in a given line. If you don't do certain things [in the design phase] you can't test for control impedance-it's impossible. Those things are automatically factored in. One of the reasons we can do this is that we have custom software here that was written using data compiled from dozens and dozens of production boards, then used to set the parameters for the line requirements. The line thicknesses take into account the production etch-back factors, process allowances, and other factors, so that you can hit the number right on the head.

After these items have been considered and taken into account, we have all critical lines and short runs in, and all those lines are locked down, we then go into our route engines. The technologies and routing strategies the designers have developed over the years determine how well the route engine works.

THE ROUTE ENGINES

A route engine is not going to work and give you a high-quality product unless you interact with it on a regular basis-it's just like anyothercomputer. An example is the Cadnetix route engine-it's an extremely sophisticated route engine. Our designers work on it regularly and spent a lot of time up front working on the strategies. They have a very good understanding of how it works and we get more high-quality, consistent, 100% routesonour engines than a company that hasn't spent the time to develop the strategies would.

The aesthetic value of a board is almost as critical as the functional value because your customer is going to see the end product. If the board aesthetically looks like a piece of junk, it doesn't make any difference how well it works-thecustomer will think it's a piece of junk.

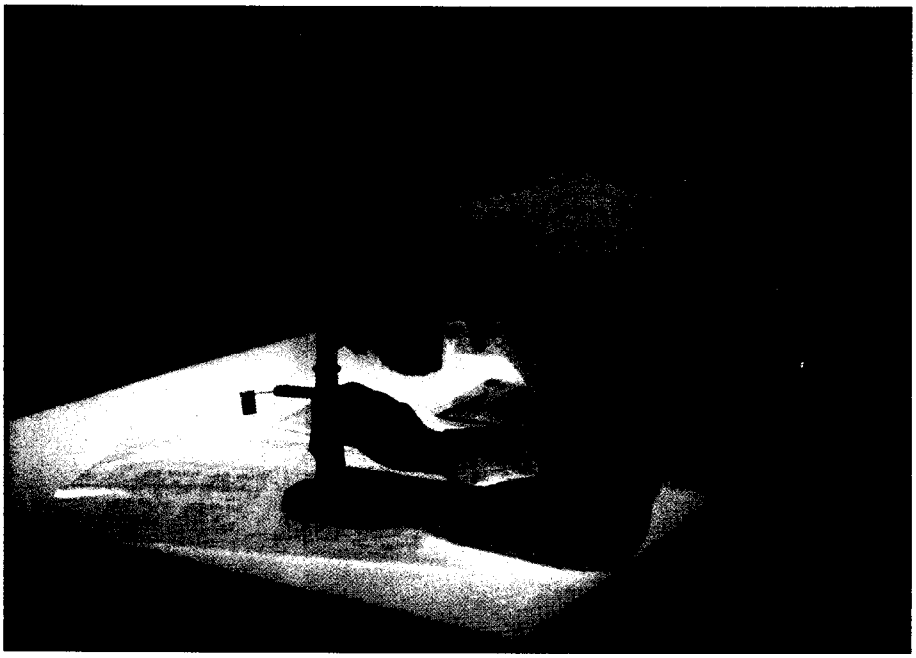
We had a problem with our route engine and went to one of our customers, an in-house operation, and leased time from them. We used their route engine as it related to the Cadnetix system. When we went down and sat with their designers, they were having a problem routing a board on their route engine. These guys were fairly experienced, but there was a production "get it through-get it out-get it fast" [mentality] and our designer was able to educate them on some of the technologies. Now, they are getting higher percentage routes than ever before, based on what our designer taught them. The CAD houses can't really teach you the routing strategies-you have to get in there and hack and play and experiment.

Our special section is on ***autorouting but I notice you using the phrase*** route ***engine. What is the difference between a route engine you use and the autorouters available on PCs?***

A route engine is an off-line computer dedicated to routing boards. It runs autorouter software.

There are several different types of autorouters. You've got rip up replace, push and shove, and others. The complexity of the board determines what type of router you use. Basically they are all autorouters because your are not getting in there and stitching each item by hand.

[Routeengineslareextremelyfast. On the Cadnetix, we did a route that was 30 ICs and 30 discretes—it routed 100% in six seconds.



Human intervention is an important part of the process. from design androuting to (shown here) the physical production of film.

What additional time was spent for ***the cleanup process?***

I think the cleanup took two or three minutes; not more than that.

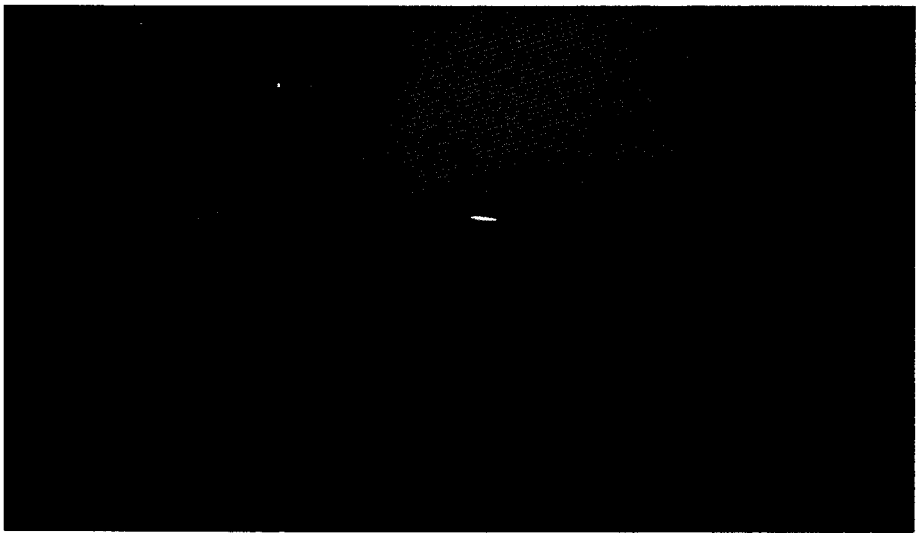
Speed is one of the big reasons for the power that we have in here. Most of your three- or four-thousand dollar route packages don't do the via minimization that the big powerful engines do. Let's face it, every hole you drill on that board is going to cost you money. Every via costs you x amount: If you can eliminate 150 vias on a board in fabrication, each costing one-half cent apiece, you can save a lot of money if you're making a hundred

thousand boards. Also a via is a weak link-it's just anotherlinkin thechain and if you can take that link out and make a direct connect you have a more reliable board. Via minimization is a tough one for these small packages.

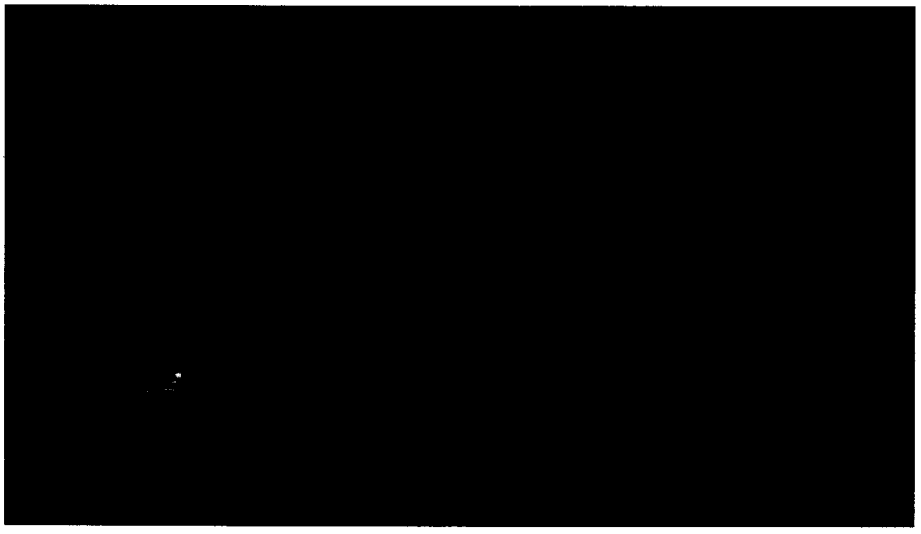
After we pull the board out of the routeengineandit'sdoneallitsmagic, we then go in and do interactive clean up. We eliminate much of the stair-stepping that is automatically put in by route engines. We do additional via minimization and we do pad centering.

A lot of the hand work is clean up for manufacturability-via minimization and the aesthetic value of the board. We eliminate the tangent connections to a pad wherever we can, primarily because they are weak links that could be questioned at any point in time. You don't want engineers and fabricators asking whether or not a connection is correct-whether it is a short or a good connection. It's just a lot of visual peace-of-mind not only to the customer and his engineers but to the fabricatorsif it'sa good fabricating house.

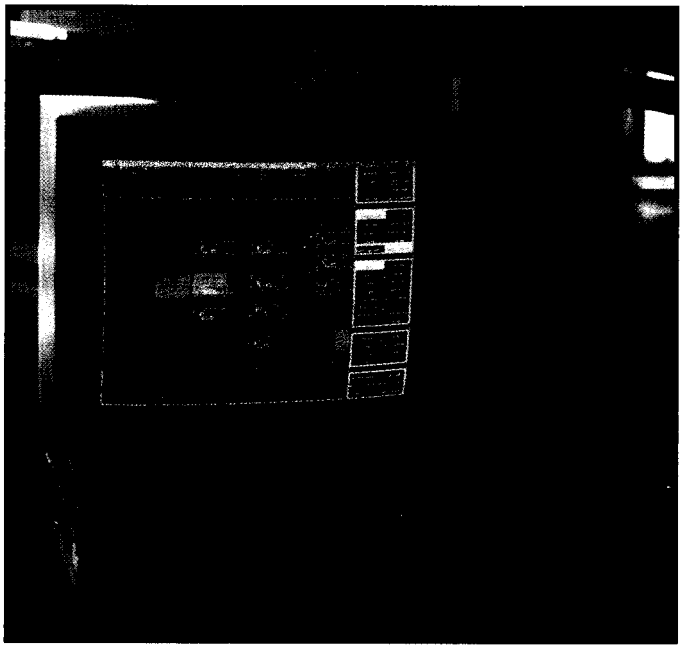
Once the cleanup is done, we then plot check prints on a 2:1 scale and submit them to the customer. Check plots are thengone over very carefully by the customer. If he hasanyrequests for changes, they are made at no cost to the customer at that point. There isn't a printed circuit board designed that doesn't have at least one change to it. Once the changes are completed, we go through and write the book on the silk screen and do the silk screen drawings. We create the drill tape, fab drawings,doourlaser-photo plotting, and the job is released.



Pen-plotted versions of the design, at 2:1 enlargement, are checked and approved by customers before final films are laser plotted.



Laser plotters and their operators work in the constant ruby twilight of a light-safe area.



The latest Calay systems use Sun Sparcstations as platforms. Each station in the system costs as much as 10 well-equipped 80386-based PCs.

TIMING IS EVERYTHING

Normally, what kind of times would be involved in a cycle like that?

Ournormaldeliveryis two to three weeks.

That is from first information in the door until the job is released?

Right-and there are stop times involved there. Now [two to three

weeks] is our normal turn-around time. That gives **us** a real delivery date that we can look for. In that two- to three-week time is a "clock-stopping" mode-actually, two and possibly three clock stops once we submit the schematic. While that may sound like a lot of dead time, things usually proceed pretty smoothly because we're building the parts for the board and the board outline while the engineer is checking the schematics.

If the engineer does his job and turns that schematic around, there is no stopping involved. But if we get to a point that we have the parts and board built, and the engineers are lagging behind us, we're at a dead standstill. We can't do anything because we don't know whether the net list is any good.

A lot a times we will begin to use the net list because we're sure it's at least 95-99% correct. We will then take the unapproved but still pretty good net list and load it in. We'll use it as a "go by" just to start the placement so that we don't lose time on the job. The engineer may have been pulled off to do something else and his priorities change for internal reasons. We try to go as far as we possibly can without stopping the job.

On many jobs, the engineer requires an interface with us in placement and critical routing. If he's available when we are, there is no stopping involved. If he's not available, then the job will stop because it's fruitless to go on without the input. The only true stop point that is built into this schedule is the check plot stage. Once we have sent the customer the check plot, we do not touch that job again until we hear from the customer.

Once we have the customer changes in hand, within 24-36 hours we are ready to be in fabrication. I'm not talking about "working hours," where 24 hours can stretch over three or four days, I'm talking about actual hours.

Engineering changes take precedence over new work within our environment. The reason is [that, with] the power of theequipment we have here, changes do not take an awful lot of time. We are cognizant of the fact that the engineer has already prototyped and tested, he's troubleshot and he is ready to go to production, and the guy that we're designing the new board for probably doesn't have all his parts yet because he's at least two weeks away, on a normal turn around time, from being ready to build a prototype. The changes are made almost instantaneously on the CAD system

What about the changes that are made after the design has been in production for a while? *Do you* ***have to go back and start over?***

We maintain an archive file of all the boards that are now, or ever have been, on the system. Let's say that, three years ago, we designed a board for somebody. If the customer changes that board we can go pull it up on the system and make whatever changes he requires to it. We feel this adds a large factor of service and convenience to our customers. There is **a** safety

factor involved here, too, because we can hand a customer his database at any time. We don't ever want a customer to feel that we are holding his design hostage to our policies or technology. Of course, if he goes and throws it into a PC and makes changes to it himself, we have no way of knowing that-the design and all the ramifications of the changes he makes become his responsibility at that point.

IS THE PRICE RIGHT?

*How **do you price your services? Are there standard charges based on size, or is pricing s trictly on an individual case basis?***

We had to come up with a standard quoting that was fair and equitable. We've found that everybody has their own standard for quoting. Now, some people use a variable factor. I use a standard factor [based on the number of viasl. believe it or not the toughest board to do on these CAD systems, and you can ask any one of the designers, is a singlcside power supply board. They are tougher to do on these systems because the systems aren't designed to do that type of layout-a large, simple board doesn't "play to the strength,, of the route engine. With a route engine, a tough board to route is an 8"×10" board with five ICs. A board with a layout that open will be considerably tougher to route than an 8" x 10" board with 80 ICs. The reason is that the route engine has so much room to work, and so many options for individual traces, it becomes overwhelmed by all the possibilities.

Too many possibilities?

Yeah-if you tie the specifications the autorouter works from down real tight and keep them tight, the design will route 10 times faster. A small factor that has to be inserted here, too, is that there are many people out there that have some of the equipment that we have here, but equipment is only as good as the operator.

NO SUBSTITUTE FOR EXPERIENCE

Let me **give you** an example of the difference experience can make. A customer came in with a database for a board that had been designed by another firm. The reason the customer came here is that he had manufacturing problems that were unbelievable. He got the database back from the original designer. We brought it up on our screen and the designer had routed two tracks between pins. With that type of technology **on** a 4-layer board—90% of the pass throughs between the pins were on the solder side of the card and not the component side! We went in and checked the technology that came with the database: The gentleman was running two tracks between-we checked them out and the technology said they were 11-mil tracks running between 0.100" center pin ICs with 0.060" pads—and he had proper spacing. It set us back a touch because we did a design rule check and everything worked, even though our experienceand common sense told

us that the board could not be fabricated as it was laid out.

We started poking into the technology specified in the database and found that the original designers didn't know how to set the technology tables up. They designed the board at 1.25:1 scale so they could use the default technology of their CAD system. Then they post-processed at a reduction scale factor and played games with the aperture list so they could photoplot it and get a product out

We totally understood now what this designer did and we couldn't believe it. We wanted to help the customer out but there was absolutely none of the previous design that was workable according to our standards. We ultimately redesigned the board from the ground up.

When we redesigned the board we used the proper technology and theproperroutingstrategy. We added three ICs to the board, repackaged it, rerouted it, and ended up using single pass-through technology instead of two-between technology. Since WC put all the pass throughs on the right side of the board, we had fewer vias on the board than the original design. The customer put that board into a production mode and hasn't had one problem since.

It sounds like you're saying that, while the route engine may be the fastest part of the system, it's not the most powerful.

I don't care how powerful the system is-if you don't have a designer that knows how to make the system work and knows how to design a board as it relates to production and manufacturing, the system will do you no good. When you get down to it, the designer is the power behind the system.+

Curtis Franklin, Jr. is the Editor-in-Chief of Circuit Cellar INK.

IRS

425 Very Useful
426 Moderately Useful
427 Not Useful

Call for Manuscripts

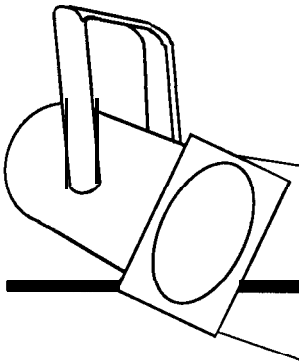
Circuit Cellar INK is looking for quality manuscripts on software for embedded control, software applications, advanced algorithms, and tutorials on tools and techniques for developing software.

These manuscripts will be considered for publication in Circuit Cellar INK. The Computer Applications Journal. and in a planned series of books to be published by CCINK.

Circuit Cellar INK offers writers and engineers a technically sophisticated audience and professional editorial guidance.

The CCINK Author's Guide is available for downloading from the Circuit Cellar BBS. Prospective authors may send mail to "Curt Franklin' on the BBS, or send proposals for manuscripts and requests for Author's Guides to:

Curtis Franklin, Jr.
Editor-in-Chief
Circuit Cellar INK
4 Park Street
Vernon, CT 06066



FEATURE ARTICLE *R. Scott Coppersmith*

A PC-Controlled Light Show

A Parallel-Communication Lighting Board with MIDI Potential

This project started when a band asked me to build a low-cost light show that could be interfaced with their MIDI equipment. The basic stage layout they wanted to use is shown in Figure 1. They wanted a controller for eight 300-500-W flood lamps: This controller would connect to both their computer and their musical instruments. The project has since evolved into 'this 8-channel light show controlled by a PC parallel port, direct audio source, or a combination of both.

The hardware is divided into three sections. The first is the control panel that interfaces directly to the PC and audio source. The second is the power module thatcontains triacs and most of the AC wiring. The third consists of two 4-outlet 110-VAC power boxes, each with its own power plug. The separate power plugs make it possible to double the available wattage usable by the system.

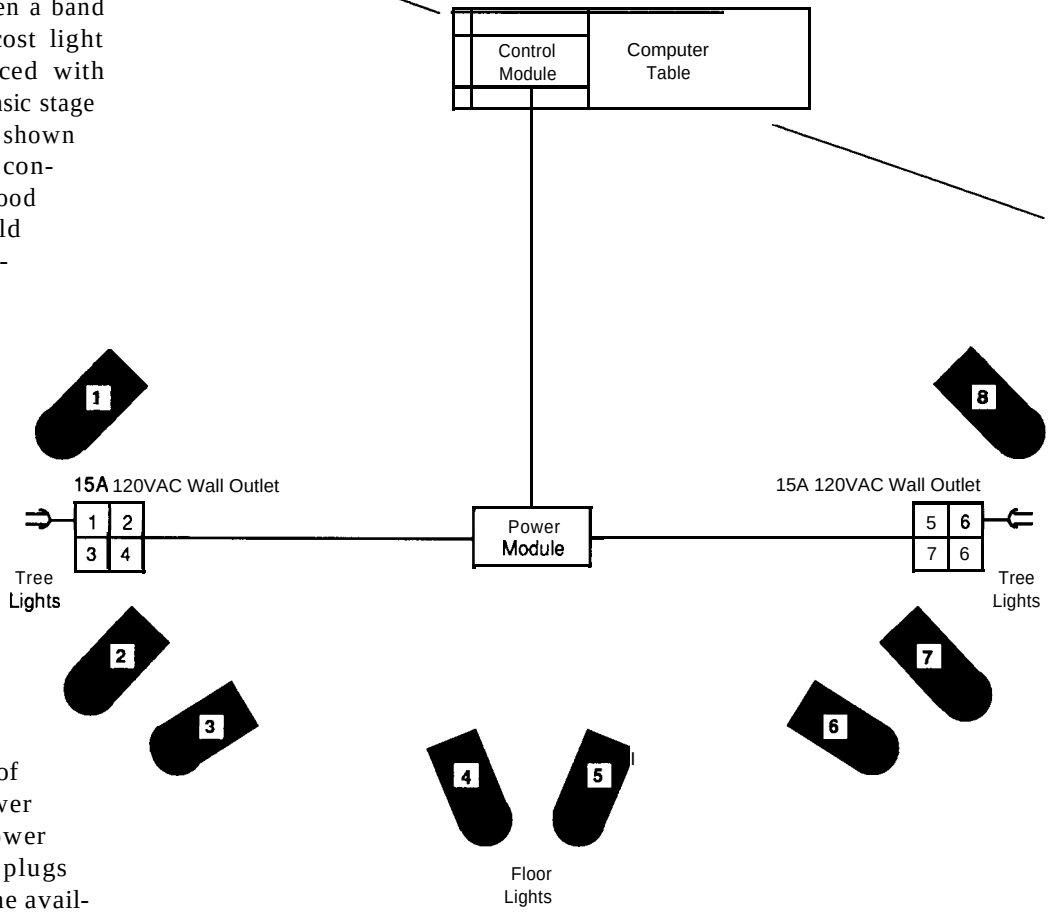


Figure 1 *-A basic layout for stage lighting can be controlled with the simple controller described here. Control can be directed via PC parallel port, music source, or both.*

CONTROL PANEL

The schema tic for the control panel is shown in Figure 2. The computer interface is straightforward. Connector P1 goes to the computer's parallel port. Pins 2-9 are data lines DO-D7. Pin 1 (*STROBE) latches the data into the 74LS573 and is also tied to pin 10 (*ACK). Pin 11 (BUSY) is tied todigital ground along with pins 18-25. Pins 12-17 are not used. The 74LS541 is used as a buffer and as one more layer of protection for the computer against the various high voltages in the system.

The other input to the control panel is a standard 1/4" phone jack. This input is for the audio source. The audio output from a drum machine can connect directly to this input with a patch cord. The output from a microphone preamp will also drive this input. This input is the voltage source for the LED bargraph display driver UAA180. The UAA180 is manufactured by Siemens, and is used in some automotive graphic equalizers. There are other choices for this driver, depending on how you want the lights to operate. Drivers are available with either linear or log outputs, and single LED or multiple LED configurations. I chose the UAA180 be-

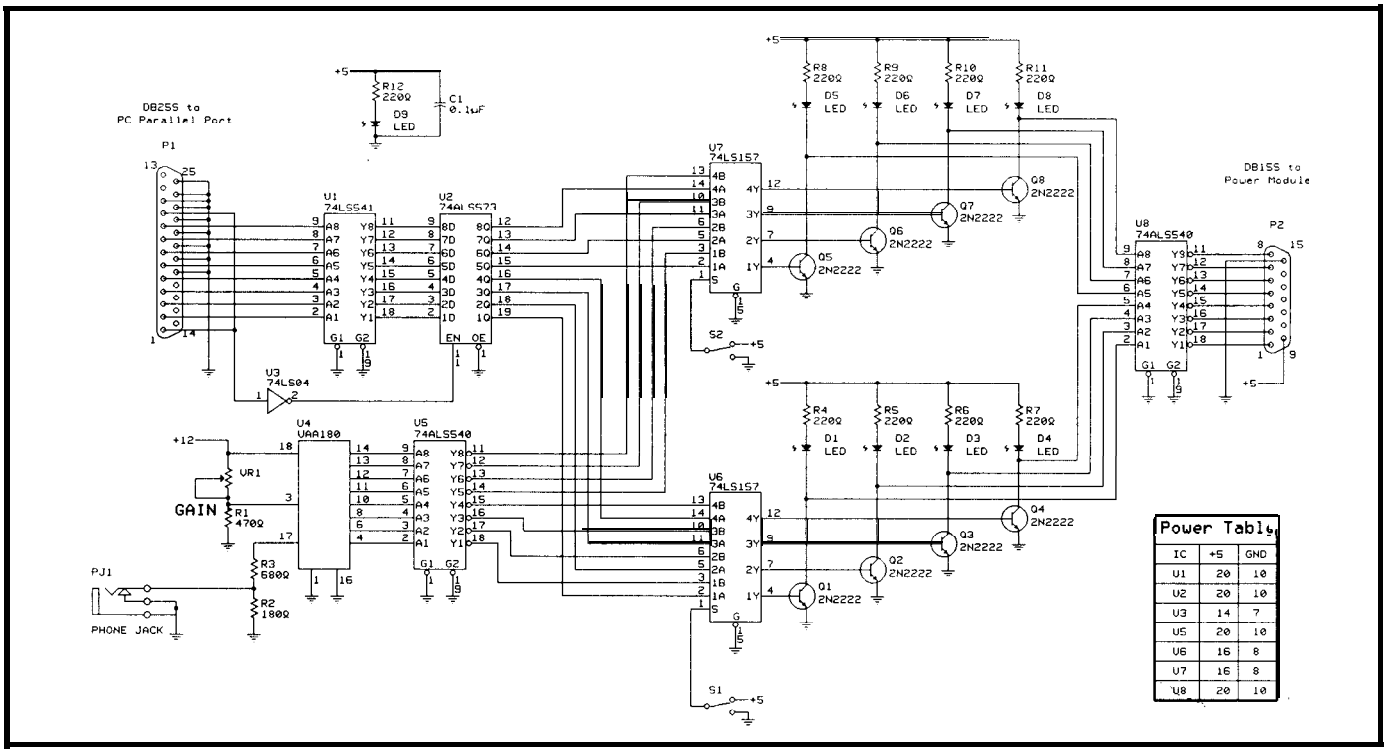


Figure 2—*The control panel* ***for the lighting controller. Eight lighting channels are grouped into two sets of four for control*** *purposes.* ***An LED bargraph provides visual feedback.***

cause I had a broken graphic equalizer lying around that I could use for parts. The 74LS540 is used to buffer the driver's outputs, and in this case to invert the signals. A 74LS541 can be substituted if you don't want the signals inverted. A gain control (VR1) for the audio source is mounted on the front panel and is used to select the channel limits for this option.

The eight channels are grouped into two sets of four. Two 74LS157s

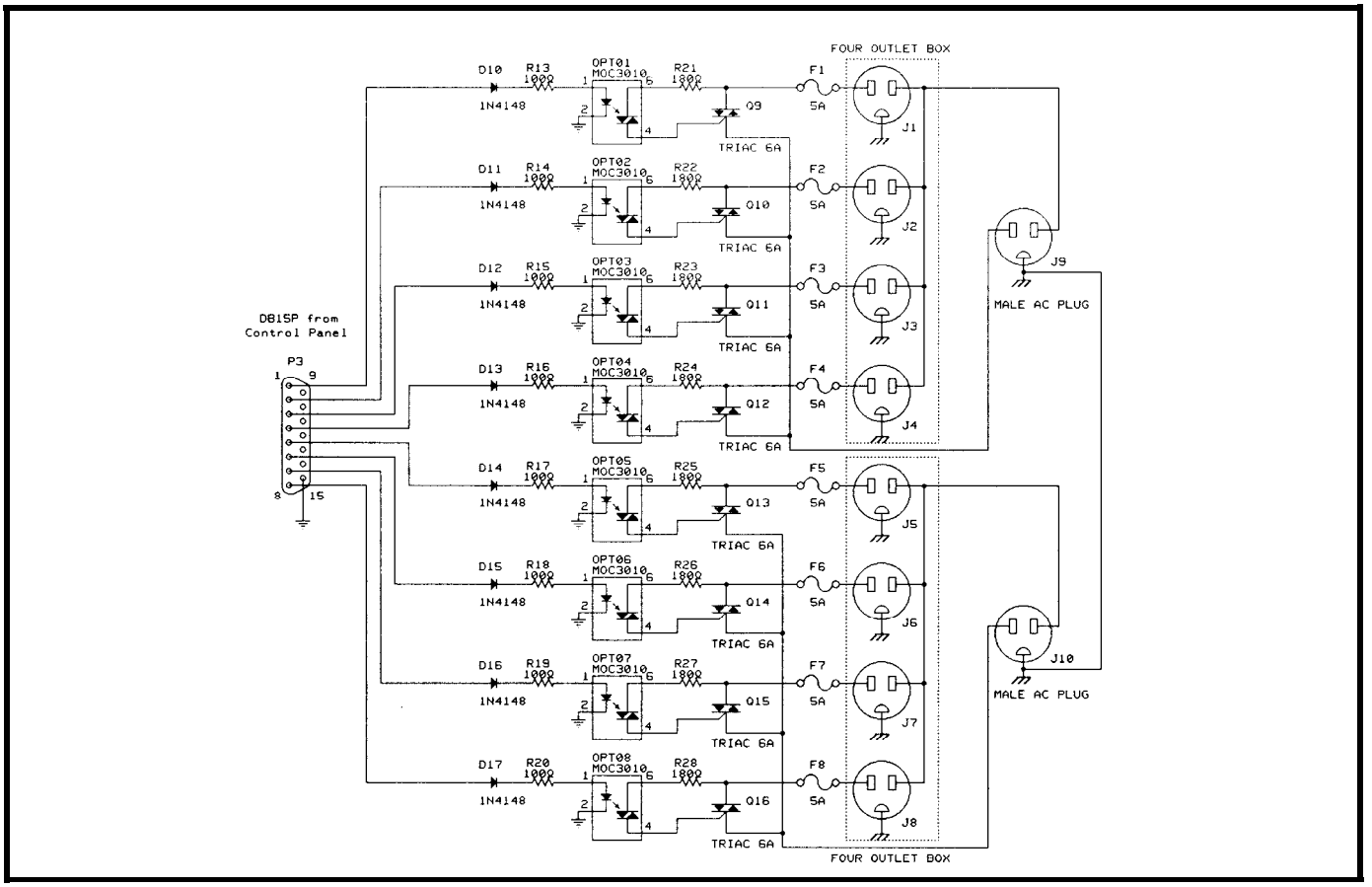
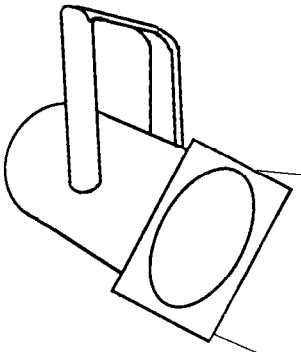


Figure 3—*The* ***power module of the lighting controller. Filtering and grounding are important considerations when lighting and audio equipment are being powered from common circuits.***



are used to switch each bank of four channels to either the computer data lines or to the audio source. Two SPDT switches are mounted on the front panel to select the controller input source.

There are nine LEDs on the control panel. D1–D8 are used as channel "ON/OFF" indicators. D9 is used for the 5-V power. The value of resistors R4–R12 should be chosen to correctly match the forward current needed for your particular LEDs.

The last IC (U8) is another inverter/buffer to drive the power module through pins l-8 of a 15-pin D connector. This enables the control panel and power module to be separated by several feet of cable. Pin 9 of this connector is tied to 5 V for diagnostic use only. The power module does not need this voltage to operate, but it helped to have it there to test the incoming data lines using a logic probe. Pin 15 is tied to digital ground.

POWER MODULE

The schematic for the power module is shown in Figure 3. The power module input data lines are protected by diodes D10–D17. Decoupling capacitors may be needed at the connectors if a noise problem develops between the control panel and the power module. The length of your cables and the drive current for the optoisolators will determine the correct values for resistors R13–R20. These values may need to be adjusted for optimum performance. OPTO1–OPT08 are triac output optoisolators. The ground connections for these devices should be connected to digital ground. The outputs from the optoisolators drive 6-amp triacs through a 180-ohm resistor. Each channel has a 5-amp fuse in line, and wire gauges become important from this stage on.

Here the system splits into two 4-channel subsystems. Hot lines from the 110 VAC are common for each of the four channels of a set, but separate between the two subsystems so that a separate 110-VAC outlet can be used to power four channels. If there are two available 15-/20-amp outlets on separate breakers, the system can handle twice the wattage in lights. If dimmers are available, each subsystem can be dimmed separately. To ensure a safe system, each 4-outlet metal box should be tied to earth ground. Do not tie digital ground to earth ground, and make sure all wiring codes are satisfied for your area.

SIMPLY SOFTWARE

Controlling the system is a programmer's paradise. With the exception of two short assembly routines, the programming for the light show can be in whatever language you are familiar with. *[Editor's Note: Complete* software for *this article is available on Software On Disk #20 or on the Circuit Cellar BBS. See page 91 for ordering and downloading information.]*

The first assembly routine is called `lpt1 reset`. It is called once at the beginning of your program and again at the end of your program to reset the computer parallel port.

The second assembly routine, `lpt1 data`, is used to send data out to the parallel port.

APPLICATIONS

Your controller application is limited only by your imagination. It can be used as a light show for your musical group, a Christmas light controller, an enhancement for your home audio system, or a general-purpose appliance controller. One setup that works well is to use a drum machine input for control of four channels, and then set the computer for random lights on the other four channels. Another possibility is to control the power module directly from the LED drivers from a graphic equalizer. Sample programs `paraport.c` and the compiled version `paraport.exe` are available to test your system.

```
/* resetpt1.c */
#include <stdio.h>
#pragma inline
void lpt1_reset()
{    /*save registers */
  asm push ax
  asm  push dx
     /* clear ax */
  asm  xor  ax, ax
     /* ah = 01 reset cmd */
  asm  mov ah,01
     /* dx = 00 for LPT1 */
     /* dx = 01 for LPT2 */
  asm  mov dx,00
     /* call interrupt 17h */
  asm  int 17h
     /* restore registers */
  asm pop dx
  asm pop  ax  }
```

listing 1 —*lpt1_reset* **does** *all the* **parallel port** *initialization.*

MIDI

For a MIDI application, you will either have to write an interrupt routine to monitor your MIDI card address, or write your own custom MIDI composer software with routines to use `lpt1_out` for a specific MIDI channel. If you need help with MIDI applications, there are many talented people on the MIDI forum on CompuServe willing to help.

```
/* lpt1out.c */
#include <stdio.h>
#pragma inline
void lpt1_data(unsigned
                    char  data)
{    /* save registers */
  asm  push ax
  asm  push dx
     /* clear ax */
  asm xor  ax,ax
     /* move data into al */
  asm  mov al, data
     /* dx = 00 for LPT1 */
     /* dx = 01 for LPT2 */
  asm  mov dx,00
     /* call interrupt 17h */
  asm  int 17h
     /* restore registers */
  asm pop dx
  asm pop  ax  }
```

listing 2—*All dealings with the parallel port are* done through *lpt1_data.*

Scott Coppersmith holds a B.S. in Electrical Engineering from Michigan Tech. University. He is currently on contract to Chrysler Motor Powertrain Electronics Division. He is also a part-time Turbo C programmer.

FEATURE ARTICLE

Ernest Stiltner

Working with Zeropower SRAM

Building ZMEM, The Zeropower Memory Chip Programmer

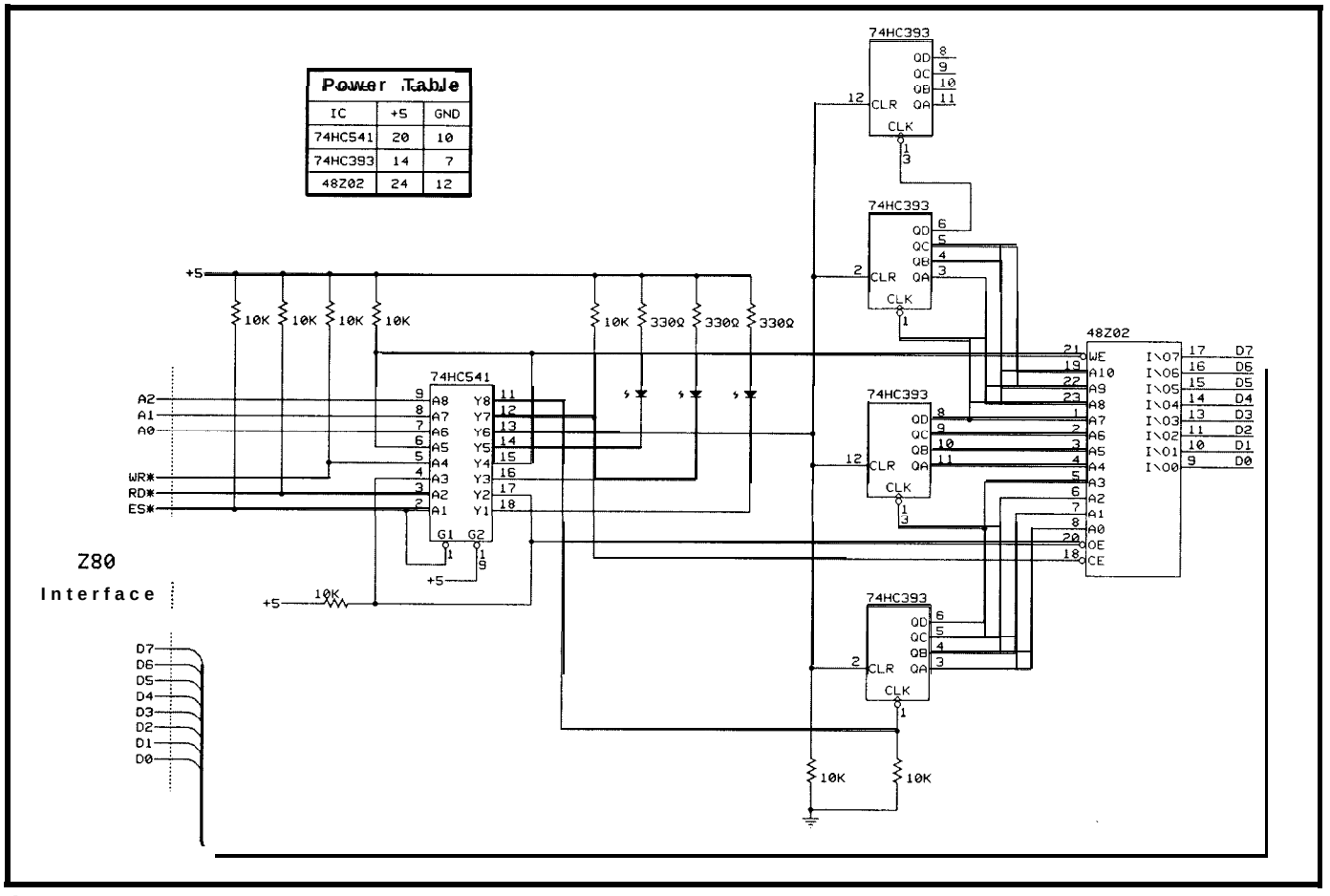
As programmable device types have flourished, engineers have been faced with a dilemma: The new devices offer new capabilities and (often) important advantages, but they seldom work with existing device programmers. This article presents a simple three-chip circuit that reads and,most importantly, writes the SGS-Thomson Zeropower static RAMs, as these devices are not supported on most EPROM programmers. In the process, we will show some tradeoffs between hardware and software design.

For background, the Zeropower RAM, SGS-Thomson part number MK48Z02, is a 2K x 8 static memory chip with an integral lithium battery. This chip has the characteristics of a CMOS static RAM with thebonusthat it retains its data when not powered. It amounts to a fast, infinitely reprogrammable 2716 EPROM.

In this application, the host CPU is a dedicated Z80 system that brings the RD*, WR*, data lines, and the lower address lines out to the external interface. In addition, an external device select line, ES*, decodes I/O ports CO-CF.

A BASIC DESIGN

The basic design increments an ll-bit counter through all states to



tlgure 1-A simple three-chip circuit, with an interface to a basic Z80 controller, is all the hardware required to program the Zeropower SRAM.

```
;
;  Zeropower RAM programmer support code
;
;       Written for the Z80 processor
;
MEMSIZ      EQU       2048              ;48Z02 N BYTES
DELAY       EQU       08                ;393 RIPPLE
RESET       EQU       0C3H              ;CLEAR THE COUNTER PORT
MEMORY      EQU       0C0H              ;SELECT THE RAM PORT
STEP        EQU       0C6H              ;INCREMENT THE COUNTER
PORT
;
;
;           ZMEMWR-WRITE ZEROPOWER RAM
;
ZMEMWR:
;
;           LOAD THE DATA ARRAY
;
            OUT       (RESET),A         ;RESET THE COUNTER
            CALL      RIPPLE            ;DELAY AWHILE
            LD        BC,MEMSIZE
            LD        HL,DATA
ZMEM2:
            LD        A,(HL)            ;FROM DATA(I)
            OUT       (MEMORY),A        ;WRITE A BYTE
            INC       HL                ;I t 1
            OUT       (STEP),A          ;COUNTER t 1
            CALL      RIPPLE            ;DELAY AWHILE
            DEC       BC                ;LOOP COUNTER - 1
            LD        A,B
            OR        C
            JR        NZ,ZMEM2          ;FOR MEMSIZ
            etc.
;
;
;           ZMEMRD-READ ZEROPOWER RAM
;
ZMEMRD:
;
;           ZERO THE DATA ARRAY
;
            OUT       (RESET),A         ;RESET THE COUNTER
            CALL      RIPPLE            ;DELAY AWHILE
            LD        BC,MEMSIZE
            LD        HL,DATA
ZMEM4:
            IN        A,(MEMORY)        ;GET A BYTE
            LD        (HL),A            ;TO  DATA(I)
            INC       HL                ;I + 1
            OUT       (STEP),A          ;COUNTER + 1
            CALL      RIPPLE            ;DELAY AWHILE
            DEC       BC                ;LOOP COUNTER - 1
            LD        A,B
            OR        C
            JR        NZ,ZMEM4          ;FOR MEMSIZE
            etc.
;
RIPPLE:
            LD        B,DELAY
RIPP02
            NOP                         ;DELAY FOR '393 RIPPLE
            DJNZ      RIPP02
            RET
;
DATA        DS        MEMSIZE
;
```

Listing 1 — *The Z80 assembler code for writing and reading the Zeropower SRAM is simple and straightforward.*

derivethememoryaddressesandread or write thememory at each step. Next, we observe that it is easier to issue an **OUT** instruction to a separate port than to decode data bits for external functions. So we assign the address lines as follows:

A0—Reset the address counter
Al-Select the RAM
A2—Increment the counter

We need an active-low RAM chip select; instead of an inverter in the circuit, we invert the address line in the program. This gives us the following port assignments:

Port C_3—Reset the address counter
Port C_0—Select the RAM
Port C_6—Increment the counter

Now we have the following implementation: The needed Z80 control bus lines are buffered with a '541. The '541 Output Enable is driven by the External Select line to recognize the selection. The buffered A0 and A2 lines go to the reset and clock inputs in a double '393 counter. Since we have carefully designed the memory select line active low, buffered Al goes to the RAM chip select.

With minor housekeeping details like pull-up and pull-down resistors (yes, use an HCT541 next time) and pretty LEDs on ES*, RD*, WR* courtesy of the unused buffers, we have the circuit shown in Figure 1 to read and write the Zeropower RAM.

At this point, a hardware reality enters the picture: the carry does not propagateinstantaneously through all stages of a ripple counter. A delay after all counter functions is in order, which we put into the software. Adding it all together, we have the Z80 code in Listing 1 to drive the Zeropower RAM.

Ernest Stiltner specializes in assembly language software for embedded systems.

IRS

431 Very Useful
432 Moderately Useful
433 Not Useful

CONNEC-TIME

Conducted by
Ken Davidson

Excerpts from the Circuit Cellar BBS

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

We've had a ve y active discussion going on lately about computer-controlled theatrical lighting. Since we have a companion article on the same subject in this issue (page 79), I thought it appropriate to share what's been going on on the BBS. There is more on the BBS than would fit here, so feel free to call and add your two cents' worth.

Msg#:36721
From: DAVID MEED To: ALL USERS

I was just wondering if anyone knows anything about theatrical lighting controllers. The ones that I have looked at commercially are quite expensive. Does anybody know what is in them?

I figured that an 8051 could monitor zero crossings on the AC power line and turn on a triac at the appropriate time to dim the light. It would receive info by serial line about how bright the light should be.

What kind of rating do you need for a triac in this instance? The lights are quartz halogen rated at 500 watts. Would a 6-amp triac do the trick or do you need to allow for a 3-5 times current surge when turning the lamp on from cold?

Msg#:36729
From: JEFF BACHIOCHI To: DAVID MEED

Zero-crossing is a good idea for turning things on as long as you want it on for a full half cycle (the next zero crossing is the only way a triac will shut off).

The standard Opto (OACx) modules are zero-crossmg switches. You can simulate dimming by the ratio of on/off cycles, but the flicker makes it ugly!

Fortunately Gordos makes a random turn-on module (suffix -RN) which allows you to turn the triac on anywhere in the cycle, then it goes off at the zero-crossing. Yes, you can have large in-rush currents, but that's the tradeoff!

Msg#:36733
From: ALAN SANDERS To: DAVID MEED

You're on the right track. I've been doing it for years. 1 can get back to you on the details but of the top of my head...

1) Correct about catching the zero crossing; I'm sure the 8031 can do it in time, but you have to get it going both ways.

2) The switching end of my dimmers are modeled after those of the big guys. For a 2.2-kW dimmer channel, I use a 40-A triac. It's a little conservative, diacs or SCRs would be better. They handle line transients more gracefully. Must have filtering on each load.

3) A 6-A triac could do 500 W, but remember that it's vulnerable to line transients (it will latch), and (got it right again) in-rush current if your filament is cold will blow it.

4) That's why we use "lamp warming." Always flow a little current. Nothing visible, lust enough to keep things warm.

5) The eye is not a linear device! There are special gamma tables of one form or another in most professional lighting systems. This provides an apparent linear change in brightness for a linear change in control voltage.

Msg#:36756
From: DAVID MEED To: ALAN SANDERS

Looks like you know all about these animals. Are there commercial units that you are aware of that don't cost an arm and a leg? Is it worth "rollingyour own" using an 8051 and a triac with some kind of RF inductor filter? Or would it be cheaper in the long run to get commercial units?

It looks too simple, so I expect that there are all kinds of things in the basic design that start to get complicated after you make one.

I am curious why you would mention SCRs. Don't they only work one way? Or do you use a pair of them?

Msg#:36865
From: ALAN SANDERS To: DAVID MEED

Professional gear is all pretty expensive. You'll still spend money onyourown,butdependingonyourneedsyouwillsave.Itreally is pretty simple. Generate an mterupt on each zero crossing. Start

a delay loop for each channel. When your timers expire, fire your triacs. The longer the delay, the dimmer the lamp.

As a side line I run sound for a local rock and roll band. The lighting system was put together by myself and an old partner. We have a big box with 32 channels at 2.2 kW each. This resides next to the stage along with a fire extinguisher (have not needed it yet!) with cables running out to each fixture. It's a lot of power so you have to be careful (kids don't try this at home!).

It's intesting you mention using a micro. The design described above is approximately seven years old and I have been meaning to convert it to digital. If I ever get enough time together I will.

In another message I saw a reference to having the dimmer at the lamp. As other messages suggest, this may not be a good idea.

Msg#:36778
From: GREG MILLER To: DAVID MEED

Actually, a lot of the questions have already been answered. Especially note comments about cold-filament in-rush current, and gamma curve relating to eye sensitivity. No, you don't need a zero-crossing device (in fact, you don't want one; you need to turn on at a varying point in the cycle, to change the brightness of the lamp). You use two SCRs back-to-back, and get much better results than with one triac.

However, in your question, you said "theatrical" lighting controllers. I've done theatrical lighting on and off (no pun intended) since 1974, and I can tell you that the last thing I'd want is a dimmer module at each lamp hanging 25 feet in the air over the stage (or out on the balcony rail in the house) when the dimmer fails in the middle of the show. The dimmers should be backstage at the lighting board, so if something dies I can try to patch around it without stopping the show in the middle of the act. So if you're really thinking about theatrical applications, I'd have my doubts about that approach. (Also, you'd then have to run a huge feeder cable to each light pipe, to supply enough current for all the lighting instruments on that pipe, and tap into that cable every 18" or so to connect another dimmer module; it seems more practical to do it the way it's done now.)

Msg#:37037
From: DAVID MEED To: GREG MILLER

My particular application is in our church for the Christmas play, Easter cantata, and plays that the day school puts on. What we have now is a dozen light dimmers and 150-W floodlights from the local hardware store. It does work, but isn't real pretty, and not terribly bright. I am looking into the feasability of getting some real lights and that is why the "theatrical" title. This was supposed to be for Christmas. It now looks like it will be a while...

My initial idea was to have a panel of dimmers backstage, but it appears more costly, although being able to patch around a bad dimmer makes a lot of sense.

It seems that we should have lights in front of the stage, over the front of the stage and at the back of the stage, and possibly some footlights. Running a separate circuit from backstage to each

possible light position gets into 30 or 40 separate circuits. If I were able to fly the dimmer packs I could run four lights from a duplex (split phase) receptacle—only 8-10 circuits. About one-third the cost in wire, not to mention plugs, receptacles, and so on.

The other consideration is dimmer noise in the sound system (I workin the sound room too). It seems to me that a short cord from the dimmer to the lamp will radiate far less noise than a circuit from backstage (6' vs. 40').

A few pertinent questions:

1) Just how big is this in-rush current? Everybody mentionsit, but I haven't seen any figures yet. A 100-W house light bulb I have here measures 9.2 ohms DC resistance. According to my calculator that indicates 13 amps at startup versus 0.83 amps running. That is on the order of 15 times in-rush current! A 500-W bulb could draw 65 A! What am I missing? AC inductance? Or is it the time element- few microseconds at 65 A won't blow the triac?

2) Are the gamma tables the same across the board, or do they vary with the lamp.

3) What resolution do I need? 100 points, 256 points, 1000 points?

Msg#:37070
From: GREG MILLER To: DAVID MEED

First, yes, a typical small stage such as you're considering should have at least two pipes over the stage for hanging lights, probably one pipe in the house (over the audience) in front of the stage (to light the apron of the stage, in front of the curtain line); footlights are rarely used any more.

You don't need a separate dimmer for each light; one big dimmer can power several instruments. You use a thing called a"two-fer" which is a wire octopus with one male plug and two (or more) female sockets; the single dimmed circuit runs up to the light pipe, the two-fer plugs into the dimmed circuit, and the lights plug into the two-fer. This is fairly standard theatricdal lighting practice. Some theatres might have lots of permanent circuits cabled to each light pipe (including front of house), but some just use lots of long cables, typically SJ or SJO cord, which they hang in place and wireas needed by each production. Basically a heavy version of the K-Mart Special orange extension cord. If you make these from 14/2 or 12/2, they can carry enough current for several lamps, minimizing the amount of wiring needed.

Dimmer noise: good consideration. But if you have 100' of wire from the power panel to the dimmer, and three feet from the dimmer to the light, whenever that dimmer turns on, you'll have a current pulse through the entire 103' of wire, and it will all radiate; doesn't matter where the dimmer is located (unless you use shielded wire, which you won't).

One solution to the RF noise and in-rush current problem is to use large chokes at the output side of each solid-state dimmer. This will extend the rise time of the pulse when you turn on the circuit mid-cycle, thus reducing the in-rush current somewhat, and also reducing the radiated noise.

Another technique is to always keep the dimmer turned on slightly, so that the filaments are always glowing dimly; this keeps their resistance high, so that when you turn them up bright, the in-rush current is less.

Gamma table is the same no matter what lamp you use (at least in practice, if not in theory). It's analogous to using log taper pots in an audio circuit, rather than linear taper, so that pot rotation seems to be proportional to perceived loudness. In fact, if you want to control your dimmers manually, you could probably find an appropriate taper control that would solve the problem.

I'm sure 256 steps would be adequate, and you could control each dimmer with one 8-bit number. I started to breadboard a similar system (only for control of slide projector lamps) years ago, as follows: built a 555 timer, locked it to power line crossing, used it to drive a linear ramp generator (from a current source circuit with an LM3900, I think). Then the ramp went to one input of a comparator, and the 8-bit word to the other input (after going through DAC). The comparator output turned on the SCR dimmer. It was actually not a digital dimmer, but a hybrid, and was easy to build with the technology at that time. You could do one now without all the analog mess, given available (i.e., affordable) technology today.

Keep in mind that your ramp (whether digital or analog) does need to be in synch with the same phase of the power line that powers that dimmer, otherwise, you'll have all kinds of strange strobing effects.

Good luck...hope you make it by Easter. By the way, there may be a lot of used lighting equipment on the market, if your ultimate interest is in a lighting system, rather than in a project, per se.

Msg#:37695
From: GREG MILLER To: DAVID MEED

By the way, I was just looking over the specs for a commercially made theatrical dimmer rack. Here's what they say about their SCRs: These dimmers are rated at 3.0 kW, so that's a nominal current of 25 amps. "individual silicon-controlled rectifiers carrying load current shall...have peak nonrepetitive surge current...of 350 amps...." This dimmer is for use at 120 VAC.

The specs also mention the inductive filter to "limit objectionable harmonics; limit radiated radio frequencies; modify the steep wave front...to reduce acoustical noise in lamp filaments." (Yes, I forgot to warn you about the acoustically audible "buzzing" noise the filaments make when you feed them with a steep wave front.) So you see the folks who have been making these things for a while have encountered the same problems you're facing...only they've already solved them, while you're just beginning to think about them!

Msg#:37895
From: DAVID MEED To: GREG MILLER

That peak nonrepetitive surge current is not the rating of the SCR unless I am mistaken. The only question is, what is the nominal current rating of that SCR?

And yes, I have heard the lamps buzzing. I have an X-10 wall light switch on one of my lights and it buzzes constantly! I had forgotten about that problem in this discussion. Is that problem taken care of by the RF choke or is there more involved?

Msg#:38003
From: GREG MILLER To: DAVID MEED

The buzzing is largely taken care of by the choke. Calling it an "RF" choke may be a bit of a misnomer, because its action probably extends down to a rather lower frequency. Yes, as the literature that I quoted said, thechokemodifies the steep wavefront into the lamp, which reduces the buzzing of the filament. It will also reduce the radiated electrical **noise, which** would otherwise extend up into the RF region. I'm not really a theoretician, and don't want to pose as such; I'm someone who has been using and installing, and at times specifying, this type of equipment. I want to be careful about what I say, so that nobody has to correct any of my errors; however, I'll try to be as accurate and as helpful as I can.

Msg#:38247
From: GREG MILLER To: DALE NASSAR

Back in 1975, I was doing lighting on a show in New York City. At that time, the theatre's backstage power was 115 volts _DC_ from Con Edison. The dimmer boards used large variable resistors that were connected in series with the lighting instruments. Unfortunately, the dimmer boards also had a few standard duplex receptacles, intended for clip lights and such. I got to see one "hot shot" sound man, who didn't believe me when I told him **the** receptacle was DC, **have** a strange experience: He plugged his Weller 8200 soldering gun into this DC source. When he pulled the trigger, the primary winding in the gun drew a very large amount of direct current, causing the body of the gun, rather than the tip, to heat up. When he released the trigger (in the act of dropping thegun!), he also discovered that DC will sustain an arc over a much longer gap than the equivalent AC voltage; so the trigger switch just arced over and kept conducting. It took quite a while for the main fuse on the dimmer board to open; by then, hissolderinggun looked like a prototype for The Blob! (And we call them "the good old days"!)

Msg#:38250
From: BOB BARBAGALLO To: ALL USERS

With all the discussions on building dimmers, I would like to present some research that I have on the reaction of lamps that are being dimmed. Here goes:

THEORETICAL SQUARE LAW CURVE

CONTROL POSITION	VOLTS (RMS)	%LIGHT OUTPUT	%LIFE
OFF	0	0	
0	o-12	0	
1	36	1	

2	52	4	
3	63	9	
4	72	16	
5	81	25	
6	89.5	36	
7	97	49	1000
8	105	64	580
9	114	81	200
10	120	100	100

The lighting industry has generally agreed that the relationship between the controller setting and the amount of light produced by an incandescent filament lamp should follow a square law curve. This means the square of the controller setting from 1 to 10 should equal the percent of light output. When this curve is maintained, the light output will appear to the eye to be linear.

Thus, position 5 on the controller dial will appear to give 50% light output even though a reading with a footcandel meter will indicate 25% (5 squared of the light available).

Lamp life data based on G.E. lamp division characteristics calculator. Well, I hope this gives you some idea on how a lamp reacts to a dimmer. Please let me know if this info was useful and if a description of a phasecontrolled dimmer would be interesting.

The Circuit Cellar BBS runs on a IO-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* for8 *data bits, 1 stop bit, and either 300, 1200, or* 2400 bps.

IRS

434 Very Useful
435 Moderately Useful
436 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 (April/May 1991) includes all public messages posted during January and February, 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.

STEVE'S OWN INK

Steve Ciarcia

Why Portable?

I have a problem. I'm supposed to write a solid editorial about portable applications and, the fact is, I don't use them that often. I've seen all the 'latest and greatest" laptop computers with the sexy black cases and big-bucks price tags, but I just can't stop looking at computers as tools, not toys. Maybe I should explain.. .

In the years I've been working in the small-computer business, I've noticed that most people can be put into one of two camps. The first is the "computersare my reason for living" camp. It's pretty easy to spot someone in this group. They were the first (and probably only) one on their block to have a talking wristwatch. Their toilet flushes on X-10 commands. They think it would be "a neat hack" for someone to surgically embed a microprocessor in the back of their head. You get the picture. These people are vital to our industry. They're often the only ones with the patience and drive to sit down and really understand the internals of a new processor or operating system. They will, to their last breath, work to wring the last few extraneous cycles out of a program control loop. They use computers incessantly, but computers are far more than mere tools to them. Computers are their friend, their vocation, their avocation, and (frequently) their livelihood. The result of all this is that they will search for ways to have computers around them, even when there is no objective requirement for the computer's presence. Having a computer around makes them feel good, even if it's just sitting there running a screen saver.

The second camp is the "computers are just tools" camp. While these people may admire a computer for the way it does a job, they see no more entertainment value in a computer than in a wooden potato masher. They will work with a computer to write a report or control a process, but taking a computer on vacation with them, or spending time with a computer "just for the heck of it" strikes them as just a bit odd. The people in this camp are also vital to our industry because they take the passions of the first camp and figure out reasons for large corporations to spend lots of money on them. Believe me when I say that IBM is much more interested in sending its sales team to the second camp than to the first. Of course, if they're trying to hire people to work in their labs, the first camp begins to look like an awfully attractive destination.

It may surprise you to hear that I place myself squarely in the "computers are just tools" category. I spend a lot of hours in front of one as I run a business, but I spend just as many hours with other business tools like telephones and calculators. I appreciate the work that computers let me do, and I even enjoy working with them (and designing them) but I don't see computers as recreation-I've never, to this day, played a computer game! All of this is a fairly lengthy prelude to saying that I haven't yet seen a portable computer that contains an application so compelling I feel I need to carry it around with me.

Part of the problem I have with portable computers is my set of work habits. Writing and electronic designing are jobs I do in specific places where I have the tools and ambiance necessary for work. Maybe I'm strange, but I don't remember the last time I felt an uncontrollable urge to do a board layout while sitting in a restaurant. If I'm traveling, I want to travel, not catch up on my correspondence. When I go on vacation, relaxing is a top priority. Furthermore, I don't need a computer to help me organize my life. I know I'm unusual in this regard, but I have a staff that helps me keep all my appointments, contacts, projects, and priorities in order. There's no way I'm going to trade the system I have now for any computer, no matter how capable.

I understand that there are people who use portable computers on a regular basis. Curtis Franklin has been preaching the gospel of portable computing for quite a while. He says that he uses his for writing, editing, and project management. According to him, it lets him do the work when he feels productive, not just when he's at a desk. In spite of his attraction to the technology, I notice that he hasn't given up his DayTimer, or his legal pad, or his fountain pen, or.. . The point is that he's treating the computer as a tool, and one that is, in the final analysis, only so useful.

I'm sure the day will come when computers become so small and so capable that I will feel compelled to have one with me at all times. Until then, I'm going to continue to view the computer as just another tool, and continue to use the tools that suit the job best.