

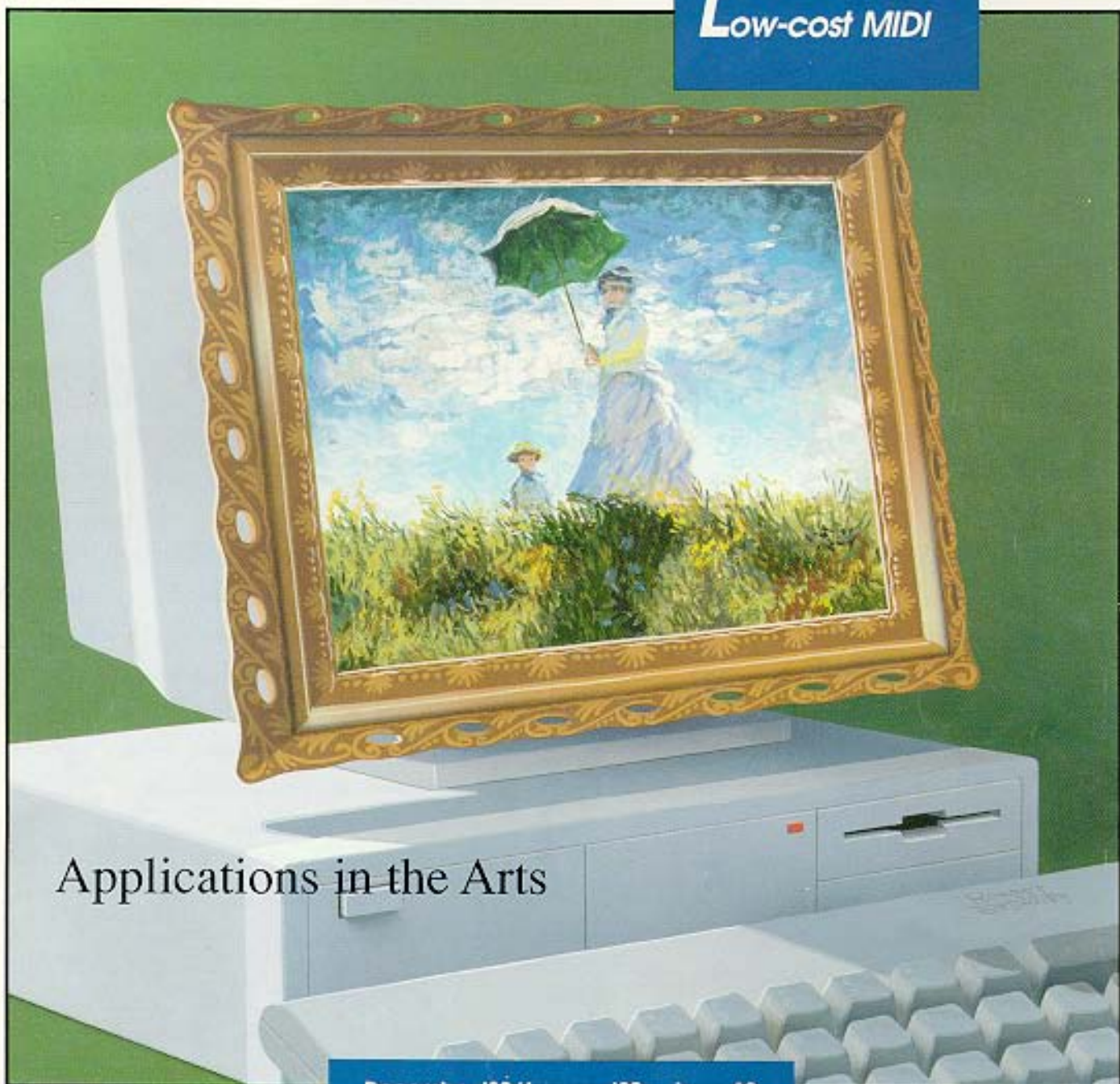
CIRCUIT CELLAR **I N K**

THE COMPUTER  
APPLICATIONS  
JOURNAL

*Professional  
graphics  
techniques*

*Build an  
80386 SX  
Computer*

*Low-cost MIDI*



Applications in the Arts

December '89/January '90 — Issue 12

\$3.95



0 74470 75349 0

## First Things First

**F**irst things first: The cover is different. Part of the reason is aesthetic—after a couple of years of looking at the same logo we felt that it was time for a fresh look for the cover. More than the matter of looks, however, was trying to let folks know exactly what CIRCUIT CELLAR INK is all about. We wanted to make it plain to even the most casual observer that this is The Computer Applications Journal. I think the new logo goes a long way toward making that clear.

Now, I want to make something clear: we haven't changed the parts of the **magazine** that make it uniquely CIRCUIT CELLAR INK. You'll still find projects, tutorials, technology, and techniques in our pages. We still stress practical, hands-on experience. We are still making this magazine for you, our readers, and for no one else. (OK, maybe for us, too, but we read the thing as well as write it.)

### IT'S BEEN ONE OF THOSE YEARS...

In the first issue of CIRCUIT CELLAR INK there was a project on building a home satellite weather center. It continued through our first seven issues, and was supposed to go for several more. As with many projects in this world, "circumstances beyond our control" intruded. We finally decided that there would be a single wrap-up article in this issue. More circumstances, and even that didn't work out. I regret to say that there will not be a concluding article. The author, Mark Voorhees, would like to continue supporting those readers who are now caught in the middle of this project. If you need more information write to him directly at:

Mark Voorhees  
P.O. Box 27476  
Phoenix, AZ 85061-7476

### ON WITH THE ART

It's hard to get any two artists to agree on anything. That said, I choose to believe that many artists would agree that computers have the potential to greatly influ-

ence all of our classic art forms (graphic arts, music, theater, and dance). I've been watching the confluence of computer and art for a number of years, and I'm ever more impressed at the results from both the artistic and computer sides of the issue.

On the artistic side, a welcome maturity is developing in the way many artists deal with the computer. Remember the early days of microcomputers? You would get the computer kit, build the machine, and bore your family and friends by making them watch the front-panel LEDs blink as the computer added a series of numbers. Eventually, the computer had to do more than simply be a computer: it had to do something useful to earn its keep. A similar change in attitudes is moving computer art. Attitudes about computers have progressed to the point where computers are more important as tools than as symbols.

The developments on the computer side are, if anything, more impressive. Where "computer art" was once accorded all the serious respect owed a dot-matrix rendition of Snoopy, the techniques explored by graphic artists are now used by imaging experts in many scientific and technical fields. Computer music which once resembled nothing so much as spinning a radio tuning dial has now broadened to include support for more traditional forms of expression (as well as experimental pieces). Through it all, art has become just another field in which computers are used, and the new uses of computers explored.



Curtis Franklin, Jr.

FOUNDER/  
EDITORIAL DIRECTOR  
Steve Ciarcia

PUBLISHER  
Daniel Rodrigues

EDITOR-in-CHIEF  
Curtis Franklin, Jr.

ASSOCIATE  
PUBLISHER  
John Hayes

ENGINEERING STAFF  
Ken Davidson  
Jeff Bachiochi  
Edward Nisley

CONTRIBUTING  
EDITORS  
Thomas Cantrell  
Jack Ganssle

CONSULTING  
EDITORS  
Mark Dahmke  
Larry Loeb

CIRCULATION  
COORDINATOR  
Rose Mansella

CIRCULATION  
CONSULTANT  
Gregory Spitzfaden

ART & PRODUCTION  
DIRECTOR  
Tricia Dzedzinski

PRODUCTION  
ARTIST/ILLUSTRATOR  
Lisa Ferry

BUSINESS  
MANAGER  
Jeannette Walters

#### STAFF RESEARCHERS

Northeast  
Eric Albert  
William Cure w  
Richard Sawyer  
Robert Stek

Midwest  
Jon Elson  
Tim McDonough

West Coast  
Frank Kuechmann  
Mark Voorhees

Cover Illustration  
by Robert Tinney

CIRCUIT CELLAR **I N K**

# THE COMPUTER APPLICATIONS JOURNAL

In This  
Issue...

22

## Image Synthesis: A Tutorial Tools for Drawing a New Universe

by Chris Ciarcia

The day has long since passed when "computer art" meant outlining a famous beagle on a dot-matrix printer. Today's computer artists can accurately render a standard living room, or an unseen world. This high-level tutorial introduces ray-tracing, shadowing, textures, and reflections, as well as other concepts important for creating images from digital data.



## DEPARTMENTS

### Editor's INK

First Things First \_\_\_\_\_ 1  
by Curtis Franklin, Jr.

Reader's INK—Letters to the Editor \_\_\_\_\_ 5

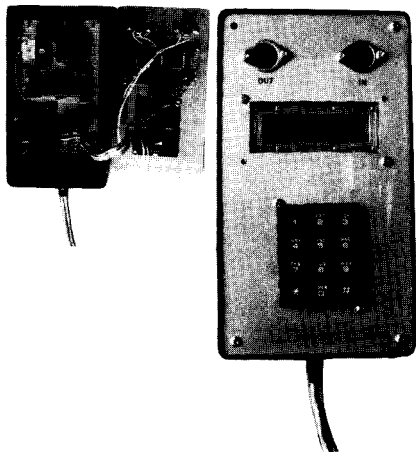
NEW Product News \_\_\_\_\_ 8

Visible INK- Letters to the INK Research Staff \_\_\_\_\_ 12

Firmware Furnace  
Simulated Reality \_\_\_\_\_ 53  
Simulating Systems for 8051 Debugging  
by Ed Nisley

From the Bench  
Gentlemen, Start Your Engines \_\_\_\_\_ 62  
by Jeff Bachiochi

Advertiser's Index \_\_\_\_\_ 73



**49** A Low-Cost MIDI Sequencer  
 Build an 8031-Based Stand-Alone  
 MIDI Sequencer  
 by Winefred Washington

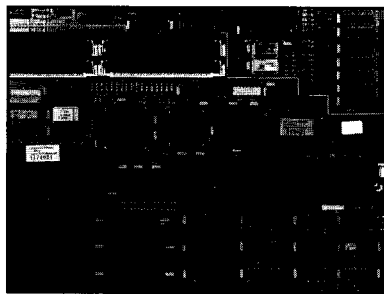
MIDI opened up a new world of electronic music. Sequencers open up new vistas of control within that world. This project from the Circuit Cellar INK Design Contest lets you build a simple, functional sequencer for less than \$75!

**14** INKnet  
 Part 3—Writing Network Applications  
*The Subtle Art of Programming for INKnet*  
 by Ed Nisley

A computer network is of little use if there's no application software to take advantage of the links. Writing software for INKnet is one way to learn the ins and outs of programming for distributed control.

**37** An Intel 386SX-based PC/AT-Compatible Motherboard-Part 2  
 by Daryl Rinaldi

In every case, a working computer is more than just the sum of its parts. Timing's importance cannot be overestimated when it comes to understanding why a particular design works as it does.



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

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

CIRCUIT CELLAR INK (ISSN 0896-8985) is published bimonthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 16066 (203) 875-2751, second-class postage paid at Vernon, CT and additional offices. One-year (6 issues) subscription rate U.S.A. and possessions \$14.95, Canada \$17.95, all other countries \$26.95. 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 2099, Mahopac, NY 10541 or call (203) 875-2199.

POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 2099, Mahopac, NY 10541.

Entire contents copyright 1989 by Circuit Cellar Incorporated. All rights reserved. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

Silicon Update		
MAXIMUM Chips	_____	69
<i>Maximum Real-World Problem Solvers</i>		
<i>by Tom Cantrell</i>		
Software by Design		
Least-Squares Curve Fitting	_____	74
<i>by Jack Ganssle</i>		
ConnectTime—Excerpts from the Circuit Cellar BBS		82
<i>Conducted by Ken Davidson</i>		
Domestic Automation		
New X-10—Compatible Products Hit the Market	_____	78
<i>by Ken Davidson</i>		
Steve's Own INK		
...And Everything in its Place	_____	88
<i>by Steve Ciarcia</i>		

# letters to the Editor

## A BETTER WAY TO BOOT

Recently I received *CIRCUIT CELLAR INK's* First Year Reprint. It contains many invaluable ideas which inspire me in my own designs. Your publication is almost error free but I found something that in my opinion should be corrected.

In *CIRCUIT CELLAR INK #6*, in *Visible INK*, you described the program WARMBOOT.COM. I have four comments concerning this program:

- 1) I cannot understand why you use the ES segment register to modify BIOS variables—I think using DS is more straightforward.
- 2) You don't have to end the program with RET—it will never be executed.
- 3) You don't have to write 112h bytes to disk—12h is enough.
- 4) When you enter the "u 100 112<enter>" command, you can also see an instruction which begins at xxxx:0112 (in this case, its content is undetermined).

Here is my version of WARMBOOT.COM. It is designed to be entered using DEBUG.

```
C>debug warmboot.com
File not found
-a100
XXXX:0100 mov ax,40
XXXX:0103 mov ds,ax
XXXX:0105 mov word ptr [72],1234
XXXX:010B jmp ffff:0000
XXXX:0110 <enter>
-r cx
cx 0000
:10
-w
Writing 0010 bytes
```

Isn't it better?

Some PC/XT-compatible computers ignore the contents of the 0040:0072 BIOS variable. In the keyboard

interrupt routine, they have a direct jump to the boot routine which bypasses the POST. A reset routine which begins at FFFF:0000 simply overwrites the contents of the 0040:0072 address, so for these computers, WARMBOOT need only contain one instruction:

```
jmp ffff:0000
```

or a jump to a specific BIOS location. In other words, for an ill-behaved BIOS, use an ill-behaved WARMBOOT.

Krzysztof Wysocki  
Warsaw, Poland

*Thanks for sending us your solution to the problem of warm boots under MS-DOS. One of the best parts of putting together a magazine like CIRCUIT CELLAR INK is the chance to learn from your readers. Editor*

## WIRE-WRAP, CONTINUED

In *CIRCUIT CELLAR INK #9*, there was a group of messages concerning wire-wrap construction in *ConnecTime*. I have an additional suggestion for building projects using wire-wrap.

Use a board that provides *Vcc* and *GND* foil buses on 0.3" centers between IC socket pin rows. This allows *Vcc* and *GND* wires to be wrapped to pins and lap-soldered to the buses, minimizing the ground jump typical of high-performance CMOS chips.

The dead-bug method is not dead. In my technique, the circuit is built on a copper-clad sheet. Chips are superglued, pins up, to the main foil (*GND*). *Vcc* islands are cut in the foil to match up with the *Vcc* pins. *Vcc* and *GND* pins are bent over to match up with the foil and islands, and soldered into place. Bypass capacitors are installed as required. The simplest \$2.95 wire-wrap tool is used to interconnect IC pins with #30 wire-wrap wire. I find that

only two or three wraps are necessary. Two-level wraps are easy, and connections may be soldered for permanence or ruggedness. Of course, the board produced won't fit into a single card-cage or backplane slot, but I have found the results to be durable. **The method** is especially good for small one-time projects, or for early prototyping.

Charles Shaw  
Ft. Walton Beach, FL

## COMPUTERS ARE ENOUGH

CIRCUIT CELLAR INK has improved as a magazine over the last year. The technical quality of the articles is probably better also-but I don't read most of them. I am interested in computers as computers, not as a fancy way to turn on my house lights or close a window. I realize that a lot of people are interested in interfacing computers to the real world, but I'm not one of them. To me, a computer is something to help me think, not to think for me. Actually, I'm **not** much interested in even that: Once a computer is designed and built and an operating system of sorts is in place, I leave the rest to the software folks. It's the machine

itself that interests me, not the uses and abuses to which others will put it. To me, the challenge is in cutting cost and complexity-the black box itself-not what is tied to it.

I continue my subscription to CIRCUIT CELLAR INK, not because I like it, but because it's the only magazine that even comes close to what I am looking for in a computer magazine.

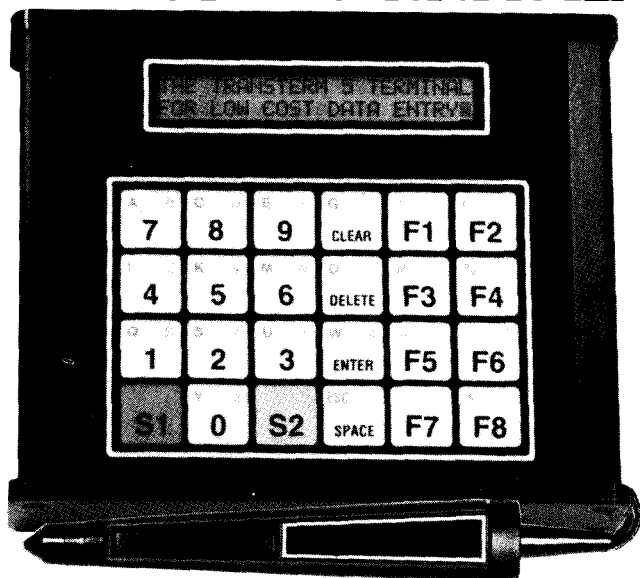
Al Seaward  
Baton Rouge, LA

*Thank you for taking the time to write in with your comments on CIRCUIT CELLAR INK: Thank you even more for the subscription renewal which accompanied your letter!*

*As you know, the full title of our magazine is CIRCUIT CELLAR INK: The Computer Applications Journal. Remaining faithful to the latter half of the title requires that we place real emphasis on computer applications. In spite of this, I believe that those who want to see computers will be happy with us in the coming months. In addition to the H16 and 80386SX computers which began in CIRCUIT CELLAR INK #1 1, we are planning projects based on the 68000, 8096, and 68HC11 processors and controllers in the next few months.*

*We'll never forget that computers are prerequisites for applications. We hope that, in covering both ends of the process, we can please most of our readers most of the time. Editor*

# \$249. TERMINAL



- Featuring
- Standard RS-232 Serial Asynchronous ASCII Communications
  - 48 Character LCD Display (2 Lines of 24 each)
  - 24 Key Membrane Keyboard with embossed graphics.
  - Ten key numeric array plus 8 programmable function keys.
  - Four-wire multidrop protocol mode.
  - Keyboard selectable SET-UP features-baud rates, parity, etc.
  - Size (5.625" W x 6.9" D x 1.75" H), Weight 1.25 lbs.
  - 5 x 7 Dot Matrix font with underline cursor
  - Displays 96 Character ASCII Set (upper and lower case)
  - Options-backlighting for display, W-422 I/O, 20 Ma current loop I/O,

**COMPUTERWISE, INC.**

302 N. Winchester • Olathe, KS 66062 • 913-829-0600 • 800-255-3739

## PC BIOS Development • Industrial Control • Embedded Systems • ROM Development



ROM  
EEPROM  
SRAM  
SRAM AS ROM

### The Kolod Research R<sup>3</sup>OM Card

SUPPORTS MOST 28-PIN JEDEC MEMORY DEVICES. YOU CAN USE STATIC RAMS AS ROMS TO DYNAMICALLY DEVELOP AND TEST PC ROM BASED CODE WITHOUT BURNING IN ROMS AND FULLY DEBUG THE ROMS YOU DO BURN

The R<sup>3</sup>OM card provides all the facilities you need to develop and deliver PC based BIOS, embedded systems, industrial control software, and ROM based applications

#### The R<sup>3</sup>OM card features include:

- \* 4 Independent 28 pin JEDEC defined sockets: each socket's address is switch-selectable to anywhere within the 0 to 1Mbyte address range
- \* ability to address and configure sockets independently or consecutively for a total of upto 126K SRAM, 256K ROM, or any combination of ROM, EEPROM, SRAM in between
- \* use only the sockets and memory you need
- \* jumper select battery backup for any or all of the sockets
- youth of the sockets means
- use slower memories in high-speed systems
- software or hardware controlled write protect for each socket (for SRAMs and EEPROMs)
- unique software driven 3 channel DMA fly-by test circuitry
- flexible port I/O addressing including technical documentation and programming with examples (source code included. of course)
- full support for PC/XT/AT/386 machines
- variable high quality multi-layer construction

Engineering Excellence



**Kolod Research Inc.**

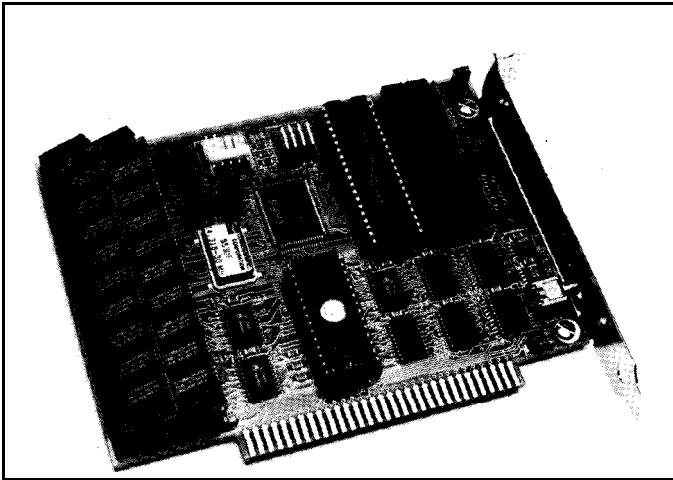
1898 Techmy Court, Northbrook, Illinois 60062-5474 U.S.A.

\$285.00 + shipping/handling  
Made in U.S.A.

( 3 1 2 ) 291-1586 options, technical information

Reader Service #129





## Turbo XT on a 4" by 5" Board

The 8810 board from Tempustech is the equivalent of an IBM-compatible Turbo XT motherboard on a 4.2" by 5.4" board. The board, which fits an industry-standard 8-bit PC bus passive backplane, offers the capability of an embedded computer in a space as small as 180 cubic inches (5"x6"x6"), about 1/10th the size of an IBM PC/XT.

Applications for the U.S.-designed and manufactured board include a diskless workstation, an embedded controller for industrial applications, as well as the CPU board in a PC. The 4-layer board uses surface mount technology, features dual-speed (10 MHz and 4.77 MHz) operation, up to 2 megabytes of SIMM DRAM, NEC V20-10 processor, optional 8087 coprocessor, and an auto-configuration BIOS with no switches to set. The board is compatible with MSDOS 2.0 to 3.x, thousands of industry-standard PC bus accessories, and includes an Expanded Memory Manager (LIM V4.0).

A plug-in CPU offers advantages in serviceability and upgrade capability. A defective unit can be repaired in minutes by replacing only one plug-in board. The passive backplane eliminates the need for a motherboard and provides an upgrade path.

The size of the 8810 gives the designer a great deal of flexibility. Since most 8-bit accessory functions are available on half-size boards, a complete XT clone with 2 megabytes of DRAM, a 3.5-inch floppy disk, a 30-megabyte hard drive, a video card, an I/O card, and a power supply could be assembled in a case about 5"x7"x9".

The 8810 board costs \$160 (quantity 10) without memory. Tempustech also offers 8-bit and 16-bit passive backplanes. IBM PC/AT and "386 board" equivalents are also available.

Tempustech, Inc.  
295 Airport Road  
Naples, FL 33942  
(8 13) 643-2424

## C Users' Group Library Directory

Volume II of the C Users' Group Directory of User-Supported C Source Code has been released by R&D Publications Inc. The Directory contains file-by-file descriptions of the code in the C Users' Group Library, the world's largest repository of public domain and user-supported C source code. Volume II of the directory describes Library disk volumes 200-249, and includes a comprehensive index plus detailed articles describing some of the most significant volumes in the library. The 208-page paperback was edited by Robert Ward and Kenji Hino.

Some of the programs cataloged in Volume II of the Directory include a 68000 C compiler, an inference engine and rule-based compiler, portable utilities, MS-DOS implementations of some popular UNIX utilities, Small C utilities, and several graphics packages.

The C Users' Group is a service of R&D Publications Inc., and facilitates the exchange of code and information among C programmers. Unlike machine-centered users groups, the Group focuses on source code useful to experienced programmers rather than tested, end-user applications. Among other services, the Group maintains a library of over 180 volumes of public domain C source code.

Volume I of the Directory, a separate publication, contains file-by-file descriptions and an index covering disk volumes 100-199, the first 99 Library volumes. Both Volumes I and II are available from The C Users' Group for \$10 each.

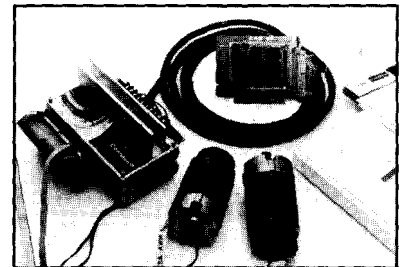
The C Users' Group  
2120 W. 25th St., Ste. B  
Lawrence, KS 66047  
(913) 841-1631

## Servo System for PC Control of Machinery

Owners of hand-operated machinery can now adapt their tools to a computer-based automation system with **Retro-Fit-Kit** from Computer Continuum. A two-axis kit consists of a dual controller, dual power amplifier, two DC servo motors with incremental encoders (2000 counts/rev), an interface card for the IBM PC, CNC/teaching software, and all cables. One cable is specially shielded for use in electrically noisy industrial environments. The controller and amplifier modules are contained in stackable plastic boxes and interface to the PC through a ribbon cable bus called LAB 40 (Local Applications Bus). Up to eight analog, I/O, and relay modules may interface to this bus with up to 40 feet of ribbon cable.

Each power amplifier delivers continuous currents of 10 amps at up to 90 volts, with peak currents of 50 amps for fast motor stops.

Protection is provided for over-current, over-voltage, under-voltage, short circuit, and over-temperature. Optional heat sinks, fans, and power upgrades, using larger or parallel MOSFETs are available. Signals between the controller and



power amplifier are optically isolated.

MotionSoft, the included automation software, supports up to four axes of synchronized motion with linear and circular interpolation on either pair of axes. Parameter set-up and motor-testing menus allow adjustment for a wide variety of motor/encoder/gearboxes for each axis. After limit switch initialization, CNC files may be used from a standard CAD system. Motor positions can be taught with PC cursor keys as jog keys or by turning the motor shaft and recording the positions from its attached encoder.

Velocities between each point are settable, and the motion playback can be either continuous or trapezoidal profile. Trajectories and automation steps are stored in CNC code format. The package includes routines in C and QuickBASIC for custom development.

A two-axis Retro-Fit-Kit starts at \$2000 each. System requirements include an IBM-compatible PC, XT, or AT with

640K of RAM and either Hercules or CGA graphics. An 80x87 math coprocessor chip is required.

**Computer Continuum**  
75 Southgate Avenue  
Daly City, CA 940 15  
(415) 755-1978

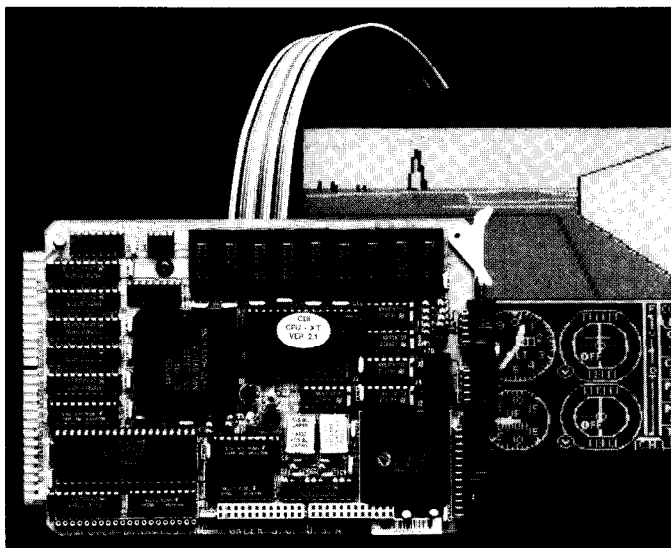
## Computer-Based HDTV Framestore

Real-time storage, retrieval, and manipulation of high-definition images for the Macintosh II computer is possible with Restore, a computer-based high definition dual-frame store from Rebo Research (a subsidiary of Rebo High-Definition Studio Inc.). Using the Mac II as its host/controller, the Restore chassis plugs into the Mac's NuBus slot and allows access to all Macintosh mass storage devices and third-party software. Captured images can be viewed on a high-definition monitor or displayed on the Mac color screen.

Restore stores images in one of two buffers. The two frames can be captured live from the camera source or from high-definition videodisc or tape. Either the live video or the buffer's contents can be viewed. Restore can reposition the image from buffer A or B on the screen. A live image may be repositioned and stored in its altered form. The product features also include 1920x1035 pixels, 24-bit color, and color correction.

Images generated entirely within the computer can be displayed or recorded in the high-definition format and processed through Restore. Any form of communication supported by the Mac may be utilized, including serial, parallel, or medium exchange (optical disc and tape). Image translation software is available through Rebo Research.

**Rebo Research, Inc.**  
530 West 25th Street  
New York, NY 10001  
(212) 627-9083  
(415) 755-1978



## PC-Compatible STD Bus Module

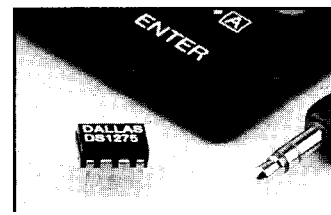
The CPU-XT module from Computer Dynamics is the first STD Bus module to offer 100% PC/XT compatibility. It incorporates an 8088 or NEC V20 microprocessor, memory, and speaker and keyboard interfaces. In addition, the CPU-XT includes: interfaces to floppy and hard disk drives; monochrome/CGA/LCD display adapters; a watchdog timer; battery-backed clock/calendar; 128K of EPROM; up to 640K of RAM; two serial ports; and a printer port. The unit is an STD Bus module, measuring 6.5" x 4.5" x 1", and features low power consumption so it can easily be placed into embedded systems. It is also keyboard switchable to operate at a 9.54-MHz clock speed.

The on-board IDE hard disk controller accommodates up to two Winchester drives and makes possible an XT-compatible hard disk interface. The CPU-XT can be used in disk- or ROM-based environments, and its bidirectional printer port makes the module compatible with security keys. It directly addresses all STD Bus I/O and up to 64K of memory on the STD Bus.

The CPU-XT is \$995 in single quantities.

**Computer Dynamics**  
107 S. Main Street  
Greer, SC 29650  
(803) 877-8700

## Transceiver Links Portable Systems to PCS



Linking hand-held equipment to personal computers without consuming battery energy is possible with the Dallas Semiconductor DS1275 Line-Powered Transceiver. The DS1275, a low-power CMOS device, has a unique circuit that steals current from the host computer's RS-232 signal when that signal is in a negative state. Since most serial communication ports remain in a negative state statically, battery current is reduced by an order of magnitude.

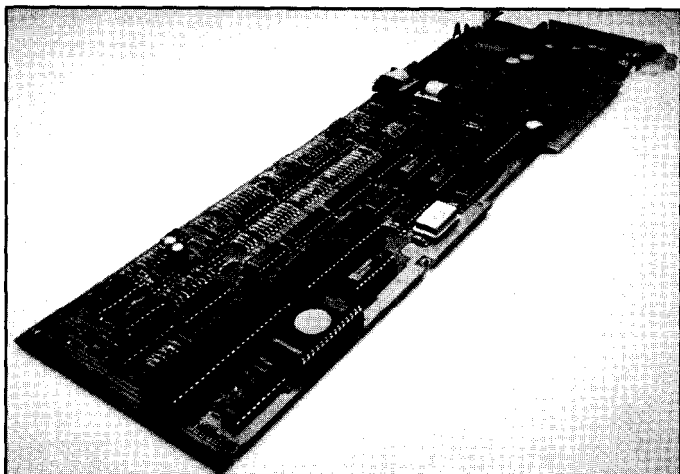
During an actual communications session, the DS1275's transmitter will use system power (5-12 volts) for positive transitions while still employing the receive signal for negative transitions. When the system is in the static state, with no data being passed, the DS1275 draws no battery current at all. This permits the portable device to go into a power sleep mode until activity on the RS-232 port is sensed by the transceiver.

The B-pin DIP or surface-mount package is compatible with RS232 signals and operates over a simple 3-wire cable. Applications include remote sensors, portable instruments, hand-held bar code readers, or any portable device that needs to send information to a PC.

The DS1275 costs \$1.73 in 5000-piece quantities.

**Dallas Semiconductor**  
4350 Beltwood Parkway  
Dallas, TX 75244  
(214) 450-0400





## Multiline Voice Communications Board

Integration of a PC/XT/AT/386 computer with Central Office, Centrex, and/or PBX lines is achieved with the Powerline II board from Talking Technology Inc. Applications include voice messaging, call processing, audiotex, telemarketing, order processing, call distribution, PBX integration, auto dialing, and alarm systems.

Powerline II supports high-fidelity digital recording and playback with or without compression. Five software-controlled compression rates ranging from 14 to 56 kbits/second give flexible control over sound quality and data storage rates.

Two on-board microprocessors control the functions of Powerline II. An Intel 8098 handles data compression and decompression, and an Intel 8032 communicates directly with the 8098 and handles housekeeping functions. The 8098 has a built-in 10-bit A/D converter, sample and hold, and multiplexer. During voice recording, the 8098 samples the filtered audio input, compresses the data through the use of a proprietary algorithm, and sends it to the 8032. During playback, the 8098 accepts encoded data from the 8032 and decodes it, making it available for playback into the telephone line, local telephone, and/or an external speaker.

The Powerline II hardware interfaces with the host computer through dual-ported static RAM, and all communications is channeled through this memory. A variety of jacks on the board provides interface to telephone lines, PBX systems, audio input and output, and digital external devices. Powerline II is priced at \$599.

The Commando Developers' Package provides all of the tools necessary to custom tailor inbound and outbound voice applications supporting as many as 16 lines (eight Powerline II cards). Commando includes a function library which supports concurrent voice messaging and call processing applications. The Commando kit includes a software applications interface for the MS-DOS environment, a graphics-based installation program for Powerline II cards, a microphone for high-quality recording, and a speaker. Commando is priced at a one-time fee of \$139.

**Talking Technology, Inc.**  
4383 Piedmont Avenue  
Oakland, CA 94611  
(415) 652-9600

## ROMable Disk Operating System

A new ROM-resident or disk-based operating system for embedded systems using the Intel 80x86 family and the NEC V20 through V50 series has been announced by Datalight. ROM-DOS is fully compatible with MSDOS 2.x and will boot on a standard PC, operate from within ROM, and run MSDOS executable programs such as Lotus 123, Turbo-C, and Microsoft Word.

The ROM-DOS operating system enables a diskless embedded system to run an MS-DOS application upon power-up. Any MS-DOS service can be accessed, provided the necessary hardware is available. ROM-DOS supports memory management, a standard file system, time functions, and DOS character and block device drivers. A device driver may be installed at link time or load time (using CONFIG.SYS). A device driver installed at link time becomes part of the operating system kernel and is burned into ROM with ROM-DOS.

A standard setup places the code for ROM-DOS and Mini-BIOS in the top 32K of ROM, and the user application (EXE file) and associated files in ROM on a ROM-disk. The ROM-disk files can be placed anywhere above system RAM, and by designating the ROM-disk as the "A" drive, ROM-DOS will search it for the initializing program upon booting.

The booting process starts when power is applied to the system. The BIOS initializes the hardware and transfers control to ROM-DOS which, after its own initialization, loads and runs the user application. This application remains operational until power down.

The ROM-DOS kernel uses 29K bytes and resides in and executes out of ROM. The Mini-BIOS uses less than 3K bytes, so it will easily fit into a 32K ROM package. A minimum RAM space of 5K bytes is needed, but a command processor (COMMAND.COM) is not required.

A Developer's Kit provides the tools, modules, source code, and instructions to port ROM-DOS to a new environment. The Mini-BIOS in assembly source form contains all of the hardware dependent code and provides the minimum hardware support required by ROM-DOS.

The ROM-DOS Developer's Kit is available for \$495 and comes with 20 duplication licenses. The license cost is the royalty paid to Datalight for the right to duplicate the software and varies from \$25.00 per license for 50 units to \$3.00 per license for 25,000 units. The license to the full source code for ROM-DOS is available for \$5000.

**Datalight**  
17505 68th Avenue  
Northeast, Suite 304  
Bothell, WA 98011  
(206) 486-8086  
(800) 22 1-6630

## Attention Manufacturers!

Circuit Cellar INK provides its readers with news about significant product developments in hardware, software, and development tools. If your company has a new product that our readers should know about, please send your product announcements to:

**Circuit Cellar INK**  
New Products Editor  
4 Park Street, Suite 20  
Vernon, CT 06066

# VISIBLE INK

answers;  
clear and simple

## Letters to the INK Research Staff

---

### OH, FOR A SIMPLE GAME PORT...

I need a multiple-channel temperature logging and data reduction system similar to the one Tom Riley presented in March/April of 1988. [See "Personal Computer-Based Instrumentation" by Tom Riley, *CIRCUIT CELLAR INK* #2.] However, I have a laptop computer with only RS-232-C and parallel ports. Will you modify this circuit's software to allow the parallel port to receive input? The Toshiba 3100 has no game port. It would also be helpful to allow more thermistors, if possible.

I know now that I can't live without a modem to access your BBS. I live aboard a boat and most everything must operate off the 12.6V system batteries. Can you suggest a modem in kit form that is available for my use? If not a kit, any lower-priced unit will be acceptable.

Jack H. Peterson  
Ft. Lauderdale, FL

The project described in the article fakes advantage of the Apple B-series game port and embedded software in the BASIC interpreter. While this could be easily duplicated by an IBM PC or compatible with a game port, your laptop is missing this feature.

There is a method of adapting this circuit to a PC with a parallel port, but if *does get* complicated a requires *some software* tricks. Normally, the data lines on a parallel port only go "one way"—to the printer. There are eight of them, so theoretically you could get eight thermistors. In reality, you have to read the status of the eight data lines. Not all parallel port hardware will let you do this. In addition, the Apple II game port is actually an A/D converter of sorts, so the voltage at the game port connectors is what's actually being measured. You can't do that with a parallel port, since it only receives a logic signal. Therefore, the approach you take must deliver a signal to charge the capacitors and measure the time for each line to change state (because of the RC time constant of the thermistor and the capacitor). You could use the data strobe line of the parallel port to control the charging.

Actually doing this will require assembly language routines to access the parallel port in the manner described. There are lots of books out there on low-level manipulations of the PC hardware, and it's difficult to recommend just one since we don't know your background or temperament.

As far as your modem purchase is concerned, just take a look at all of the 9V-battery-powered "pocket modems" now on the market. Most recent consumer-oriented computer magazines are filled with ads for them. At current prices, it's impossible to justify going to a kit. Another option is to find an external modem designed to be powered by 110 VAC. Most of these have transformers which convert the 110V wall current to 9-15V for the modem itself. Find one with a 12V or lower DC input, fit it with a cord, and away you go.

---

### RESETTING AN AT

Can you help me by supplying a circuit for a reset switch for an AT clone, or telling me how I would go about installing one? I have a switch that produces an NMI interrupt, but it is frequently insufficient. I don't want to turn the power switch off and on too often because it cycles the hard disk as well, but I do need to be able to reset the 80286 (and the bus, if possible).

David S. Bakin  
San Francisco, CA

Adding a reset switch to your AT clone is not very difficult, and there are a couple of ways it might be done. The most obvious way, driving the RESET DRV pin on the expansion bus, is *not* the way to do it. This is an output signal used to reset other things. We only mention this because we have seen others make reference to grounding this pin to reset the system.

The system will reset if the POWER GOOD line from the power supply is pulled down for a short time. This is the normal way that the system begins its boot process on startup. The POWER GOOD line stays low until the power supply output voltages are within spec, then goes high. In a True Blue AT, this line goes directly to the RES\ pin on the 82284 clock generator. When this pin goes high, it generates a reset signal at its reset pin. This is the signal that actually resets the CPU and begins the boot process.

The POWER GOOD line comes out of the power supply and goes to pin 1 of one of the main power supply connectors. The

connector you want is labeled P8 on PC power supplies, and P3 in ATs. In any case, there are two main power supply connectors, one with the key on pin 1, and one with a key on pin 4. The one you want is the connector with the key on pin 4. Ground is pins 5 and 6 on this connector.

To add the switch, all you need is a normally open momentary-contact push-button switch and a few feet of hook-up wire. One side of the switch is to be connected to the POWER GOOD line, and the other side will be connected to ground. It will probably be convenient to make both connections to the leads coming out of the power supply, or to the pins in the connector itself. The easiest way to make these connections is to splice the switch leads onto the power supply leads with crimped-on, solderless, splice connectors, but you could strip in a short section of insulation on each lead and solder the switch leads in, or insert the switch lead wires into the connector from the rear.

---

## WORKING WITH THE FCC

I am working on starting up my own business and plan on designing a computer as one of my products. What will I need to do to get the required FCC certification? According to an article I read, the paperwork, delays, and **money required** for certification are prohibitive for a start-up company. Is this true? If so, something needs to be done about it!

This brings up another question. I read in the June 12, 1989 issue of *ComputerWorld* that IBM has urged the FCC to close the "loophole" allowing user-assembled personal computers to be sold without certification. How can we urge the FCC not to create the same problems in this area as they have with factory-assembled computers? Will they listen if individuals write to them?

Dan Barr  
Napa, CA

FCC certification is, at present, an expensive process for a small company. We suggest you get a copy of the **FCC Rules and Regulations, Part 15**. This is available from the Government Printing Office, Washington, D.C. The last time we checked, the price for this particular publication was around \$15. The rules will tell you the engineering requirements for certification, as well as the administrative details you'll have to take care of. There are a lot of rules, but there are also a lot of wrong public impressions about what the rules say. Your device or product should be compared to the rules relating to its particular product category.

The FCC does welcome public comment, and you as a citizen have the right and duty to inform them of your position on matters relating to their administration. If you follow the comment procedure, they must consider your input. This doesn't mean that they will necessarily produce a rule that favors you, but if there are aspects of rulemaking that are incongruous

or unduly burdensome to a small business such as yours, you have a good chance of getting at least some attention. To have a real impact, though, you must become informed of exactly what the FCC is doing, and that is rarely reported with any accuracy in the news media. A local library might have the Federal Register as a periodical. By law, all proposed rules and administrative changes must be reported within the registers so that the public may comment.

A good source for help in obtaining information from government agencies is the book "Information U.S.S." by Matthew Lesko. You can find this in most national chain bookstores.

In Visible INK, the Circuit Cellar INK Research Staff answers microcomputing questions from the readership. The representative questions are published each issue as space permits. Send your inquiries to:

INK Research Staff  
c/o Circuit Cellar INK  
Box 772  
Vernon, CT 06066

All letters and photos become the property of CCINK and cannot be returned.

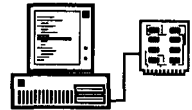
---

## IRS

- 201 Very Useful
- 202 Moderately Useful
- 203 Not Useful!

## The Ultimate in Embedded Systems Programming.

**Remote DSD87** is the most advanced source debugger for embedded systems. This package and your compiler form a complete solution for embedded development. Supports Microsoft and Turbo C. Includes a **FREE** copy of Native **DSD87!**



**ROM-Link** is the fastest link and locate utility for use with Remote debuggers, In Circuit Emulators, and EPROM Programmers. Supports Microsoft debugging information.



**Call Our Bulletin Board System at (213) 559-1449 for FREE Demonstrations!**

 **SoftAdvances**  
10811 Washington Bl. Ste. 205 • Culver City, CA 90232 • (213) 559-7015

Reader Service # 151

## Writing Network Applications

*The Subtle Art of Programming for INKnet*

The first two articles in this series covered the “how and why” of networks. With that under your belt, I can now explore programming techniques for applications that exchange control and status information using INKnet. Rather than an academic discussion, I will describe four programs that cooperatively manage a single control problem.

The sample project is a 100-liter mixing tank. While this is typical of many process control problems, I’ve made some simplifying assumptions to keep the programs manageable.

Despite the fact that INK’s readers are hands-on types, it is unreasonable to expect you to tackle a major plumbing project just to see how INKnet works. Ordinarily, one of the RTC52 boards would monitor the sensors and drive the valves and mixer motors, but I added some BASIC code to simulate the temperature measure-

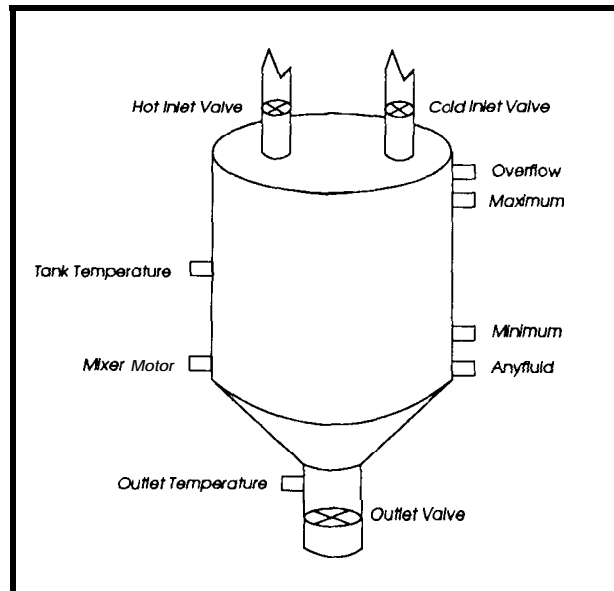


Figure 1 — The hypothetical tank has inlets for hot and cold, an outlet, sensors for content level, a mixing motor, and temperature sensors. It is designed to mimic a “real world” tank application.

ments and produce fake switch values. You can thus run the whole show on your desktop without the faintest risk of drowning your keyboard.

### SIZING THE PROBLEM

Figure 1 shows our mixing tank, which has both “upstream” and “downstream” connections to other parts of the manufacturing process that we’ll ignore here. There are two inlet valves that provide hot and cold fluid and one outlet valve that drains fluid. We control the inlet valves, but the outlet is opened and closed by the machine drawing fluid from the tank. Our job is to maintain enough fluid at the correct temperature to keep the downstream machine contented whenever it needs juice.

There are four fluid level switches along the side of the tank, so we can

determine how much fluid is present, albeit with limited precision. A pair of temperature sensors report on tank and outlet temperature, which may be different if the tank wasn’t stirred while adding fluid. The mixer motor is under our control and should be turned off whenever it isn’t needed.

Figure 2 diagrams the network nodes and I/O connections needed for the project. The Tank Simulator (Node 2) and Control (Node 1) RTC52 boards are the only truly essential ones, because you can control the outlet valve manually using the Tank Simulator’s console interface.

The RTCIO expansion boards are used mainly as outputs, so you can omit them and read the results from the AT’s display. The Control node reads Time-of-Day information from the Real-Time Clock on its I/O expansion board, but the code will continue

to work with bogus values if the chip is missing.

## TIMING IS EVERYTHING

As you should know by now, there is an intimate relation between the data rate required by your programs, the network bandwidth, and the success of your project. For much the same reasons you can't win the Indy 500 in a Yugo, you can't commute to work in an Indy racer: you must make sure that you're using the right tool for the job. In this case, we must match the network's data rate against the problem's control requirements.

Our tank holds up to 100 liters of fluid; the Minimum and Maximum level sensors change state at 40 and 75 liters, respectively. The Overflow and Anyfluid sensors inform you that the floor is getting wet or the tank has run dry, but under normal conditions Overflow is OFF and Anyfluid is ON.

The Outlet valve drains 0.3 liters/second from the tank. The Cold inlet can also supply 0.3 l/s, but the Hot inlet is limited to 0.2 l/s. While these sound like trickles, 0.3 l/s is about 4.75 gallons/minute.

The Outlet temperature must be maintained between 55° and 65° C, but the Hot fluid arrives at 200° and the Cold fluid at 0°. Obviously, the fluid isn't water; think of lubricant or feedstock chemical. The Mixer stirs 5 liters/second of tank contents, although the simulation simply adjusts the Outlet temperature toward the Tank temperature by the ratio of the Mixer flow during one simulation step to the total tank contents.

The Control node must maintain the temperature without overflowing or emptying the tank. Because the only level information comes from the four sensors, the control algorithm must shut the inlets off when the Maximum sensor goes on and turn them off when Minimum shuts off.

Adding 0.5 liters per second to a tank that is about half full results in a 1% volume change per second. The Hot and Cold flows and temperatures combine to add 133-degree fluid to the 60-degree tank contents; the corresponding temperature change is,

again, about 1% per second. The allowable temperature variation is 10%, so the network has on the order of 10 seconds to respond to any changes.

Although I have not described the message format yet, most of the network traffic will be polls and short responses. As you saw in the last article, a short INKnet message requires about 15 milliseconds, so each node in a 5-node network will be polled 6 or 7 times per second. Because the Control node must exchange several messages with the Tank Simulator

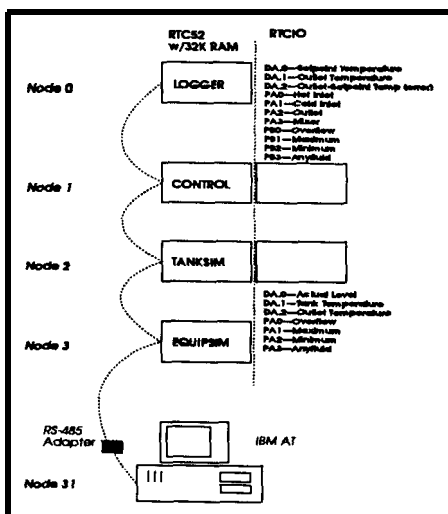


Figure 2—The network nodes and I/O connections needed for this project. The Tank Simulator (Node 2) and Control (Node 1) RTC52 boards are the only truly essential ones, because you can control the outlet valve manually using the Tank Simulator's console interface.

node, the response time will be about a second and the control system will have time to stabilize the system.

If your system changes by 10% in half a second, a controller responding in a second will not be able to hold the system within 10% of the setpoint. INKnet is suited for industrial problems that change relatively slowly; it is not the backbone network for a fly-by-wire system!

## PASSING MESSAGES

Figure 3 shows the message flow between the four RTC52 nodes. In effect, there are four simultaneous program loops working on four different problems, each one swapping data with the others as needed. All four

BASIC programs are also updating their console outputs and scanning for keyboard input, so there is a lot going on under the covers.

Each network message includes an opcode that indicates the purpose of the message and the format of any data following the header bytes. The firmware handles all of the console I/O, polling, and network status messages; the application programs can concentrate on their own information.

Figure 4 shows the opcodes associated with each network message. Opcodes between 10 and 1F hex are commands sent to a node to elicit a specific action. That action will trigger a response message with an opcode 10 hex greater than the command opcode. The response does not directly trigger any network traffic.

For example, the Control node sends opcode 11 hex with a single "zero" data byte to the Tank Simulator to close the Cold inlet valve. The Tank Simulator responds with opcode 21 hex and "zero" data to acknowledge that the valve is shut, or "one" to indicate that it isn't. The Control node thus is assured that the commanded action has actually taken place.

The INKnet firmware does not guarantee that messages will reach their destination, so your programs must handle this function. The sample programs retransmit the most recent message if the acknowledgement has not arrived within two seconds.

The Control node must send and receive three messages to collect the current valve, switch, and temperature status from the Tank Simulator node. I chose this rather chatty protocol to provide an excuse to discuss building a state machine to manage network traffic. In your applications you should minimize network traffic by combining as much information as possible into a single message.

## RUNNING IN LOOPS

Because the programs must respond to network messages at any time, they have a different structure than you may have used for your BASIC applications. Figure 5 shows a generic flowchart that applies to each

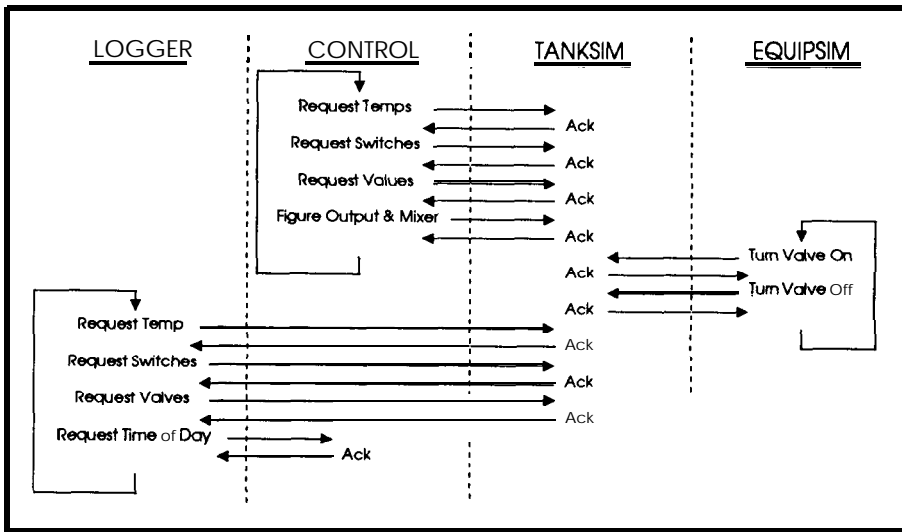


Figure 3—The message flow between four RTC52 nodes. In effect, there are four simultaneous program loops working on four different problems, each one swapping data with the others. All four programs are also updating console outputs and scanning for keyboard input, so there is a lot going on under the covers.

of the four programs, and may apply to your new networking code as well.

Each program handles three separate tasks: watching for incoming messages, handling local control functions, and checking the console for keystrokes. You must give each task control often enough to avoid missing network messages, bungling the control task, or skipping console keystrokes. There are practical constraints that prevent your code from reaching this ideal, but the goal is clear.

BASIC-52's console I/O is the weakest link in the flowchart, because your program must stop processing both network messages and local functions when it executes an `INPUT`

statement to get keystrokes from the network. BASIC-52 isn't a multi-tasking language (what do you expect in an 8K interpreter?), so there is no way to continue processing while snatching keystrokes from the console. The only way to manage this is to ensure the remaining nodes do not "hang" forever when another node is busy.

Listings 1 through 3 correspond to the three main decision blocks of Figure 5 and together form the skeleton of the Control node program. I have omitted some code to reduce the size of the listings, but the full source for all four programs is available on the BBS. [Editor's Note: Software for

this article is available for downloading from the Circuit Cellar BBS and on *Software On Disk #12*. For downloading and ordering information, see page 86.1

The first few lines of Listing 1 adjust `MTOP` and set up pointers to three message buffers. There are two outgoing buffers (SBF1 and SBF2, for "Send Buffer") to handle both responses and original messages, but only one buffer (RBF1, for "Receive Buffer") for incoming messages.

The number of message buffers depends on the amount of simultaneous traffic your applications must support. Each node may send one message during a polling cycle, but the message will be blocked if the receiver is busy. If your code communicates with several nodes, it must ensure that a (temporarily) busy node doesn't hang up messages to the others. The easiest way to handle this is to maintain copies of the messages until they're acknowledged by the receiver.

The responsiveness of your program depends on how much work it does on each pass through the main loop. The sample programs make 3-6 loops each second, so the code can handle that number of messages per second without overruns. If higher rates are needed, you must check for incoming messages more frequently, perhaps by embedding subroutine calls at several locations in your code.

## RECEIVING AND RESPONDING

Because receiving a message from another node and responding to it are critical to network applications, I'll start by describing how to implement this task. There are two parts to the problem: capturing an incoming message, then transmitting the response back to the originating node.

Lines 10010 and 10020 check the network interface status on each pass through the main loop. These programs use only the "message pending" status bit, but you can examine other bits to tally checksum errors or other network events.

If an incoming message is present, lines 10100-10120 load it to RBF1. If two or more messages are received before `NET INPUT` executes, the status

Opcode	Meaning	Data
10	Set Hot inlet valve	0=closed, 1=open
11	Set Cold inlet valve	0=closed, 1=open
12	Set Outlet valve	0=closed, 1=open
13	Set Mixer motor	0=off, 1=on
18	Request valve/mixer status	none
19	Request level switch status	none
1A	Request tank & outlet temp	none
1D	Request current time of day	none
20	Ack Hot inlet valve	0=closed, 1=open
21	Ack Cold inlet valve	0=closed, 1=open
22	Ack Outlet valve	0=closed, 1=open
23	Ack Mixer motor	0=off, 1=on
28	Return valve/mixer status	4 bytes as above
29	Return level switch status	4 bytes as above
2A	Return tank & outlet temp	2 bytes
2D	Return current time of day	6 bytes: MDDYYHHMMSS

Figure 4—These opcodes identify the contents of the messages sent from one node to another. When a node responds to a message it ORs the incoming opcode with 10h and returns it to the originator along with the appropriate data. The first node thus has a positive acknowledgement that the message was delivered correctly. The data bytes follow the message header. Not all nodes respond to all opcodes.



```

100 rem - allocate message buffers and string space
110 net workarea
120 pop wa : mtop=wa-1-3*256
140 dim t(15) : rem for TOD value
150 sbf1=mtop+1 : sbf2=mtop+257 : rbf1=mtop+513

<<< code omitted >>>

10000 rem - network interface
10010 net status
10020 pop s1
10021 if (s1.and.01h)<>0 goto 10100 : rem new incoming msg?
10022 if (rsnd.and.01h)=0 goto 20000 : rem resend old msg?
10024 if time>tm41 gosub 41100
10026 goto 20000

10100 rem - grab & dispatch net message
10110 net input,rbf1
10120 pop s1 : on s1 goto 10130,10140,10122
10122 print "*** Message overrun!" : rem fall through & decode

10130 if (rsnd.and.1)=0 goto 10150
10132 print "*** New message while SB1 in use, old SB1 discarded"
10134 rsnd=rsnd.and.not(1) : tm41=0 : goto 10150

10140 if time>tm41 gosub 41100 : goto 20000

```

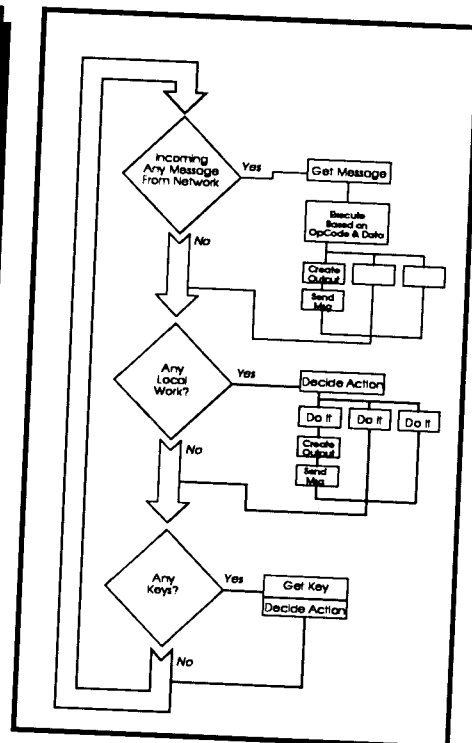
(continued)

**Listing 1**—This code allocates the network buffers, then checks the interface for incoming messages and dispatches them to the appropriate routines. Those routines may generate a response message, which may in turn require retransmission if the target node is busy.

variable indicates an overrun error. Because we have already checked for a message before executing NET INPUT, line 10400 should never execute; it never hurts to be defensive, though.

An incoming message may trigger an outgoing response, so if a sec-

ond message arrives before the first response was sent the outgoing message will be overwritten. Lines 10130 through 10134 detect this event. In fact, opcodes 20 through 2F do not produce responses, so this test could be tightened up a bit. You could also



**Figure 5**—This is a basic generic flowchart for node traffic on the network. It works to describe the general message flow for this project, and will probably describe most problems you work on, as well.

## Introducing the MICROMINT T-286 CPU

Micromint announces the latest addition to its line of high-quality CPUs for industrial and OEM applications: The Micromint T-286. This IBM PC/AT-compatible computer packs the features our customers requested most into the single expansion card format made popular by the OEM-286. Add the quality and support that have made Micromint famous, and you have a CPU that can't be matched for total price, features, and performance!

The T-286 is 100% AT-compatible, with clock speeds switchable (via jumpers or keyboard) between 6 and 12 MHz. Among the features offered on the T-286 are an on-board real-time clock, socket for an 80287 numeric coprocessor, on-board keyboard connector, and 4 megabytes of on-board RAM capacity. The T-286 features the industry-standard Award BIOS, and offers OEM customers the advantage of a 32K ROM, expandable to 64K for custom applications. ISA Bus compatibility is assured, with the T-286 taking up a single slot in an ISA Bus passive backplane, and requiring only +5V power for operation.

Low-power operation; speed; maximum configurability—the T-286 is the perfect choice for critical applications where ISA Bus expansion or MS-DOS software development are spec'ed. Call Micromint today for more information on the T-286 CPU and custom system availability.

T-286/0 — 12 MHz 80286 CPU w/OK RAM

**\$399.00**

T-286/4 — 12 MHz 80286 CPU w/ 4MB 80-ns, 0-wait-state RAM

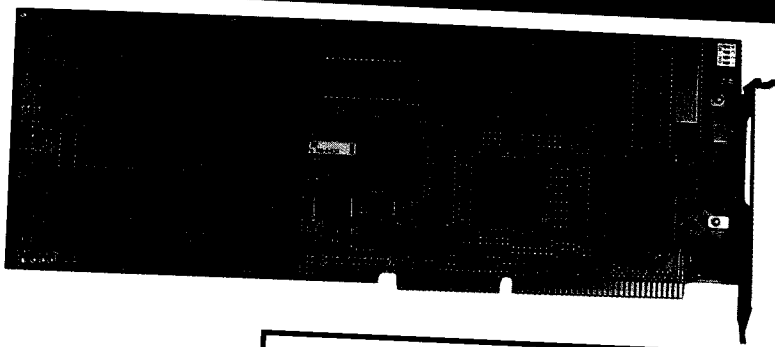
**Call for Prices**



**MICROMINT, INC.**

4 Park Street, Vernon, CT 06066

Tel: (203) 871-6170 • Fax: (203) 872-2204



### T-286 CPU Technical Specifications

- 100% IBM PC/AT compatible
- 80286-12 processor, operating at 6/12 MHz, 0 or 1 wait state
- Speed selectable with keyboard or jumpers
- 256K, 1 meg, or 4 megs of on-board RAM
  - can use 4164, 41256, or 411000 DRAMs
- Battery-backed real-time clock (DS1287) and configuration RAM
  - 10-year on-board battery
- Socket for 80287 Numeric Coprocessor
- 32K ROM, expandable to 64K
- Award BIOS
  - Configuration and setup routines in ROM
- On-board keyboard connector (5-pin DIN)
- On-board tone beeper
- Needs only +5V for operation
- Single-slot, full-length expansion board form-factor
- ISA Bus interface

```

10150 op=xby(rbf1+3)
10152 if (op>=10h).and.(op<=1fh) goto 10180
10154 if (op>=20h).and.(op<=2fh) goto 15000
10160 print "*** Bad opcode:",
10162 ph0.op: goto 20000

10180 rem - dispatch opcodes 10-1fh
10190 if op<18h on op-10h goto 14990,14990,14990,
14990,14990,14990,14990,14990
10192 on op-18h goto 14990,14990,14990,14990,14990,
11500,14990,14990

11500 xby(sbf1+1)=16: rem send current TOD back out
11502 gosub 40400
11504 xby(sbf1+10)=t(9)*16+t(8): xby(sbf1+11)=t(7)*16+t(6)
11506 xby(sbf1+12)=t(11)*16+t(10)
11508 xby(sbf1+13)=t(5)*16+t(4): xby(sbf1+14)=t(3)*16+t(2)
11510 xby(sbf1+15)=t(1)*16+t(0)

12000 rem - set up rest of message
12010 xby(sbf1+2)=xby(rbf1+4):rem dest = incoming source
12020 xby(sbf1+3)=xby(rbf1+3)+10h: rem set opcode
12030 gosub 41100 : goto 20000

14990 print "*** Unused opcode:",
14994 ph0.op: goto 20000

15000 rem - dispatch opcodes 20-2fh
15100 if op<28h on op-20h goto 15200,15210,15220,
15230,19990,19990,19990,19990
15110 on op-28h goto 15300,15310,15320,19990,19990,
19990,19990,19990
15200 valve1=xby(rbf1+10): rem Hot inlet acknowledgement
15204 ctlstate=0 : tm8=0
15206 goto 20000

<<< code omitted >>>

15320 temp1=xby(rbf1+10) : temp2=xby(rbf1+11) : rem Temp status
15324 ctlstate=2 : tm8=0
15326 goto 20000

<<< code omitted >>>

19990 print "*** Unused opcode:",
19994 ph0.op : goto 20000

```

Listing 1 -continued

allocate more receiving buffers to handle higher loads; each new incoming message would go into the next available buffer for later processing.

Lines 10150 through 10154 extract opcodes from the message buffer and pass control to decoding routines. If the opcode isn't in the right range, lines 10160 and 10162 tell you about it. The firmware doesn't pass opcodes 80 through FF hex to your programs, so you don't have to process polls or other network messages.

Lines 10180 through 10192 and 15000 through 15110 dispatch control to the lines responsible for each opcode. Not all nodes handle all opcodes, so there are two dummy routines to report any messages with unused opcodes.

After all that, the actual processing associated with a message is almost anticlimactic! Foreexample, Lines 11500 through 11510 respond to opcode 1D hex by extracting the current time of day from the Clock/Calendar chip on the RTCIO board and creating

a return message. The subroutine at line 40400 isn't shown, because it is similar to the one in the manual.

Lines 12000 through 12030 fill in the message header information common to all responses from this node and call the routine that sends the response back to the originator. Regardless of whether the message was sent successfully or not, control passes to line 20000.

Listing 4 shows the code needed to both send a message and handle retries. Line 41102 tumsona bit in the `RSND` variable and sets a timeout that will trigger a **retry** after a failure. Lines 41104 through 41112 make sure that neither node is currently busy handling another message, while lines 41120 through 41132 send the message and check for "can't happen" errors. The last line turns off the retry flags to indicate that the message was sent successfully.

Lines 10022 through 10026 in Listing 1 should now make more sense. They check to see if a resend is needed

# INTROL CROSS DEVELOPMENT SYSTEMS

SAVE Development  
and Debugging Time  
of Embedded  
Microprocessor Systems!

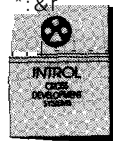
- INTROL-C Cross-Compilers
- INTROL-Modula-2 Cross-Compilers
- INTROL Relocating Macro Cross-Assemblers

COMPILER PACKAGES INCLUDE:  
Compiler • Assembler • Linker  
• Runtime library, including a multi-tasking executive • Support utilities • Full year's maintenance

TARGETS SUPPORTED:  
6301/03 • 6801/03 • 6809 •  
68HC11 • 68000/08/10/12 •  
68020/030/881/851 • 32000/  
32/81/82

AVAILABLE FOR FOLLOWING  
HOSTS: VAX and MicroVAX;  
Apollo; SUN; Hewlett-Packard;  
Macintosh; Gould Power-  
Node; IBM-PC, XT, AT, and  
compatibles

INTROL CROSS-DEVELOPMENT  
SYSTEMS are proven, accepted  
and will save you time, money,  
and effort with your develop-  
ment. All INTROL products are  
backed by full, \* & r  
meaningful,  
technical support.  
CALL or WRITE  
for facts NOW!



**INTROL  
CORPORATION**

647 W. Virginia St.  
Milwaukee, WI 53204  
414/276-2937 FAX: 414/276-7026  
Quality Software Since 1979

and pass control back to Listing 4 to try again. The retries will continue until the message is finally sent off. Notice, however, that there is no "backup" retry built into the code because the recipient does not acknowledge the acknowledgement.

The code omitted from Listing 1 is simply more of the same: lines similar to the chunk at 11.500 that perform whatever actions are called for by each opcode. If your application uses many messages, you may need to have an auxiliary opcode as the first data byte that is decoded after the main header opcode. You may also need to use the message length byte to determine how much data is contained in messages with variable lengths; these applications use only fixed length messages, so no tricky code is required.

## DOING USEFUL WORK

A network application is no different than a single-board application, except that the sensor and output ports are on a node across the network.

Instead of reading and writing I/O ports directly, the control program must compose, send, and receive messages to achieve its goals.

However, a network does not respond immediately to a message. The elapsed time between an outgoing message and the incoming response may be an appreciable fraction of a second or more, particularly under heavy network loads. The control program cannot simply "stall" after sending a message, because it may miss incoming messages from other nodes. Obviously, a different programming strategy is required for network applications.

The secret is to divide the control program into distinct, quickly completed states. The transitions between states occur when the program starts a lengthy process such as sending or waiting for a network message.

Listing 2 shows the first two states of the Control node's program, along with the overhead needed to make it work. I have omitted most of the code to highlight the critical sections.

`CTLSTATE` indicates which state the computation is in at each pass

through the main loop. Lines 20050 and 20052 give control to the appropriate lines on each pass.

For example, the Control node starts a new control cycle whenever the current time exceeds the value of `TM2`. This action is handled in the first `State`, `CTLSTATE=0`, in lines 21000 through 21006 of Listing 2. Notice that the code simply bypasses all computations until the triggering event occurs.

States 1 through 4 request status information from the Tank Simulator node. Each state starts by composing and sending the request message and ends when the acknowledgement returns. State 5 actually performs the control algorithm and may send a message to the Tank Simulator to adjust the valves or mixer. When that action is acknowledged, the program returns to State 0 and waits for the start of the next cycle.

The key point is that the main loop continues to execute even while the program is waiting for something to happen. You may not use "busy loops" to delay for a few seconds; make that interval a separate state and use the timer function to trigger the next state. Your program must respond to any incoming messages even when it's waiting!

State 1 is divided into two sections. The first section, lines 21103 through 21108, **sends** out the tank temperature request message and sets `TM8` to the timeout value. Each subsequent main loop iteration will pass through lines 21150 through 21180. If an acknowledgement isn't received from the Tank Simulator node within `TM8` seconds, line 21180 will resend the message and reset the timeout.

Notice that there are two timeout values at work at the same time. Variables `TM42` and `RSND` determine

```

20000 rem - control loop
20010 if ((rsnd.and.2)=0).or.(time<tm42) goto 20020
20012 tm8=time+tm7: gosub 41200 : goto 29000
20020 rem - no pending messages

20050 if ctlstate<5 on ctlstate goto 21000,21100,21200,
      21300,21400
20052 if ctlstate<10 on ctlstate-5 goto 21500,21600,21700,
      21800,21900

21000 rem - state 0, delay for control interval
21002 if time<tm2 goto 29000
21004 print : print ">>>>> Starting control cycle"
21006 tm2=time+tm1 : goto 28000

21100 rem - state 1, ask for temp status
21102 if tm8<>0 gotb 21150
21103 .print "Requesting temperature status"
21104 xby(sbf2+1)=10 : xby(sbf2+2)=tsnode : xby(sbf2+3)=1ah
21106 gosub 41200 : tm8=time+tm7
21108 goto 29000
21150 rem - waiting for response
21160 if time<tm8 goto 29000
21170 .print "** Timeout waiting for temperature status!"
21180 rsnd=rsnd.or.2 : tm42=time : goto 29000

21200 rem - state 2, ask for switch status
21202 if tm8<>0 goto 21250
21203 print "Requesting switch status"
21204 xby(sbf2+1)=10 : xby(sbf2+2)=tsnode : xby(sbf2+3)=19h
21206 gosub 41200 : tm8=time+tm7
21208 got.0 29000
21250 rem - waiting for response
21260 if time<tm8 goto 29000
21270 print "** Timeout waiting for switch status!"
21280 rsnd=rsnd.or.2: tm42=time : goto 29000

<<< code omitted >>>

28000 rem - step to next state
28010 ctlstate=ctlstate+1: goto 29000

<<< code omitted >>>

29000 rem - update monitor outputs

<<< code omitted >>>

```

listing 2— This code performs the 'useful work' ordinarily associated with a microcontroller. The work is divided into distinct states, which are called in turn by the code near line 20000. Transitions between states are controlled by external events, elapsed time, or other factors; the code cannot 'wait forever' in a tight loop.

when to retry a message that hasn't been sent yet, using the code in lines 20010-20012. Once the message has been sent, that timeout is no longer in effect. TM8, on the other hand, resends the message if the Tank Simulator doesn't respond within the timeout.

TM8 is the reason why nodes do not need to acknowledge acknowledgements: it is an overall timeout that resends the original message if the acknowledgement is not received. The originating node resends until it gets an answer, regardless of who dropped the ball.

Lines 15320-15326 in Listing 1 show what happens when an acknowledgement arrives. The CTLSTATE is set to the appropriate value and TM8

```

30000 rem - console inputs
30010 c9=get
30020 if (c9<>20h).and.(c9<>0dh) goto 39900
30030 rem
30100 print : print "====="
30110 print "Control Node Variables"
30120 print "*** Execution stops until you select an entry!"
30130 Print "Options (use lowercase letters only):*"
30131 Print using(###.#)," a Outlet setpoint",temp5,"(deg C)"
30132 print " b Setpoint deadband",temp6
30133 print " c Update interval",tml,"(seconds)"
30134 print " d Message trace",trace
30190 if c=20h goto 39900 else print "Enter letter: ",
30192 gosub 40100 : pop c : if (c<61h).or.(c>64h) gosub 40150 :
      got0 30190
30194 print : print "Enter value:" : gosub 40200
30196 if c<65h on c-61h goto 30200,30210,30220,30230 else pop
      c9 : goto 30190
30200 pop temp5 : goto 32500
30210 pop temp6 : goto 32500
30220 pop tml : goto 32500
30230 pop trace : goto 32500

32500 rem - ensure console is drained...
32510 if get<>0 goto 32510

39900 rem - back to the top of the Main Loop!
39940 got0 10000

40100 rem - get 6 push char
40110 c9=get : if c9=0 goto 40110 else print chr(c9) : push c9 :
      return
40150 rem - beep!
40160 print chr(7),: return
40200 rem - get & push number
40210 input c9 : push c9 : return

```

Listing 3—This code watches for keys at the console input and displays a menu of choices. Because BASIC does not have a multitasking INPUT statement (few languages do!), the program stalls until a menu item is selected. The final statement branches back to the top of the loop in f/sting 1.

entered. The remaining lines stuff the value into the appropriate variable.

This suffices for programs that can be interrupted for brief intervals. Because the other nodes don't know that this node has stopped processing messages, they continue to send messages. The first message will be received correctly, but then the node will then be BUSY until the main loop resumes. The sample programs simply resend their messages until that time; you may want to investigate a more complete solution.

Now you have everything you need to get started writing a network

application: the hardware, the firmware, a few sample programs, and that hankering to write The Great American Application that got you into this business in the first place.\*

Ed Nisley is a member of the Circuit Cellar INK engineering staff and enjoys making gizmos do strange and wondrous things. He is, by turns, a beekeeper, bicyclist, Registered Professional Engineer, and amateur raconteur.

## IRS

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

single-

## CONSOLE I/O

As I mentioned above, console I/O is the weak link in the network. Listing 3 shows one way to work around the limitations.

Lines 30010 and 30020 use GET to snag a waiting character from the console. If there is no character, the function returns a zero and the code continues along the main loop. A space or carriage return triggers the menu display and the program will stall in the subroutine at line 40100 until the user enters a letter. If it's one of the menu choices, the subroutine at line 40200 stalls until the new value is

```

41100 rem - send msg from buffer 1
41102 rsnd=rsnd.or.1 : tm41=time+tm3 : rem assume busy
41104 net status
41106 pop s1 : if (s1.and.0ch)=0 goto 41110
41108 if trace print "*** Response pending" : return
41110 net status,xby(sbf1+2)
41112 pop s2 : if (s2.and.80h)=0 goto 41120
41114 if trace print using(##),"** Node",xby(sbf1+2),"busy" :
      return
41120 net output,sbf1
41122 pop s3 : if s3=0 goto 41130
41124 print using(##),"** Error",sb3,"sending message!"
41130 if trace print using(##),"- SBI to ",xby(sbf1+2),"op",
41132 if trace ph0,xby(sbf1+3),: print " data", :
      ph0,xby(sbf1+10)
41134 rsnd=rsnd.and.not(1) : tm41=0 : return

```

Listing 4—This routine sends (or resends) a message from the 256-byte buffer at SBF 1. These messages are responses to incoming messages and are created by code similar to lines JO100- JO122 in listing 1. The code near line 10000 determines if a retransmission is needed because the call to this routine did not complete successfully.



## IMAGE SYNTHESIS: A TUTORIAL

*Tools for Drawing a New Universe*

by Chris Ciarcia

**W**hether programmer or painter, the modern artist can stimulate the imagination, create a message, or pass on a thought within the world of computerized image synthesis. Instead of a palette of paints, he uses a high-resolution computer graphics screen, lighting colored pixel arrays instead of layering paintbrush strokes. Using image synthesis techniques to enhance the media, he can visualize a design, study a structure, or visually test a theory in a most realistic fashion.

The goal is using computer graphics drawing techniques to produce realistic pictures, indistinguishable from photographs



of real "or imagined" objects. The artist simulates the optical processes of the real world by modeling the physics of light propagation and its interaction with matter. Effects such as mirror reflections, fuzzy reflections, glossy reflections, transparency, refraction, indirect lighting, color bleeding, caustics, directional light sources, smooth shadows with umbra and penumbra, motion blur, depth of field, surface textures, and fog and haze are fair game for "realistic" visualization.

Anyone who's been to the movies in the last five years knows how far computerized image simulation can take us. The question is, how does it get us from here to there?

## LIGHT IN THE REAL WORLD

Image synthesis is the simulation of the optical processes at work in the real world—the physics of how light propagates and interacts with nature. Based on these processes, creating a "realistic-looking image" is achieved in three stages: image formation, object illumination, and shading. Image formation is the determination of what's in the picture and how it's to be seen. Such operations as "the hidden surface" problem are addressed here as well as the calculation of reflections from mirrored surfaces and the refractions through transparent objects. Illumination is the determination of image components that are dependent solely on the propagation of light throughout the scene, while being independent of the viewpoint. Such effects as shadows or internal reflections (both diffuse and specular) from objects and light passing through transparent or translucent objects are incorporated here. And finally, shading is the process of determining the distribution of light leaving a surface, determined by specifying the nature of

the incident light and the optical properties of the surface.

Incorporating all of these "light manipulations" into one single image

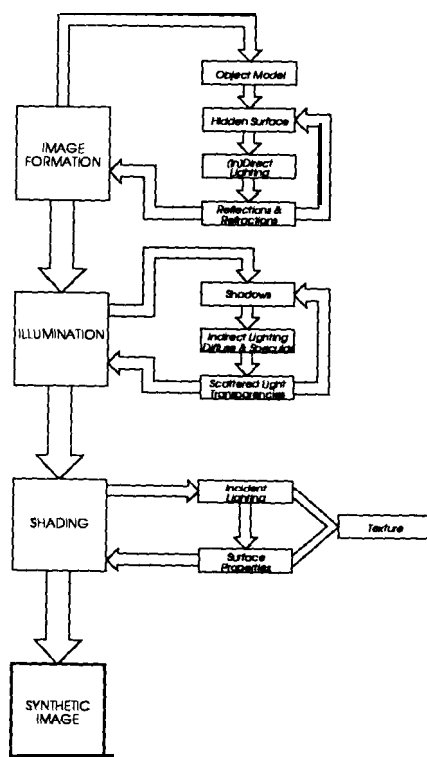


Figure 1 -There are three main steps in image synthesis: Image formation, which is dependent on viewer point of view; Illumination, which is point of view independent; and Shading, which considers the surface properties of the image.

synthesis system is an elaborate undertaking. At present I know of no one integrated image synthesis system on the market that does a truly great job in all areas. There are just so many effects that can be modeled to

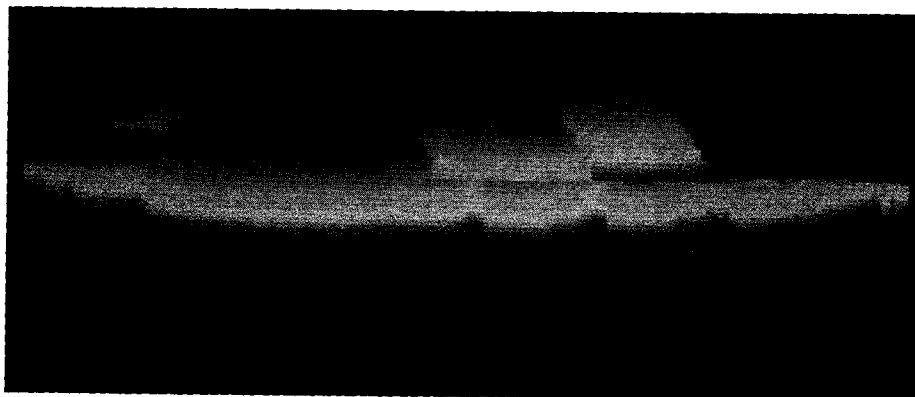
various degrees of image-realism effectiveness. The possibilities range from single-viewpoint models to elaborate statistical algorithms such as Monte Carlo techniques to solve truly intractable modeling problems.

Before we can start to create synthesized images, we have to understand the relationships between light, the viewed objects, the monitor on which we wish to display our image, and the device, be it our eyes or some optical recorder, that views the image. Then we must make the best modeling assumptions that give the most ideal image in our quest for realism. From the start, the very use of a CRT immediately places us at a disadvantage. It imposes a 2-D format, limited image size, limited color selection, fixed spatial resolution, and a limited intensity range on our stated goal. Are we defeated? No! If my daughter can model the family car on the bedroom wall, can I do less on my '386?

## SYNTHESIS

Generating imagery on a computer involves creating data representations, planning scene environments, computing visibility, shadows, shades, texture, and color, and then displaying the results. The basic algorithmic components of this artistic intent are: the determination of visibility, shadowing, shading, aliasing, texturing, and modeling, and then (finally) synthesis!

In our current format we can, of course, only consider the simplest and most idealized form of each of these synthesis areas, but these simplified descriptions of concepts and algorithms truly demonstrate the known, accepted, and fundamental strategies for creating realistic images with your computer.



The color photos in this article illustrate the concepts discussed. Each makes use of the ray-tracing, shadowing, and surface texture techniques to give weight to objects composed of varying slices of standard geometric shapes.

## 8031 $\mu$ Controller Modules

NEW...'

### Control-R II

- ✓ Industry Standard 8-bit 8031 CPU
- ✓ 128 bytes RAM / 8 K of EPROM
- ✓ Socket for 8 Kbytes of Static RAM
- ✓ 11.0592 MHz Operation
- ✓ 14/16 bits of parallel I/O plus access to address, data and control signals on standard headers.
- ✓ MAX232 Serial I/O (optional)
- ✓ +5 volt single supply operation
- ✓ Compact 3.50" x 4.5" size
- ✓ Assembled & Tested, not a kit

\$64.95 each

### Control-R I

- ✓ Industry Standard 8-bit 8031 CPU
- ✓ 128 bytes RAM / 8K EPROM
- ✓ 11.0592 MHz Operation
- ✓ 14/16 bits of parallel I/O
- ✓ MAX232 Serial I/O (optional)
- ✓ +5 volt single supply operation
- ✓ Compact 2.75" x 4.00" size
- ✓ Assembled & Tested, not a kit

\$39.95 each

#### Options:

- MAX232 I.C. \$6.95 each
- 6264 8K SRAM \$10.00 each
- 8052BASIC CPU \$25.95 each

#### Development Software:

- PseudoSam 51 Software (\$50.00)  
Level II MSDOS cross-assembler.  
Assemble 8031 code with a PC.
- PseudoMax 51 Software (\$100.00)  
MSDOS cross-simulator. Test and debug 8031 code on your PC!

#### Ordering Information:

Check or Money Orders accepted. All orders add \$3.00 S&H in Continental U.S. \$6.00 for Alaska, Hawaii and Canada. Illinois residents must add 6.25% tax.

#### Cottage Resources Corporation

Suite 3-672, 1405 Stevenson Drive  
Springfield, Illinois 62703  
(217) 529-7679

## DETERMINING VISIBILITY

The most fundamental problem in image synthesis is visibility determination. If I view a set of intermixed objects, what portions of those objects do I observe? The answer to this is, of course, the old "hidden-surface" problem.

The most frequently solved problem of this type is one with a fixed viewpoint, as a single snapshot in time, using a single point-source of light. The algorithms for determining visibility are fairly straightforward, unless you are as masochistic as I and always try to incorporate the higher-order visibility problems such as motion blur, depth-of-field, penumbra, gloss or indirect illumination (footnote 1). But, within this simplified case, we can outline our problem as:

#### GIVEN,

- a specified viewpoint,
- \* an intermix of 3-D objects and their associated surfaces,
- \* the orientation of the image plane,
- the relative field-of-view,

#### DO FOR

- each point within the image plane within the field-of-view
- \* determine the closest surface point to the viewer on a line from the viewer's eye to the image plane

#### ENDDO

Two classes of algorithms, continuous and point-sampling, have evolved to determine this visibility. Each approaches the problem from a different angle. The continuous algorithms operate by performing visibility determination over continuous areas that entirely cover the image plane. Each and every individual point on each surface is evaluated regardless of location or size. Point-sampling algorithms create an approximate solution to the problem. They determine visibility at only a finite number of sample points and then

**[footnote 1]** I refer to a technique called distributed ray tracing. This is a multidimensional sampling technique where a sample is chosen by independent random variables in each dimension. As a result, the number of samples required for a given accuracy is greatly reduced compared to sampling on a fixed grid. This has allowed such phenomena as motion blur, depth of field, penumbra, and gloss to be efficiently simulated 12,391.

make assumptions, according to pre-defined rules, as to the visibility of surfaces between sample points.

In practice, continuous algorithms give better results, but their tremendous complexity, limited primitives, and wide range of special effects supported become computationally cost ineffective. Point-sampling algorithms, on the other hand, allow computational efficiency when handling many advanced effects and complex models, such as shading, multiple reflections, and refractions from curved surfaces.

Both forms operate using a very simplified model of light propagation. A light ray is assumed to travel in straight lines through homogeneous media. It is only allowed to interact with an object on its surface according to the basic geometric optical laws of physics. Such second-order effects as diffraction, phase, polarization, and scale-to-wavelength relationships are ignored. The models are time independent and contain no quantum effects. However, there is a class of "wavefront propagation" models that treat light as a complex-valued wavefront phenomenon incorporating these second-order effects. The results of such models have been generally discouraging. To handle the added complexity, they have by necessity restricted themselves to limitations on primitive sizes and spacing. As a result, I have decided to pass over them here. Having once been an optical design engineer, I shudder at the thought.

I tend to favor the point-sampling approach, so I will describe the four most basic forms of point-sampling algorithms that are currently used in the bulk of today's image synthesis work [2]. These are the ray-tracing, Z-buffer, painter's, and scan-line algorithms. Simplified versions of the algorithms are shown in the sidebar on page 29. Since I am most familiar with the ray-tracing technique, let's look at it in more depth.

Implementing a simple form of ray-trace algorithm is fairly straightforward. Draw a set of light rays of infinitesimal width from your point of view, through your object set, to each

pixel within your image field. Then test each ray against each object within your database and find the closest surface intersection point with respect to your point of view. Then assign a relative color and/or intensity value to that pixel location on your screen.

These days, however, nobody keeps things simple. The general trend within the ray-tracing community is to incorporate shadows, reflection, and refraction [1] into the ray-trace model. This is accomplished by generating a secondary set of rays whenever a primary ray intersects a surface, and tracing each ray within the secondary set from the intersection point, out toward each specified point-light source.

If the secondary ray encounters an object between surface and light source, that intersection point can be considered within its shadow. The associated pixel value is then diminished according to a specified illumination model. Each time a reflected ray intersects a surface, it generates a set of secondary rays, thus cascading throughout our scene. And a ray can also be traced in the direction of refraction, depending on the surface optical characteristics. As a result, this level of recursion in the algorithm indicates that an individually reflected or refracted ray can generate additional shadow rays, and so on.

A basic outline form for such an algorithm is:

```

DO FOR *each pixel
  *generate a ray R from the point of
  view, through the object space to the
  image plane
  DO FOR *each object 0 within the scene
    •calc. the intersections of 0 and R
  IF
    *the ray R hits no object
  THEN
    *set the intensity to background
    *go get the next pixel ray
  ELSE
    *find 0 with the closest intersection
    generate a secondary ray S from
    the intersection point to the light
    source
  IF
    *the S intersects another surface
    on the way to the source, it is in
    shadow
    -diminish pixel accordingly
    •go get the next pixel ray
  ENDIF
  • the point is not in a shadow
  *calculate the appropriate intensity

```

```

value
*go get the next pixel ray
ENDIF
ENDDO
ENDDO

```

Such algorithms which consider shadows, reflections, and refractions are fairly easy to implement. However, they tend to suffer from two primary difficulties: aliasing and computational speed. Aliasing occurs because of the inherent point sampling nature of the algorithm. And, the computational problem is blatantly obvious. The large number of ray-surface intersections can easily form a bottleneck in terms of actual run time.

The world of ray tracing seems to have an infinite dimensionality, and I don't have the space here for all the details. I suggest you read references [1]-[9] for more information. After muddling through the above, and after becoming totally confused, you will have noticed that the ray-trace algorithm is similar to the Z-buffer algorithm, and that, in turn, each is of the flavor of the other. The difference between the ray-trace and Z-buffer techniques only lies in the nesting order of their main loops. Where the ray-trace form handles one pixel at a time and then compares each object's depth, the Z-buffer algorithm handles

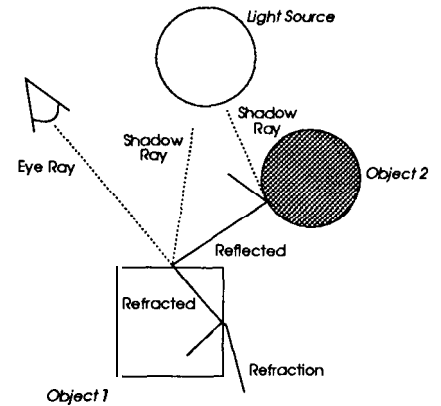
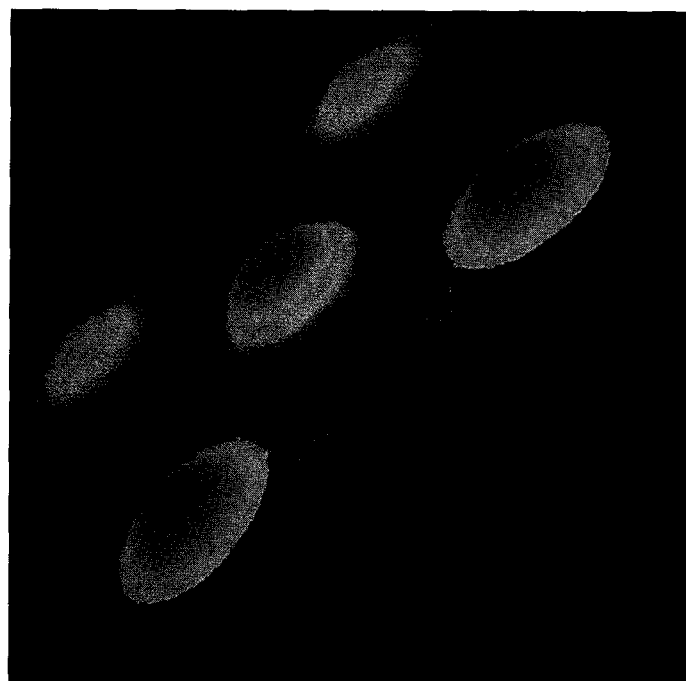


Figure 2—Shadows, reflections, and refractions are incorporated into the ray-trace model by generating a secondary set of rays whenever a primary ray intersects a surface. Each ray within the secondary set is traced from the intersection point to each specified point-light source.

each object at a time and studies each pixel that an object covers. The difference between the Z-buffer and painter's algorithms lies only in how depth comparisons are made. The Z-buffer makes comparisons throughout the inner loop calculation cycle, while the painter's algorithm makes all of its depth comparisons in a preliminary sorting phase. In turn, the scan-line algorithm only differs from the Z-buffer because it first divides the 2-D image into a sequence of 1-D "scan lines" and then applies a simpli-

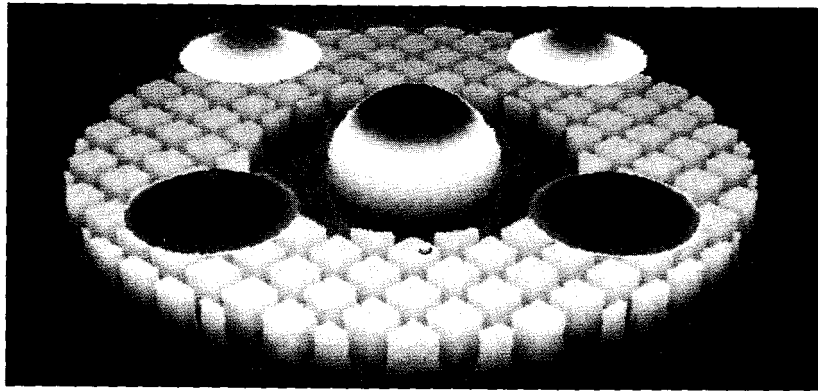


fied 1-D Z-buffer visible-surface algorithm to each line [2,9].

### SHADOWING

Within the world of Image Synthesis, shadowing is often referred to as the problem of direct illumination. The ultimate goal here is to determine what portions of your image are illuminated either directly or indirectly by light sources and what parts are obstructed, or in shadow. As I described in the previous section, one way to attack the problem is to incorporate shadow rays within your visibility point-sampling code. But you can also approach the problem from another perspective.

One technique I have tested and found to work effectively is called the "depth map" procedure. It was first proposed by Lance Williams, in a paper he gave at the 1978 SIGGRAPH convention [10]. It involves the calculation of a depth map, the Z-buffer image computed from the viewpoint of the light source. Then, for each pixel, the distance to the light source from the visible object is calculated and compared to the corresponding location within the depth map. If the depth map contains a lesser value,



shadow patterns added. Visible surfaces that are not illuminated by a light source have only ambient light intensity applied. Visible surfaces that are illuminated by a light source are shaded by combining the intensity model and the pattern arrays. Projected shadow areas are shaded with the ambient light intensity only.

then an object intervenes between that pixel of the visible object and the light source. The object is therefore in a shadow.

Depth mapping is a straightforward approach and can be applied redundantly to the image for each light source. It has been improved, as all older and proven techniques are, by the addition of stochastic supersampling and the storage of floating-point values within its depth buffer. If you wish to learn more of this technique and others, I refer you to references [2], [9], and [11].

The shadow patterns generated by any hidden-surface (source viewpoint visibility) technique are valid for any selected "actual" viewing position, as long as the light source positions are not changed. Surfaces that are visible from the view position are shaded according to the intensity model, with surface patterns and

projected shadow areas are shaded with the ambient light intensity only.

Just remember, as with all shadow algorithms, multiple light sources can be treated with independent applications of the same algorithm for each light source. However, too many light sources can easily bog down your illumination processing time.

### SHADING

The process of determining the light intensity leaving an object is called shading. The goal is to model the interaction of light with surface reflection properties so that we can specify an appropriate color and image intensity. As such, it is dependent on the incoming light intensity and distribution as well as the optical properties of the object. Indirect illumination can be handled by repeated application of local shading techniques

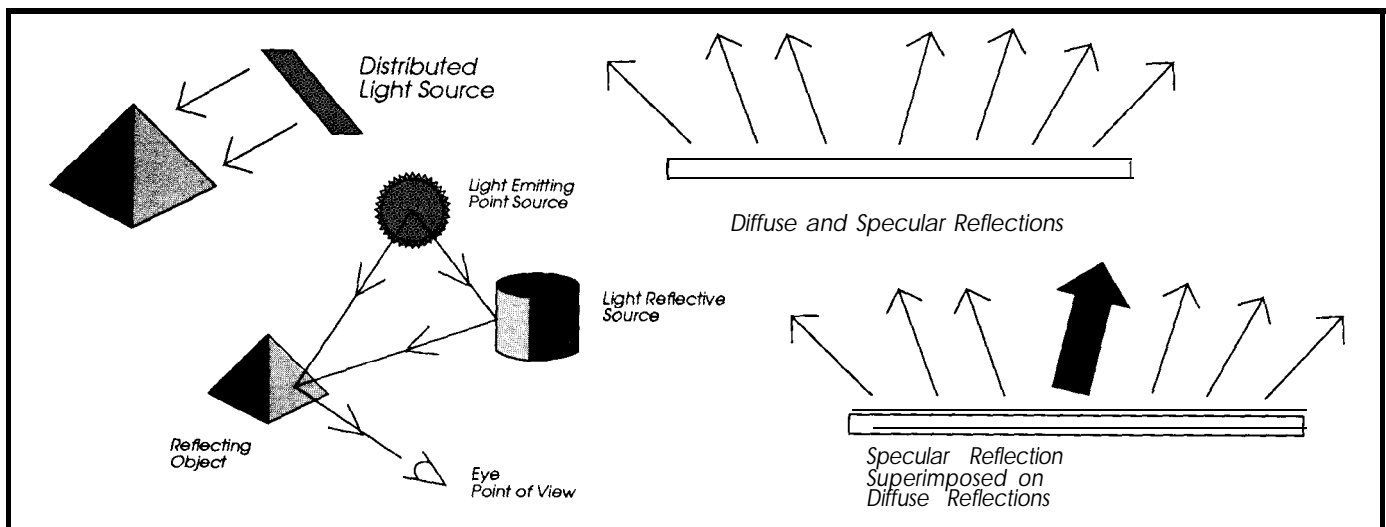


Figure 3—The intensity of light that we view from an image depends on both the direction of the light source and the reflection properties of the surface. At certain angles, shiny surfaces reflect all of the incident light. In a real object, the specular reflection within this narrow angle will be combined with other, more diffuse reflections.

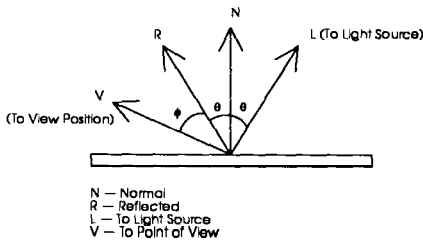


Figure 4—For a perfect reflector, the angle of incidence = the angle of reflection.

in order to incorporate light traveling between objects.

The intensity of the light that you and I see as the viewers of an image depends on the direction of incident light source and the reflection properties of the surface. In its most general form, it is often modeled in the following integral form [2,3]:

$$I(x,y,z) = \int_0^{2\pi} \int_0^{\pi/2} L(x,y,z,\theta,\phi) R(\theta,\phi) \sin\theta d\theta d\phi$$

where,

$(x,y,z)$  = a visible surface point

$L(x,y,z,\theta,\phi)$  = illumination reaching  $(x,y,z)$  from direction  $(\theta,\phi)$  in spherical coordinates with the surface normal as axis

$R(\theta,\phi)$  = the reflectance toward the viewer from the surface for light coming from the direction  $(\theta,\phi)$

$\sin\theta d\theta d\phi$  = is an element of solid angle.

The complexity of performing this integration can be simplified if you make the following assumptions [3]:

First assume that  $L$  is a delta function and that it is zero for all other

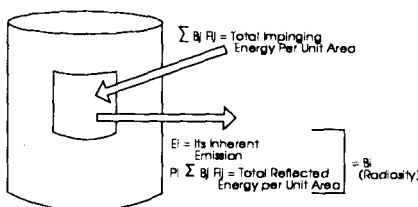


Figure 5—The best way to handle indirect illumination for local shading considerations is through radiosity. In the real world, a large percentage of the total light in an interior space maybe accounted for in this way.

directions except in the direction of "point" light sources. The integral can now be replaced by a simple sum over certain discrete directions.

Next, assume that all the directions that are not light source directions can be defined as from an ambient light source.  $L$  is therefore independent of the angle of incidence and it may be removed from the integral. The integral of  $R$  may then be replaced by an average, or ambient, reflectance.

And finally, assume that the reflectance function  $R$  is also a delta function. This defines the surface point as a mirror and it reflects light only from the mirror direction. This will of course cause sharp reflections.

Thus the reflected intensity can be defined as

$$I_{ref} = R(\theta_{in}, \phi_{in}) L_{in}$$

If we now include Lambert's Law, the reflective intensity becomes:

$$I_{ref} = \left( \frac{R \cos \phi_{in}}{\pi} \right) L_{in}$$

At certain viewing angles, a shiny surface can reflect all of the incident light, independent of the reflectivity values. As such, it produces a ray of reflected light that is the same color as the incident light. Since we normally use white light, the reflection will be a bright white spot. For an ideal reflec-

#### Point-Sampling Algorithms

##### Ray Tracing Algorithm

```
for all pixels *x,y){
  for all objects {
    compare z }
```

##### Z-Buffer Algorithm

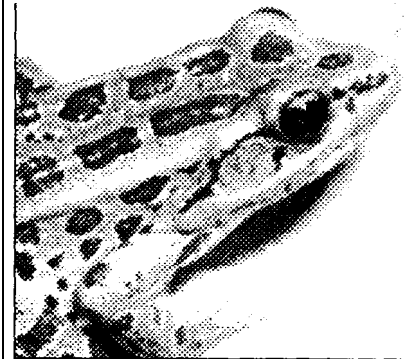
```
for all objects {
  for all covered pixels (x,y)
    compare z }
```

##### Painter's Algorithm

```
sort objects by z, for all objects {
  for all covered pixels (x,y)
    paint }
```

##### Scan Line Algorithm

```
sort objects by y, for all y {
  sort objects by x, for all x
    compare z }
```



## Develop powerful image applications quickly and easily

### Introducing VICTOR, the video capture and image processing library

VICTOR, the video capture and image processing library, is the latest product from the developers of ZIP Image Processing software. Victor is a library of functions for C programmers that simplifies development of scientific imaging, quality control, security, and image database software. Victor gives you complete control of your video frame grabber and also includes image processing, display, and TIFF/PCX file handling routines.

Application development is simplified because we've taken care of the details of device control and image processing. All the hard low level coding has been done -- and you can concentrate on your application.

Your software can have features such as: live video on VGA, pan and zoom, display multiple images with text and graphics, pixel editing and image processing. And, to get you up and running quickly, we've included our popular Zip Image Processing software for rapid testing and prototyping of image processing and display functions.

Victor supports Microsoft C and Quick C, and includes over 100 functions, demonstration and prototyping software, full documentation, and source code for device control routines. .. all for only \$195.

Victor and Zip support ImageWise and other popular video digitizers.

VICTOR LIBRARY \$195  
includes ZIP Image Processing

ZIP Image Processing \$79  
please specify digitizer

Video frame grabbers are also available.

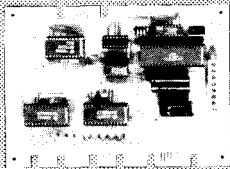
Call (314) 962-7833 to order  
VISA/MC/COD

CATENARY SYSTEMS  
470 BELLEVIEW  
ST LOUIS MO 63119  
(314) 962-7633

# RELAY INTERFACE

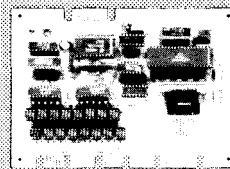
**AR-16 Relay Interface** ..... \$ 89.95  
 Provides software control for 16 external relays, expandable to 126 relays with EX-16 expansion cards Plug-in relay cards, power relays and latched relays with status available. May be connected directly to serial printer port or RS-232

**EX-16 Expansion Card** ..... \$ 59.95  
 (16 channel)



# ANALOG TO DIGITAL

**ADC-8 (8 channel)** ..... \$ 89.95  
**ADC-16 (16 channel)** ..... \$ 99.95  
 Input temperature, joystick movement, voltage, pressure, energy usage, light levels etc Voltage input level is adjustable (0 to 5 volts typical). Connects to RS-232 or RS-422 ports.



# STATUS INPUT

**STA-8 (8 channel)**\* ..... \$ 99.95  
**STA-18 (16 channel)**\* ..... \$ 119.95  
**STA-16-U (touch tone input)** . . \$ 159.95  
 Input on/off status of switches, relays, thermostats, security devices, smoke detectors, pressure switches and hundreds of other devices. Touch Tone input decodes all standard telephone touch tones and may be connected directly to a telephone line using an inexpensive coupler Connects to RS-232 or RS-422 ports.

\*Inputs are expandable to an additional 128 status inputs or 16 analog inputs. Add up to 112 relay outputs using EX-16 expansion cards.

- FULL TECHNICAL SUPPORT provided over the telephone by our staff. A detailed technical reference manual is provided with each order including software examples in Basic and Assembly Language.
- Engineered for continuous 24 hour industrial applications.
- Use with IBM and compatibles, Tandy, Apple and most other computers with RS-232 or RS-422 ports. All standard baud rates and protocols may be used (50 to 19,200 baud) default is 9,600 baud 8 data bits, 2 stop bits, no parity. Use our 800 number to order free information packet.  
 Technical Information (614) 464-4470.

24 HOUR ORDER LINE (800) 842-7714  
 Visa Mastercard American Express COD

ELECTRONIC ENERGY CONTROL, INC.  
 380 South Fifth Street, Suite 604  
 Columbus, Ohio 43215

tor, the angle of incidence and the angle of reflectance will be identical. If we let  $V$  be a vector that points in the direction of the viewer,  $R$  be the direction of the reflected specular light,  $N$  be the surface normal, and  $L$  be a vector in the direction of the light source, then specular reflection can be seen with a perfect reflector whenever  $V$  and  $R$  coincide ( $f=0$ ) [17].

Real objects exhibit specular reflection over a range of positions about the vector  $R$ . Shiny surfaces have a narrower reflection range and dull surfaces have a wider reflection range.

One method for modeling the light intensity leaving a surface is called the Phong model. It sets the intensity of the reflected light proportional to the cosine off (angle between viewer and reflected ray) to the  $n$ th power. Here, the value of  $n$  is chosen to specify the type of surface to be viewed. A very shiny surface is modeled with a large  $n$  ( $>200$ ) and smaller values on the order of unity define duller surfaces. For a perfect reflector,  $n$  approaches infinity. Since reflection also depends on the angle of incidence (the intensity increases as the angle of incidence increases), then a good approximate reflection model can be formulated as:

$$I = K_d I_a + \frac{I_p}{(d+d_0)} \left[ K_d (\overline{N \cdot L}) + K_s |V \cdot N| \cos^n \phi \right]$$

- here,
- $K_d$  = coefficient of reflection
  - $I_a$  = ambient light intensity
  - $I_p$  = source light intensity
  - $D$  = distance from the surface to the point source
  - $d+d_0$  = in the denominator accurately models the intensity reflections for surfaces at varying distances from a nearby light source
  - $N$  = unit normal vector from the surface
  - $L$  = a vector from the surface point to the light source
  - $V$  = a vector from the surface point to the point of view
  - $K_s$  = a surface-dependent constant
  - $n$  = the dull-to-shiny value

In this model, constant values are assigned to the parameters  $K_d, K_s$ , and  $d_0$  for each illuminated surface. Intensity values for the ambient light and the point sources are set. So for each point on an illuminated surface, you can calculate the relevant dot products and determine the intensity of the reflected light.

The best way to handle calculations of indirect illumination, for local shading considerations resulting from light scattering between objects, is radiosity. It was originally developed by engineers studying radiative heat transfer, then first applied to computer graphics by Goral et al. [16]. The primary motivation for investigating radiosity methods in the context of image synthesis was to understand and simulate the effects of diffuse interreflection between surfaces within an environment. This type of interaction has often been lumped, rather arbitrarily, into the ambient term of the local and global reflection models. In reality, it accounts for a fairly large percentage of the light energy contained in a typical interior space.

In order to pursue this further, let's make the following assumptions: We will assume that all objects within our image synthesis environment are Lambertian diffuse surfaces such that, at any given point, they exhibit an equal intensity in all directions. The energy arriving at a surface may arrive from any direction with any distribution, requiring simulation of diffuse interreflections using a point-sampling method such as ray tracing. In radiosity, we therefore solve for the intensity at discrete points (on objects) within the environment and not the intensity of pixels at the image plane. This "object space" nature of such an algorithm coupled with the diffuse assumptions leads to a "view-independent" result.

The theoretical foundations of radiosity rely heavily on conservation of energy. The radiosity of a surface area ( $B$ ) is defined as the energy (i.e., light) per unit area leaving the area per unit time. The total radiosity is therefore the sum of any energy ( $E$ )



emitted directly by the visible surface (i.e., a glowing object) and all reflections from that specified surface. This basic radiosity relationship has the form,

$$B_i dA_i = E_i dA_i + \rho_i \sum_j B_j F_{ji} dA_j$$

where,

$B_i$  = radiosity of surface area  $i$  (energy/unit area/unit time),

$E_i$  = emission of surface area  $i$  (energy/unit area/unit time),

$A_i$  = area of the surface piece  $i$ ,

$A_j$  = area of the surface piece  $j$ ,

$F_{ij}$  = form-factor from  $i$  to  $j$  (fraction of energy leaving the surface area  $j$  and arriving at surface area  $i$ )

$r_i$  = reflectivity of surface area  $i$ , and

$n$  = number of discrete surface areas.

A reciprocity relationship states that,  $F_{ij} A_j = F_{ji} A_i$ . So if we use this and then divide our form of the total radiosity by  $dA_i$ , we derive an expression for the total energy emitted by a surface area  $i$ ,

$$B_i = E_i + \rho_i \sum_j B_j F_{ji}$$

Here, the integral relationship has been replaced with a discrete summation by dividing the environment into small discrete surface areas for which a constant radiosity is assumed.

If we now consider the  $n$  surface areas that make up our synthesized image, we can construct a series of  $n$  simultaneous equations of the form,

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1-\rho_n F_{nn} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

which can be easily solved [18].

The form-factors discussed above require by far the largest computational effort within any radiosity algorithm. They determine the amount of direct energy exchange which can take

place between two discrete surface areas. Unfortunately, evaluating the form-factors involves solving a separate hidden-surface problem between each pair of surface areas within the environment. Mathematically, the form-factor is given by,

$$F_{ij} = \frac{1}{A_i} \iint \frac{(\cos \phi_i \cos \phi_j)}{\pi r^2} [\text{hidden}] dA_j dA_i$$

The actual value of *[hidden]* takes on 1 or 0 depending on whether  $dA_j$  can see  $dA_i$  or not, thus implicitly handling visibility and partial visibility between surface areas. There are several analytical and numerical techniques for solving the above integral. I refer you to references [19] and [20] for more details.

## ALIASING

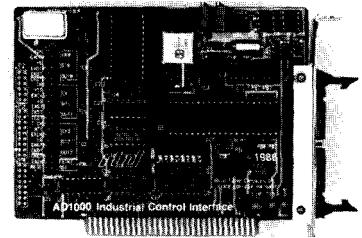
Aliasing is an artifact caused by the inability of the frequency components of a signal above the Nyquist Limit to be reconstructed from discrete samples [2,9]. Since the output of any image synthesis procedure is a raster image, it is therefore a discrete representation of a continuous image. As such, the amount of information contained within the image is determined by its sampling frequency and size. Attempts to incorporate higher frequencies (finer detail) into the image results in incorrect lower-frequency components not contained within the original image called "aliases." The most common examples of spatial aliasing within an image are the jagged stairstepping of slanted edges or the motion blur created when generating a sequence of images for animation.

It was always assumed that as display devices evolved, higher resolution would diminish the effects of aliasing. In reality, aliasing produces regular patterns that are easily noticed and must be correct for, especially in animation sequences [2,12]. Within this format it can cause [2]:

classical temporal-like the backwards spinning wagon wheel in the movies

# PC Bus Data Acquisition and Control Cards

STOP paying a small fortune for PC data acquisition and control in terfaces!



## AD1000 — \$295

Real Time Devices, Inc. designs and manufactures the lowest cost industrial/scientific interface cards for the PC/XT/AT bus. Our commitment is to offer only high quality U.S. designed and manufactured Interfaces — not cheap imports. All our cards are backed by a one year warranty, 30 day NO RISK return policy, and free technical support!

AD1000 — 8 channel 12-bit 20 uS/A/D; sample & hold; three 5 MHz timer/counters; 24 TTL digital I/O lines. \$295

AD200 — 4 channel 12-bit 125 uS/A/D; three 5 MHz timer/counter, resistor configurable gains, 24 digital I/O lines. \$239

AD500 — 8 channel 12-bit integrating A/D; programmable gains of 1, 10, & 100. Extremely stable & accurate! \$239

AD100 — Single channel version of AD500; optional D/A, gains of 1, 10, & 100. Plus 10 digital I/O lines \$149/\$198

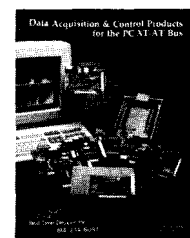
ADA300 — 8 channel H-hit 25 uS/A/D; single K-bit D/A; 24 TTL digital I/O lines. \$239

DA600 — Fast settling dual/quad 12-bit D/A; internal double buffering. \$179/\$239

DG24/96 — 24/48/72/96 line TTL compatible digital I/O cards, 10 MHz, 8255-based. Optional buffers and pull-ups. \$95/\$239

TC24 — Five 5 MHz timer/counters; uses powerful Am9513 chip Also 24 digital lines. \$189

## FREE CATALOG



CALL or WRITE today for your copy of the latest catalog and price list for our complete line of PC bus data acquisition, control, prototyping, extender cards, technical hooks and accessories! OEM discounts begin at a quantity of three!

Real Time Devices, Inc.

*rtid* 531 E. Marylyn Avenue  
P.O. Box 906  
State College, PA 16804 USA

814/234-8087

strobing-causing a fast-moving object to seem to jump in discrete steps  
 scintillation-small particles that drift between samples blinking on and off  
 "crawling ants effect"-the spatial aliasing "jaggies" slowly "moving" between frames  
 stretching and shrinking-a slowly moving small object (a few pixels in size) will seem to stretch and shrink in one-pixel steps as it crawls across the screen  
 beating-vertically moving objects beating with the interlace of broadcast TV

In order to reduce the effects of aliasing within an image, we must filter the image to remove or reduce the high-frequency components before sampling [2,13,14]. In effect, we will reconstruct an image "as near exactly as possible" from its samples, provided our input image is bandlimited to some maximum frequency which is less than one half the sampling rate; with this spatial sampling rate for our synthetic image being fixed by our final image resolution.

For this purpose, let's define the following entities:

$i(x,y)$ = an input image  
 $f(x,y)$ = a kernel or point spread function  
 $s(i,j)$ = sample antialiased filtered-sampled image, with  $i,j$  being integers

where, the most standard form of the "antialiasing integral" is defined as,

$$s(i,j) = \iint i(x+i,y+i)f(x,y)dx dy$$

and for temporal antialiasing,

$$s(i,j, \tau) = \iiint i(x+i,y+i,t+\tau)f(x,y,t)dx dy dt$$

All antialiasing algorithms solve or approximate this integral for each pixel contained in the synthesized image. Usually they vary only in the

type of numerical technique used and in the form of the point-spread function  $f(x,y)$ . My experience has shown that the best spread function is a low-pass filter with a smooth, all-positive, and nearly finite frequency response of the form,  $e^{-x^2}$ , the standard Gaussian. It easily removes all the high-frequency details without distorting the low-frequency regime. And, unlike other low-pass filters, it minimizes the effects of sharp transitions which cause noticeable ringing within the filtered image (the Gibbs phenomenon).

If you plan to incorporate antialiasing into your image synthesis, I suggest you read Joy's book [9], "Computer Graphics: Image Synthesis." He discusses many variations and applications of antialiasing backed by a good reference section. In general, however, I find myself extremely lazy. Although I have tried several antialiasing correction schemes in the past, I usually end up passing over this "required" step. I like to develop my images at a higher resolution than the final output. I then create the final output pixels by combining several pixels which overlap the output pixel using a weighted averaging scheme. It works quite well and provides good results.

## TEXTURING

Texture is the surface detail of a displayed object. It is dependent on the optical properties of the object material and is easily modeled by adjusting the intensity values furnished by a shading model. This is achieved by altering the surface normal so that it is a function of position over the surface. If the surface normal is allowed to randomly vary, an irregularly textured surface reminiscent of a raisin is obtained. Irregular surfaces can be generated by dividing each surface area into a collection of small randomly oriented surfaces. And, as in both cases, we could allow the coefficient of reflection to vary with position and thereby obtain quite a bit wider variation in overall intensity.

It is also possible to use what's called a texture-mapping method. This involves the application of stored patterns to surfaces of 3-D objects, much like applying patterns to 2-D objects. Each array pattern is treated as a plane surface with the position and orientation of the pattern specified relative to the object. The artist sets a pattern reference point and defines two vectors that specify the orientation of the array plane. These object-linked pattern planes are then stored and referenced by a hidden-surface utility. Intensity values stored in pattern arrays are used to modify or replace intensity values calculated in the shading model.

## MODELING

The creation and manipulation of the objects within our image environment according to some "defined" system representation is called "modeling." Models for a system can be graphical or purely descriptive, such as a set of equations that define the relationships between system parameters. This process includes the definition of shape, location, and orientation of each object, and the location, spectral wavelength band, angular distribution, and intensity of each light source.

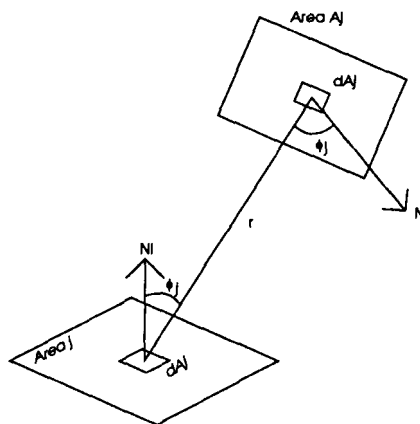


Figure 6—The amount of energy exchanged between two bodies is figured through a relatively complex set of simultaneous equations. Worse yet, evaluating the form factors requires solving a separate hidden-surface problem between each pair of surface areas within the environment.

Graphical models are often referred to as "geometric models," because the component parts of a system are represented with geometric entities such as lines, circles, polygons, spheres, cylinders, parametric patches, and implicit equations. Each is usually a tradeoff between the "extent of realism" of the more accurate model and the "expense" of implementation. However you want to define it, modeling is the architectural design and layout of your image synthesis code. It can be organized as a hierarchy of symbols to be used within a versatile system, or a structured code environment designed to implement specific features (which I tend to do). I leave that to you. In all honesty, when I work on an image, I very rarely think in terms of "modeling architecture." Instead I plug along in a fairly linear fashion. I set up a proposed environment of objects, the number of sources I want, and their orientations. Then I choose each type of synthesis technique I plan to apply and the order of

application. Off I go. Come to think of it, that's called modeling!

...AND ON TO THE REAL WORLD

One of the greatest long-term joys I have experienced in life is being directly involved in the development of some new discovery or new technique. Even if it's the smallest of contributions, I feel a sense of accomplishment. And I'm even making a living doing it! It almost makes me feel guilty, but not quite. I've enjoyed every minute of the last ten to fifteen years. I've been able to closely couple my work in physics to that of the world of computers through the application of computer graphics and image synthesis techniques (for real-world modeling of physics problems). So I get extra joy from being able to pass some of this "fun experience" on to you.

I've honestly had a difficult time with this article because there are so many options and possibilities within

image synthesis. Where does one begin? How do I end? I don't know how to truly answer these questions except to offer you a brief outline of some basic questions that must be considered whenever you decide to attempt to implement some of the techniques we discussed above [21]. It should give you a start at evaluating your image synthesis routine while reminding you of some of the fundamental problems. If the software you generate responds to these questions efficiently, market it! You'll make millions. Just remember to send me a percentage of the royalties.\*

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

IRS

- 207 Very Useful
- 208 Moderately Useful
- 209 Not Useful

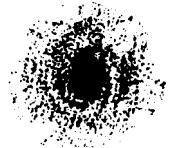
**MC68000/10/20 SIMULATORS**

**DEBUG 68000/10/20 PROGRAMS ON YOUR PC**

SIM68K and SIM20 are software simulations of the MC68000/10 and MC68020 CPUs, designed to run on the IBM PC and compatibles with DOS 2.0 or higher. No additional hardware required. They accept files generated by a 68000/10/20 assembler. They are complete debugging tools, providing access to registers, flags, and memory. All 68000/10/20 instructions, addressing modes and condition codes are supported.

- Load/Dump Facilities • Disassembler
- Single-Step & Fast Execution • Breakpoints
- Interrupts • Terminal I/O Support
- Symbolic Debugging • Execute Batch Files

SIM68K (MC68000/10).....\$285  
SIM20 (MC68020)..... \$345



**BIG BANG SOFTWARE, INC.**  
7151 W. Hwy. 98, Suite 286  
Panama City Beach, FL USA 32407  
Phone 904-784-7114 Fax 904-235-3475

**68000 for the PC-AT**

Put a complete 68000 computer into your PC

*An exciting 68000 development platform that gets your software out!*

- Actually runs a 68008—no simulations required to test your code.
- Real-time embedded multitasking applications can be developed using VRTX32 and RTscope.
- Familiar tools (editor, cross-assembler) can be used on the PC to write and document programs.
- Downloading to the system is fast: on-board monitor provides direct communication with the PC through the I/O channel.

PLUS you can develop multiprocessing techniques and programs by installing two or more boards in the PC.

Features of the MISTER-8

- Low cost: operates in any PC, AT, or clone.
- Two or more boards can be installed to do multiprocessing.
- Communicate using the PC's serial port and your favorite modem program — or directly through the FIFO.
- 126K ROM in three application sockets; 64K SRAM.

MISTER-8 with monitor EPROM ..... \$495



717-524-7390 or

60 SOUTH EIGHTH STREET, LEWISBURG, PA 17837

717-523-0777

## AN EXAMPLE

To help you get started, I am including here a few modular **procedures** which will give you a flavor of how to write your own image synthesis code. These are derived from a code which features:

- standard reflection, transmission, and shadows
- extensible primitives
- CSG—constructive solid geometry with an intersection routine
- a return list of all intersection points
- antialiasing

Of course I have taken an exceptional liberty here. I have not discussed CSG modeling within the text of this tutorial, due to space constraints. So please forgive me. Instead, I refer you to Chapter 4 of reference [9], a paper written by Atherton on the subject; and reference [22], a paper called "Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System" by Goldfeather et al. If I admit to doing this by design, don't hold it against me. I know that the following code is "somewhat" confusing, but I did want to provide you with more than words and theories. First, though, I want to cover the basics, so just a few words on CSG before you dig up these references and have at it.

CSG uses a set of 3-D primitives, such as blocks, pyramids, cylinders, cones, and spheres, to create object definitions of composite shapes. These 3-D constructions can be accomplished by the user by choosing a set of solids and a corresponding set of operations to be performed. For example, the union operation can be demonstrated by positioning a pyramid on top of a box and creating a house-like object. An intersection operation can be exemplified by the overlap volume of two primitive objects. And, of course, the difference operator represents the subtraction of one volume from another. Using a superposition of these three operators, one can create a multitude of shapes. The organization of these composite shapes is kept within a tree structure. It is designed so that each node corresponds to a region of 3-D space. Each of these regions is divided into octants and allows for the storage of eight data elements at each node. Individual elements are called voxels [23].

## The Code Development

To create such a code, you should first read in an object model and create your data structures. Then generate a set of primary rays according to your viewpoint and display and **TRACE** each primary ray through the scene, generating and tracing secondary shadow rays as necessary. At appropriate points, make your calculations for shading, shadowing, and the ultimate display color. Of course, the heart of any such system is the ray **TRACER**, the **SHADER**, the **SHADOWER**, and **INTERSECTION** routines. The following are examples of such. All code examples are written in C. Some programming steps are described and left to the reader for proper implementation [15].

```

#define define:
typedef double Flt;          /* floating-point data type */
typedef Flt Vec[3];         /* a 3-vector */
typedef Vec Point;         /* xyz point data type */
typedef Vec Color;        /* rgb color data type */

typedef struct Ray {
    etc.
};

typedef struct Prim {
    etc.
};

typedef struct Comp {
    etc.
};

typedef struct Surf {
    /* a surface type
    with diffuse reflection coeff. = kdiff
    specular reflection coeff. = kspec
    transmission coeff. = ktran
    surface + body color = color
    index of refraction = refrindex
    etc.
};

typedef struct Isect { /* an intersection point
    etc.
};

with
    - point position (origin)
    - unitized direction of P
    - normal to a surface point
    - line parameter at intersection (as in P+tD)

col
    - color of light returning along ray

lit
    - is the intersection list

I
    - is the incident ray

level
    - the depth parameter

weight
    - is the cumulative weight for a ray's color in
    final pixel value

modelroot - the root of the CSG tree

PROCEDURES:

/* trace a ray through the scene and return a color */
/* find first intersection for t>0 and shade it */
trace(level, weight, ray, col)

int level;
Flt weight;
Ray *ray;
Color col;
{
    Prim *prim;
    Point P, N;
    Isect hit [ISECTMAX];

    I /*intersect ray with everything in scene */
    if (Intersect (ray, modelroot, hit)) {
        /*find prim, point P, 6 normal N at 1st intersection */
        prim = hit[0].prim;
        RayPoint (ray, hit [0].t, P);
        (*prim->procs->normal) (&hit [0], P, N);
        if (VecDot (ray->D, N) > 0.) /*flip norm if necessary */
            VecNegate (N, N);
        /* shade that surface point */
        Shade (level, weight, P, N, ray->D, hit, col);
    }
    else {
        /* if no intersections return background color */
        ShadeBackground (ray, col);
    }
}

Shade (level, weight, P, N, I, hit, col)
int level;
Flt weight;
Point P, N, I;
Isect *hit;
Color col;
{
    Ray tray;
    Color tcol;
    Surf *surf;

    /* compute diffuse */
    col=0
    for all lights
        L = direction vector from P to light
        if N.L>0 and
            Shadow (ray from P toward light, distance to light)>0
            then col+= (N.L)*lightcol

    /* if we're not too deep then recurse • /
    if (level+1<maxlevel) {
        VecCopy (P, tray.P); /* start point for new rays */

        surf = hit [0].prim->surf;
        /*recurse on specular reflection ray if significant */
        if (surf->kspec*weight > minweight) {
            VecAdds (-2.*VecDot (I, N), N, I, tray.D);
            Trace (level+1, surf->kspec*weight, &tray, tcol);
            VecAdds (surf->kspec, tcol, col, col);
        }
    }
}

```

```

/*recurse on transmission ray if significant • /
if (surf->ktran*weight > minweight){
/* hit[0].medium and hit[1].medium are
   exiting and entering media */
if (TransmissionDirection(hit[0].medium,
hit[1].medium, I, N, tray.D)){
Trace(level+1, surf->ktran*weight, &tray, tcol);
VecAdd(surf->ktran, tcol, col, col);
}
}
}
/* determine fraction of unblocked light in ray direction
for a light at t=tmax, for penumbras, this routine
would return a fraction */

Flt shadow(ray, tmax)
Ray *ray;
Flt tmax;

int nhit;
Isect hit[ISECTMAX];

nhit = Intersect(ray, modelroot, hit);
if (nhit==0 || hit[0].t > tmax-rayeps) return 1.;
else return 0.;
}

TransmissionDirection(m1, m2, I, N, T)
Surf *m1, *m2;
Point I, N, T;

Flt n1, n2, eta, c1, cs2;

n1 = m1 ? m1->refrindex : 1.;
n2 = m2 ? m2->refrindex : 1.;
eta = n1/n2; /* relative index of refraction • /

c1 = -VecDot(I, N); /* cos theta 1 */

```

```

cs2 = 1.-eta*eta*(1.-c1*c1); /* cos squared theta 2 */
if (cs2<0.) return 0; /* total internal reflection • /
VecComb(eta, I, eta*c1-sqrt(cs2), N, T);
return 1;
}

/* intersect a ray with the solid, which can be either
composite or primitive. Put a sorted list of
intersections in hit and return the number of
intersections. • /

Intersect(ray, solid, hit)
Ray *ray;
Comp *solid;
Isect *hit;

int nl, nr; /* #intersections */
Isect lhit[ISECTMAX], rhit[ISECTMAX], /* lists • /
if (solid->compflag){/* composite solid */
/*recurse on left • /
nl = Intersect(ray, solid->left, lhit);
if (nl==0 && solid->op!='|'){/* '|' = union • /
/* optimization: if l is nul then l&r and l-r are
null, so skip r */
return 0;
}
} else {
/* recurse on right */
nr = Intersect(ray, solid->right, rhit);

/* merge left and right lists */
return IntersectMerge(solid->op, nl, lhit, nr, rhit, hit);
}
} else /* primitive solid • /
return (*(Prim *)solid->procs->intersect) (ray,
(Prim *)solid, hit);
}

```

The above code supports CGS, so the model will consist of a binary tree of solids with each solid being a composite (*Comp*) or primitive (*Prim*). These composite solids form the inner nodes of the CGS tree and primitives form the leaves. To allow the tree to be built out of a mixture of these nodes, we start the two structures the same and include a composite/primitive flag at the beginning. Routines can check the flag node to determine what type it is and to cast a *Prim* pointer to a *Comp* point or vice versa if necessary. The composite solid structure is simple, consisting merely of an operation code equaling intersection, union, or difference and two pointers to its two subsolids, which can be either composite or primitive.

This CSG technique (a combination of scan-line and z-buffer visibility codes) is widely used in the CAD area. It forms a z-buffer depth list at each span endpoint (the scan-line algorithm line segment between two consecutive endpoints). The ordered elements of this list are then evaluated versus the CSG tree to determine the visible element. If the visible elements calculated for two consecutive span endpoints agree, the visible element is thought to be visible throughout the span. If not, the span is divided and a new z-buffer list is generated at the midpoint of the interval (the left and right in our code) and the process is repeated on each subinterval (see reference [9]).

## REFERENCES

- [1] Whitted, T., "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, 23, 6 (June 1980), 343-349.
- [2] Joy, K.I., C.W. Grant, and N.L. Max, "SIGGRAPH '88 Tutorial #9: Image Synthesis," Proceedings of SIGGRAPH '88, Anaheim, California.
- [3] Cook, Robert L., T. Porter, and L. Carpenter, "Distributed Ray Tracing," *Computer Graphics*, 18, 3 (July 1984), 137-145.
- [4] Dippe, M.A.Z. and E.H. Wold, "Antialiasing Through Stochastic Sampling," *Computer Graphics*, 19, 3 (July 1985), 69-78.
- [5] Lee, M.E., R.A. Rednerand, and S.P. Useton, "Statistically Optimized Sampling for Distributed Ray Tracing," *Computer Graphics*, 19, 3 (July 1985), 61-67.
- [6] Blinn, J.F., "A Generalization of Algebraic Surface Drawing," *ACM Transactions on Graphics*, 1, 3 (July 1982), 235-256.
- [7] Hanrahm, P., "Ray Tracing Algebraic Surfaces," *Computer Graphics*, 17, 3 (July 1983), 83-90.
- [8] Roth, S.D., "Ray Casting for Modeling Solids," *Computer Graphics and Image Processing*, 18, 2 (February 1982), 109-144.
- [9] Joy, K.I., C.W. Grant, N.L. Max, and L. Hatfield, "Computer Graphics: Image Synthesis," Computer Society Press, Washington D.C., August 1988.
- [10] Williams, L., "Casting Curved Shadows on Curved Surfaces," *Computer Graphics*, 12, 3 (August 1978), 270-274.
- [11] Reeves, W.T., D.H. Salesin and R.L. Cook, "Rendering Antialiased Shadows with Depth Maps," *Computer Graphics*, 21, 4 (July 1987), 283-291.
- [12] Szabo, N.S., "Digital Image Anomalies: Static and Dynamic," *Computer Image Generation*, Bruce J. Schachter, Ed., John Wiley & Sons, NY, 1983, 125-135.
- [13] Oppenheim, A.V. and R.W. Shafer, "Digital Signal Processing," Prentice-Hall, Englewood Cliffs, N.J., 1975.
- [14] Slepian, D., "On Bandwidth," Proceedings of the IEEE, 64, 1972, 292-300.
- [15] Heckbert, P.S., "Writing a Ray Tracer," Introduction to Ray Tracing Tutorial Notes, SIGGRAPH '87, July 1987.
- [16] Gora, C.M., K.E. Torrance, and D.P. Greenberg, "Modeling the Interaction of Light Between Diffuse Surfaces," *Computer Graphics*, 18, 3 (July 1984), 213-222.
- [17] Hearn, D. and M.P. Baker, "Computer Graphics," Prentice-Hall, Englewood Cliffs, N.J., 1986.
- [18] Cohen, M.F., "A Consumer's and Developer's Guide to Radiosity," Proceedings of SIGGRAPH '88, Atlanta, Georgia.
- [19] Cohen, M.F. and D.P. Greenberg, "A Radiosity Solution for Complex Environments," ACM SIGGRAPH '85 Conference Proceedings, July 1985, 31-40.
- [20] Nishita, T. and E. Nakamae, "Continuous Tone Representation of 3-D Objects Taking Account of Shadows and Interreflections," ACM SIGGRAPH '85 Conference Proceedings, July 1985, 2330.
- [21] Haines, E., "A Consumer's Guide to Ray Tracing," Proceedings of SIGGRAPH '88, Atlanta, Georgia.
- [22] Goldfeather, J., J.P.M. Hultquist, and H. Fucks, "Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System," *Computer Graphics*, 20, 4 (August 1986), 107-116.
- [23] Casale, M.S. and E.L. Stanton, "An Overview of Analytic Solid Modeling," IEEE Computer Graphics and Applications, 5(2), February 1985, 45-56.

# An Intel 80386SX-based PC/AT-Compatible Motherboard

## Part 2

*Timing is almost everything*

by Daryl Rinaldi

In part 1 of this article, we looked at the basic logic of an 80386SX ISA Bus computer. I also described some important components and PC/AT system characteristics. To refresh your memory of how the components relate to one another, look at the first two figures for this article. Figure 1 shows the bus structure of the standard 80286-based AT computer; Figure 2 is a diagram of the 80386SX computer described in this article. Now that we know what the key devices are, let's look at how they work together in this design.

One of the best ways to study the workings of a system is to look at the bus cycles. We will look at eight bus cycle types: RESET, Local Memory cycles, Numeric Coprocessor cycles, Expansion Bus Cycles, X Bus cycles, ROM cycles, HOLD/HLDA cycles, and INTR/INTA cycles.

### RESET

There are two types of reset in a PC/AT: CPU reset and system reset. Depending upon certain system conditions, the 82230 will initiate one of the two types of reset. The 82335 interacts with the major system components to ensure these resets produce predictable results.

A system reset occurs on power-up to force the system into its initial state. When the system is first powered up, the 82230 activates RESET and RESCPU. Once PGOOD\ (power good) becomes active, the 82230 will deactivate RESET and RESCPU. Both the 82230 and 82231 reset themselves internally upon activation of RESET.

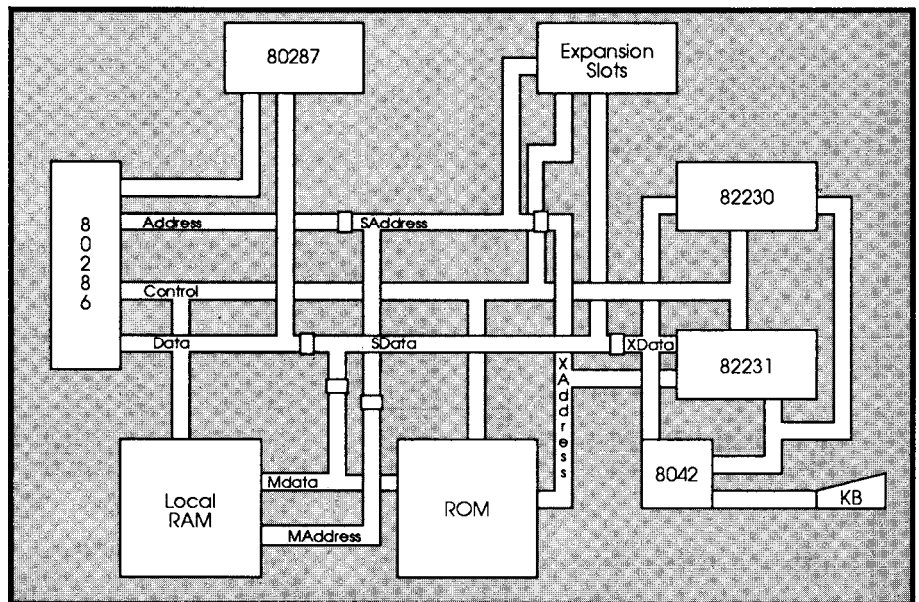


Figure 1 -A standard 80286-based AT-compatible has several separate buses for passing addresses and data through the system.

Upon receiving RESET and RESCPU, the 82335 activates RESETSX to the processor and RESETNPX to the numeric coprocessor. These devices go through their respective initialization routines. The 82335 also resets itself. In addition, the 82335's REFRESH counter is reset and all the internal registers go to their initial states.

Clock synchronization also occurs during power-up. Most timing in the system is derived from one of four clocks: CLK2, PCLK\, PROCCLK, or SYSCLK. CLK2 is a 33-MHz output of the 82335, generated from its 32-MHz EFI input, and is an input to the 386SX and the 387SX. PCLK\ is a 16-MHz output of the 82335. It is half the frequency of EFI and feeds the 82230's X3 input. The 82230 inverts X3 and

buffers it to generate the 16-MHz PROCCLK output. The 82230 also divides PROCCLK by two to generate the 8-MHz SYSCLK output.

All four of these clocks must be synchronized for the system to operate. This important process may not be immediately obvious from the component data sheets and could use some explanation.

The 82335 synchronizes PCLK\ with CLK2 after the falling edge of SYSRESET. This means that PCLK\ is low in phase 2 and high in phase 1 of CLK2 and that PCLK\ edges correspond to rising CLK2 edges. The synchronization sequence is shown in Figure 3.

After the deactivation of RESETSX, the second rising edge of CLK2 is the start of a phase 2 clock cycle.



# étude™

**25 MHz EI-bit**  
ANALOG-TO-DIGITAL CONVERTER

Based on the TRW THC1068-1 hybrid flash converter, its high signal-to-noise ratio yields excellent accuracy at the Nyquist limit.

- 4 KB of cache SRAM or to host as converted at DMA speed
- I/O or DMA data transfer
- 10-MHz full-power bandwidth
- 3.92 mV resolution
- 16 jumper selectable base addresses
- External clock and trigger inputs TTL compatible
- Software source code included

**\$495<sup>00</sup>**

ALSO AVAILABLE AS A KIT FOR \$99, INCLUDING:

- Printed Circuit board
- Software
- Manual & assembly instructions

Requires: PC compatible 1/2 length 8-bit expansion slot DOS 2.11 or greater EGA, VGA or Hercules display needed for graphic representation of data.



**Silicon Alley Inc.**

P.O. BOX 59593  
RENTON, WA 98058  
206.255.7410

©1989 Silicon Alley Inc. étude is a trademark of Silicon Alley Inc. Other brand or product names are trademarks or registered trademarks of their respective holders. Prices and specifications subject to change.

Reader Service #149

This edge corresponds to a falling edge of PCLK\ ensuring that PCLK\ is low in phase 2 and high in phase 1. Deactivating RESETSX and RESETNPX at the same time also ensures that the coprocessor is in phase with the CPU.

A SYSRESET also synchronizes SYSCLK to PROCCLK. From the start of a SYSRESET until the first PC/AT bus cycle, SYSCLK remains low. S1\ goes low during phase 1 for the initial code fetch. SYSCLK goes high at the beginning of the next phase 1 and continues oscillating at half the PROCCLK frequency. PROCCLK is derived from X3 which is connected to the 82335's PCLK\ output. The 82335 controls the PROCCLK/SYSCLK synchronization through this process.

CPU reset is much less dramatic. The 82230 activates RESCPU due to either a keyboard-controller-initiated reset or a CPU shutdown condition.

A keyboard-controller-initiated reset is generated via a low pulse on the 82230's RC input which is an output of the 8042 Keyboard Controller. The pulse is caused by writing 1111 xxx0 to I/O location 64H, programming the 8042 to produce an RC pulse of approximately 6 μs. This in turn causes a RESCPU pulse of approximately the same length-4 μs.

CPU shutdown signifies that the CPU has experienced an unrecoverable error. A CPU shutdown is indicated by M/IO286\ high, combined with SO\, S1\, and A1 low. The 82230 decodes the shutdown and generates

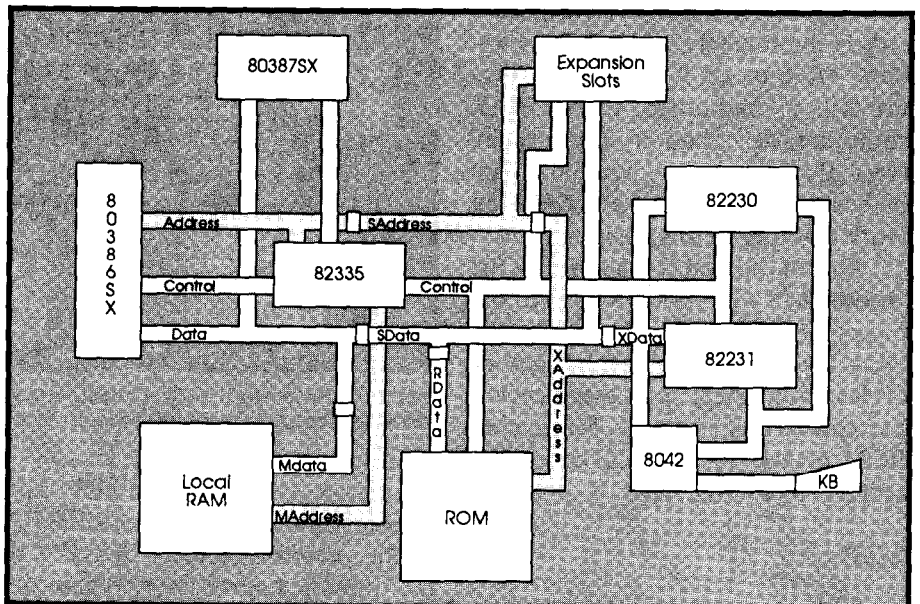


Figure 2—An 80386SX-based machine has a layout very similar to that of the 286-based machine.

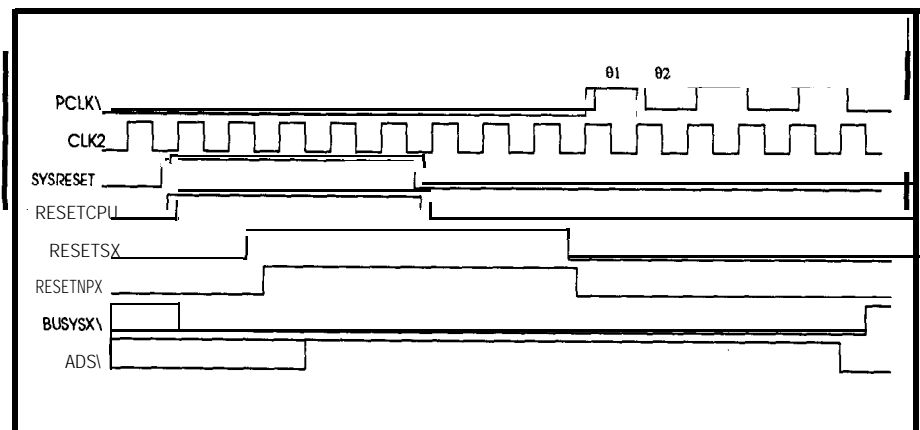


Figure 3—During a system reset, the 823.35 synchronizes PCLK\ with CLK2 on the falling edge of SYSRESET.

a RESCPU pulse of approximately 1.5  $\mu$ s.

The 82335 passes RESCPU on to the processor as RESETS $\bar{X}$  but does not reset itself internally nor does it activate RESETNPX. All 82335 register information remains valid and the 82335 REFRESH counter is not reset. Only the CPU initializes itself. All clocks continue running normally and no clock synchronization takes place. The 82335 deactivates RESCPU at the appropriate time to ensure that PCLK $\bar{N}$  stays in phase with CLK2.

The 80286 must be reset in order to switch from protected to real mode. The 386SX does not require a reset to go from protected to real mode, but 286-based software sometimes resets the CPU in order to accomplish the switch.

## LOCAL

The most frequent bus cycle generated by most applications is local memory access. The 82335 uses paging to allow most local memory accesses to be done without any wait states.

The 82335 can operate in Turbo or Non-Turbo mode. Non-Turbo mode accesses always takes six 16-MHz 386SX T-states, equivalent to three 8-MHz PC/AT T-states. The equivalence is important for software with timing loops based on I-wait-state PC/AT bus cycles. Local memory accesses in an IBM PC/AT use one (286) wait state.

Turbo mode accesses don't have the timing constraints of Non-Turbo mode accesses. Therefore, Turbo mode accesses are faster, though their speed varies depending on DRAM speed, page hits/misses, and bank hits/misses. The TURBO $\bar{N}$  input switches the 82335 into and out of Turbo mode.

A O-wait-state (Turbo mode) pipelined local memory read is shown in Figure 4. Address pipelining simply means that the address and bus cycle definition information for the next bus cycle becomes active before the current bus cycle is over.

This memory access starts with the assertion of the CPU's NA $\bar{N}$  signal.

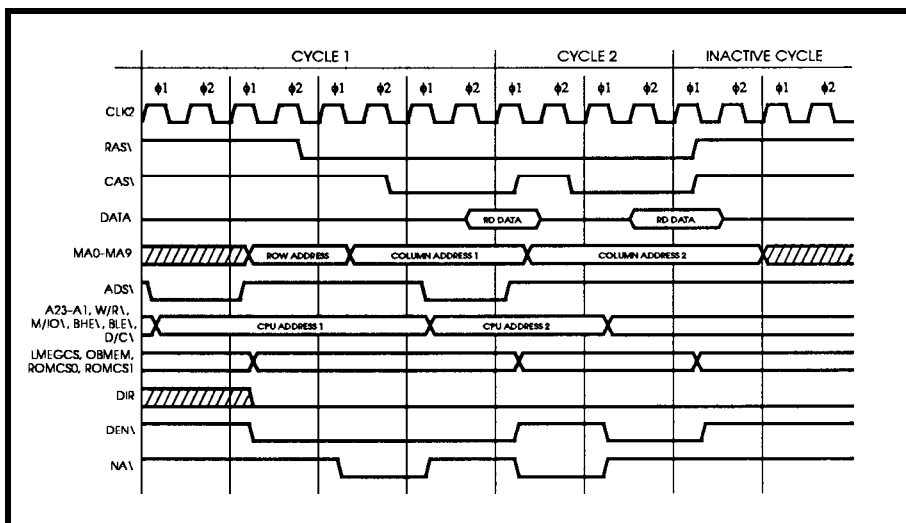


Figure 4—Address pipelining makes the timing for local memory reads more complicated.

NA $\bar{N}$  asserted causes the next address to come out in the next bus state, or in other words, initiates address pipelining. (A note about address pipelining: Although address pipelining cannot improve upon O-wait-state performance, it will provide more address setup time for the local DRAMs. What this means is that with address pipelining, you may be able to use slower DRAMs and still get the same performance as you would using faster DRAMs without the address pipelining.)

As the next address is output, so is the next bus cycle definition. ADS $\bar{N}$  goes low to signal that the outputs are valid. The 82335 decodes the incoming address and bus cycle definition. OBMEM is then activated to signal an on-board memory access. In an on-board memory access, the 82335 will not activate the status outputs-SO $\bar{N}$  and S1 $\bar{N}$ —to the 82230/82231. This basically disables the 82230/82231.

Since this is a read cycle, DIR will be low, causing data to flow from the MD bus to the local data bus.

Since this is a page hit, the 82335 outputs only the column address on MA0-MA9 and activates CASH0 $\bar{N}$  (high-byte bank) and CASL0 $\bar{N}$  (low-byte bank). If this were not a page hit, then the 82335 would have output the row address and RAS before outputting the column address and CAS.

After outputting CAS, the 82335 will activate DEN $\bar{N}$  to enable the MD-to-Local Data Bus buffers. The 82335 will then output READY $\bar{S}\bar{X}$  to the CPU to indicate it has valid data on the local data bus. The CPU will latch this data and end this bus cycle.

A local memory write is basically the same as a local memory read. The only difference is that DIR will be high, WE $\bar{N}$  will be active (low), and, of course, READY $\bar{S}\bar{X}$  will signal to the processor that the DRAM is ready to input data and not output data.

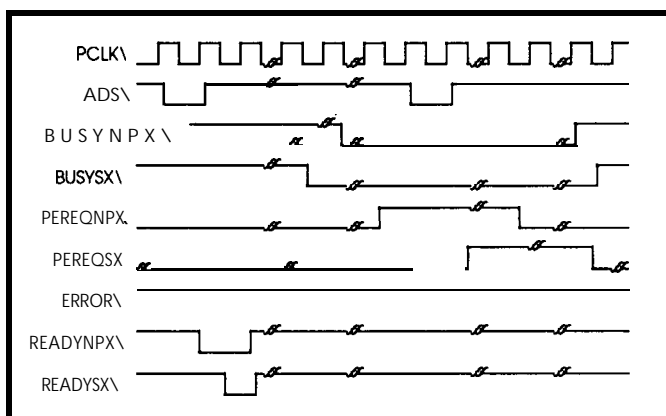


Figure 5—A typical NPX bus cycle uses four key signals: READY $\bar{N}$ , BUSY $\bar{N}$ , PEREQ $\bar{N}$ , and ERROR $\bar{N}$ .

## NUMERICS

When the 386SX does I/O accesses with A23 high (beyond I/O space), it signifies numeric coprocessor cycles. Because it uses a 32-bit bus, the 386DX uses A31 high for numeric coprocessor cycles. Other than the difference caused by bus size, the 386DX/387DX interface is the same as the 386SX/387SX interface. The bus size difference is transparent to software. Because A23 is high, the 82335 recognizes that these are numeric coprocessor cycles, so it starts the CPU-to-coprocessor interface instead of starting an I/O cycle.

A typical numeric coprocessor bus cycle is shown in Figure 5. The important signals are READY\, BUSY\, PEREQ, and ERROR\. READY\ signals the CPU that the coprocessor is ready to end the bus cycle. BUSY\ signals the CPU that the coprocessor is currently executing an instruction. PEREQ signals the CPU that the coprocessor either requires data or has data to output. ERROR\ signifies that a coprocessor error has occurred

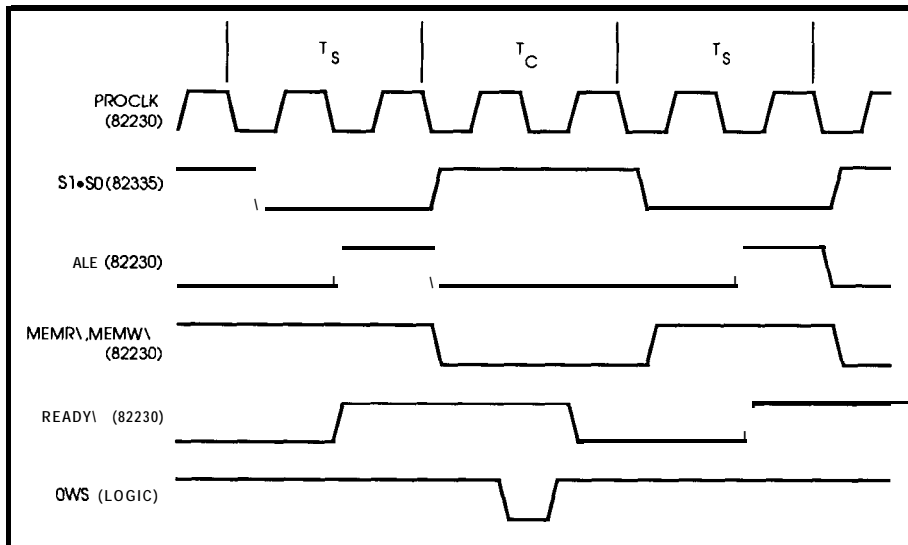


Figure 6-Devices on the expansion bus can override the default number of wait states via the use of two signals: OWS and IOCHRDY.

## EXPANSION BUS

Accesses to add-in cards, such as mouse cards, disk controllers, VGA cards, and so on, that reside on the PC/AT expansion bus require Expansion Bus Cycles.

The 82335 serves as a signal translator when the CPU accesses the PC/

AT part of the computer. This signal translation combined with the slower speed of the expansion bus results in PC/AT accesses taking longer than local memory cycles.

When calculating 386SX T-states required for expansion bus accesses, keep in mind that one B-MHz PC/AT T-state equals two 16-MHz 386SX T-states. Remember that signal translation adds up to three extra wait states, so a O-wait-state PC/AT access can take seven 386SX T-states (five wait states).

In a PC/AT system, the 286 combines with the 82288 Bus Controller (in the 82230) to control bus cycles. The 286 uses status outputs (S0\ and S1\ ) and control lines (M/I0\ and INTA\ ) to start and define a bus cycle. The 82288 uses these signals to generate Memory Read/Write (MEMR\, MEMW\), I/O Read/Write (IOR\, IOW\), and Address Latch Enable (ALE). The 386SX on the other hand, uses M/I0\, D/C\, and W/R\ to define a bus cycle and ADS\ to start it.

When the 82335 recognizes either an I/O access or an off-board memory access, it will output S0 and S1. This starts the expansion bus cycle.

The 82230 outputs ALE which causes A0-A19, M/IO286\, and BHE\ to be latched onto the System bus. ALE is buffered to produce BALE. One of the command signals (MEMR\, MEMW\, IOR\, IOW\ ) will then become active. After the default one

# The DA/M™

## A Low Cost Data Acquisition System

- 8-A/D Channels, B-Digital I/O Channels, 1-Counter/Timer.
- Runs on 12 to 24 VDC.
- 15 Systems or 255 points per RS232/RS485 Communications Port.
- Connect to Sensors that output 0-4.59V, 0-5V, 0-10V or 4-20Ma or add optional onboard amplifiers for lower signal levels.

DA/M 100-0	DA/M System	<b>\$200.00</b>
DA/M 100-1	RS232-RS485 Converter Cable	<b>\$20.00</b>
DA/M 100-2	ROM/RAM Piggy Back Card	\$50.00
DA/M 100-3	RS232-RS485 Converter Unit	\$100.00
DA/M 100-5	Screw Terminal Prototype Card	<b>\$100.00</b>
DA/M 100-7	Isolator/Relay Card	\$160.00
DA/M 100-B	Opto-22 Interface Card	\$140.00
DA/M 100-9	RAM/ROM/Real Time Clock Card	<b>\$180.00</b>

**DA/M**  
DATA ACQUISITION AND MANAGEMENT CORPORATION LTD.  
 #140 17303 102 Avenue  
 Edmonton Alberta Canada T5S1J8

Phone 1-403-486-3534

Fax 1-403-486-3535

8-MHz PC/AT wait state for 16-bit cycles or four 8-MHz PC/AT wait states for 8-bit cycles, the 82230 will activate READY286\.

The expansion bus device can override the default number of wait states via two signals: OWS and IOCHRDY. OWS can be used to shorten the bus cycle. If, during a PC/AT bus cycle, OWS is sampled low on the falling edge of PROCCLK, then the 82230 will activate READY286\ after that falling edge. OWS is only sampled on the falling PROCCLK edge that occurs in the middle of the second 286 T-state. The timing is shown in Figure 6. IOCHRDY can extend the bus cycle. By pulling IOCHRDY low, the accessed device can indicate that it is not ready. IOCHRDY will be sampled on the rising edge of SYSCLK. READY286\ will be inhibited until IOCHRDY goes back high.

OWS and IOCHRDY override the default number of wait states. There are also signals that can be used to define the current bus cycle type. These don't override the default number of wait states, but they do ensure that the correct default number of wait states are used.

IOCS16\ signifies a 16-bit expansion bus I/O access. MEMCS16\ indicates a 16-bit expansion bus memory access. MEMCS16\ is inverted to produce F16 and FSYS16 inputs to the 82230. These cause the cycle to default to one wait state. If both IOCS16\ and MEMCS16\ are inactive, then the 82230 assumes an 8-bit cycle and defaults to four wait states.

On the next falling edge of PROCCLK after READY286\ is asserted, the 82230 deactivates the command signal. The rising edge (plus some setup and hold time) is when data must be valid on the bus. At this point data will be accepted by the expansion bus device for a write, or by the system data buffers for a read. Note the 74F646s, U3 and U7, that connect the local data bus to the System Data bus. XDTR\ from the Control PAL controls the direction of these buffers. When XDTR\ is high, the local data bus drives the System Data

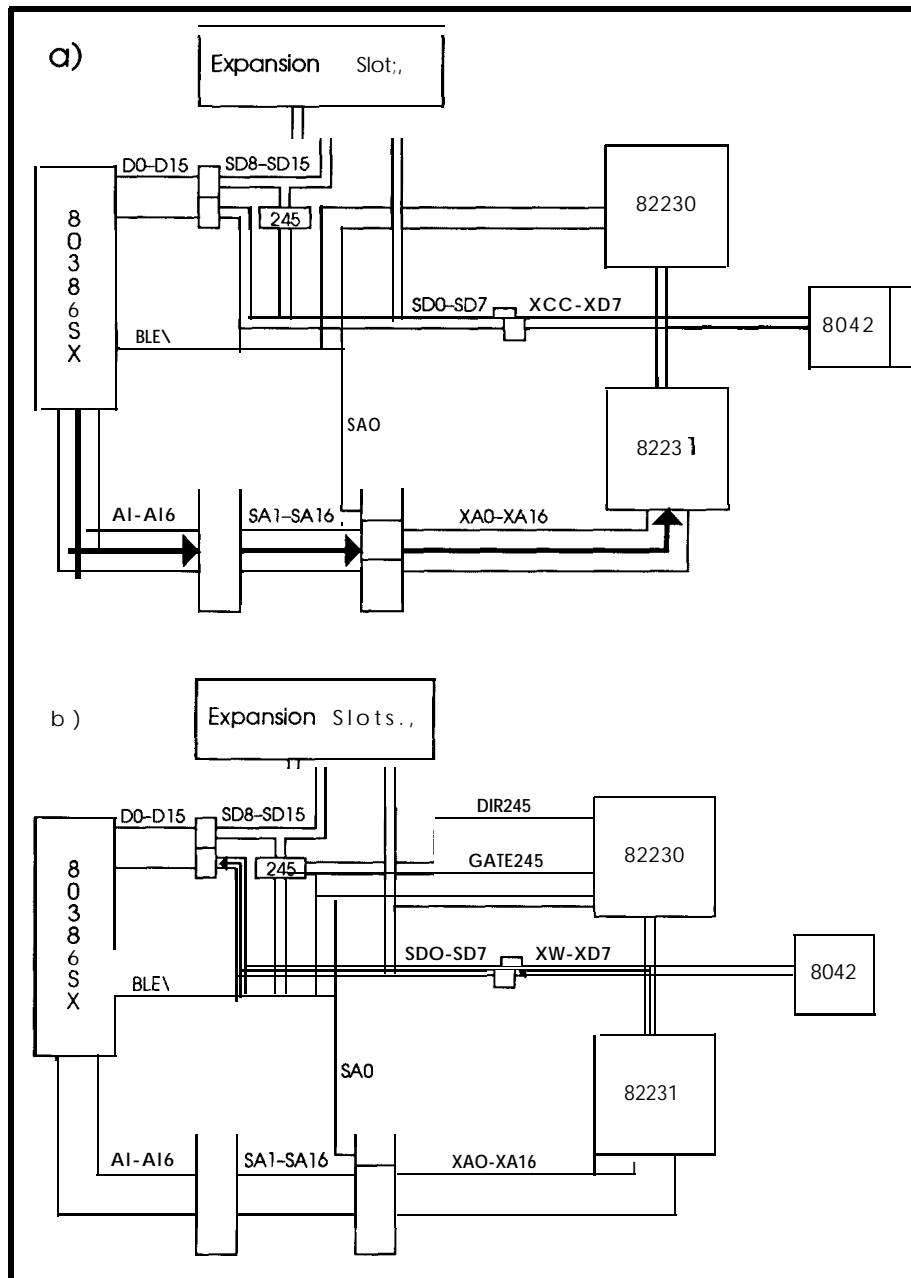


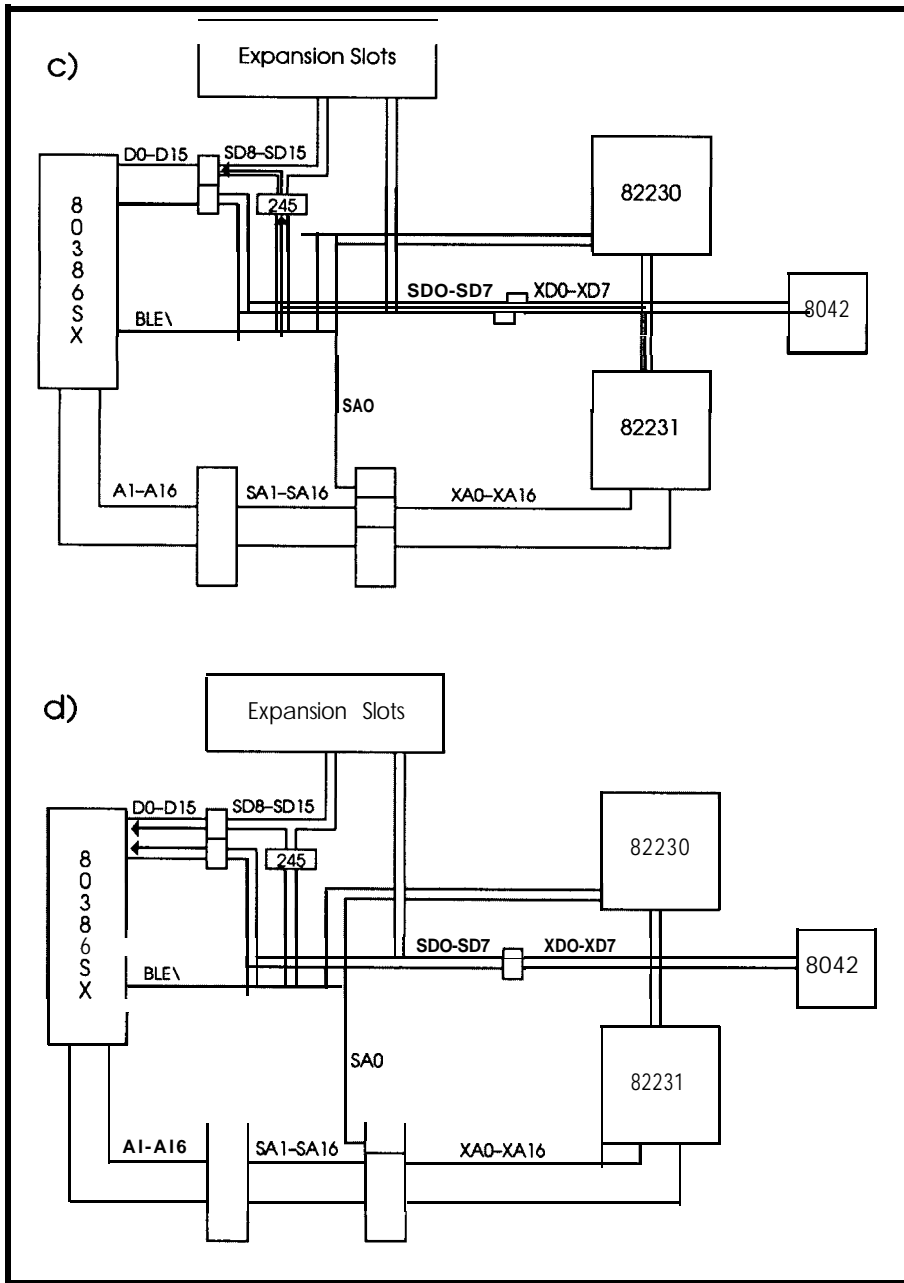
Figure 7-A number of steps take place when a 16-bit read is performed on an 8-bit device: a) the CPU starts the 16-bit read from the 8-bit device. b) the 82230 does an 8-bit read with SA0=0 and GATE245 inactive. SDO-SD7 are latched into the low-byte data

bus, and when XDTR\ is low, the System Data bus drives the local data bus. XMSDOE\ and XLSDOE\ control the output enables. LATCHH and LATCHL control the latching of the high and low bytes, respectively.

Due to the delay from READY286\ to READYSX\, by the time the CPU receives READY, the data from the expansion bus may no longer be valid. For this reason, data is latched into the 74F646s on every PC/AT cycle.

For 16-bit transfers, the data is latched on the falling edge of PROCCLK (PCLK\ while MEMR\ or IOR\ and MSDEN\ or LSDEN\ are still active. For an 8-bit transfer, EOC2\ (READY286\ conditioned by SYSCLK) with MSDEN\ or LSDEN\ active is used to latch the data.

The 286 can read and write data 16 bits at a time, however the PC/AT on-board peripherals are only 8 bits wide. Add-in cards can be either 8 or 16 bits wide. Software doesn't have to know



buffer. c) The 82230 does a second 8-bit read. SA=1, DIR245=1, and GATE245 is active. SD8-SD15 are latched into the upper-byte data buffer. d) The 82230 activates READY to end the cycle. The 386SX reads the DO-D15 latched data.

whether a particular device (or address) is 16 bits or 8 bits wide. The computer performs the 16-bit-to-8-bit or 8-bit-to-16-bit translation in hardware.

The CPU can do a 16-bit transfer to or from an 8-bit device. While the CPU only does one long 16-bit transfer, the PC/AT part of the system does two shorter 8-bit transfers, one of which involves a byte swap. A byte swap simply refers to the act of swapping a high-order byte onto the low-

order byte of the data bus or vice versa. Although they are no more common than any other bus cycle, they are a bit more complicated and require additional explanation.

The byte swap is accomplished via the 74LS245 at location U2. This transceiver connects the System Data bus low byte to the high byte. The general flow of events in a 16-bit read from an 8-bit device, for example, is as follows. Figure 7 is a rough diagram of what takes place.

The 386SX starts a 16-bit read cycle. The 82230 generates an 8-bit read into the low-byte 74F646 where it is latched. The 82230 deactivates IOR, but doesn't return READY286.

Next, the 82230 does another 8-bit read. DIR245 from the 82230 is high to set the data flow direction through the LS245. GATE245 from the 82230 enables the LS245, allowing the data from SDO-SD7 to flow through the transceiver into the high-byte 74F646 where it will be latched. The 82230 generates READY286. The CPU then reads all 16 bits of data and ends the cycle.

A 16-bit write to an 8-bit device is basically the same. Again the CPU will do one 16-bit cycle which will result in two 8-bit PC/AT cycles. During the second 8-bit cycle, DIR245 will be low to allow the byte swap to go from the high to the low byte, and the low order 74F646's outputs to the SD bus will be disabled. Data bus flow will, of course, be from the CPU to the peripheral on a write cycle. Since the CPU will drive valid data until it receives a READY back, latching is not necessary.

## X BUS

PC/AT on-board peripherals reside on the peripheral bus (X bus). An X bus access is basically the same as an 8-bit I/O expansion bus access, although the on-board peripheral bus cycles do not use some signals such as OWS and IOCHRDY. For this reason, X bus cycles use the default four wait states.

As with any PC/AT access by the CPU, the address flows through the 74LS245s between the SA and XA bus. The address flows from the SA to the XA bus except during a DMA cycle when this direction is reversed. In an Xbus transfer, the 82231 uses RDXDB to control the direction of flow through the SD to XD buffer. ACK from the 82231 enables this buffer.

## ROM

A ROM access is much like any other expansion bus memory cycle,

with only a few relatively minor differences.

In a ROM cycle, the 82335 activates the appropriate ROMCS\.. ROMCS\ is tied to the BIOS ROM's chip enable (CE\ ) pin and the 82230 MEMR\ signal is tied to the ROM's output enable (OE\ ) pin. The ROM outputs its data onto the RD bus. The combination of ROMCS\ and MEMR\ active allows the data to flow through the 74LS244 transceivers to the SD bus. A ROM access is treated as a standard expansion bus memory access and, as such, is a one-PC/AT-wait-state cycle.

**HOLD/HLDA**

The 386SX allows another master to take control of the bus. If HOLD is asserted, then some latency period later the 386SX will tristate all of its outputs except for HLDA, which it will drive high to signify that it is in a Hold Acknowledge state. During HLDA, address and command information is no longer coming from the CPU kernel. For this reason, the con-

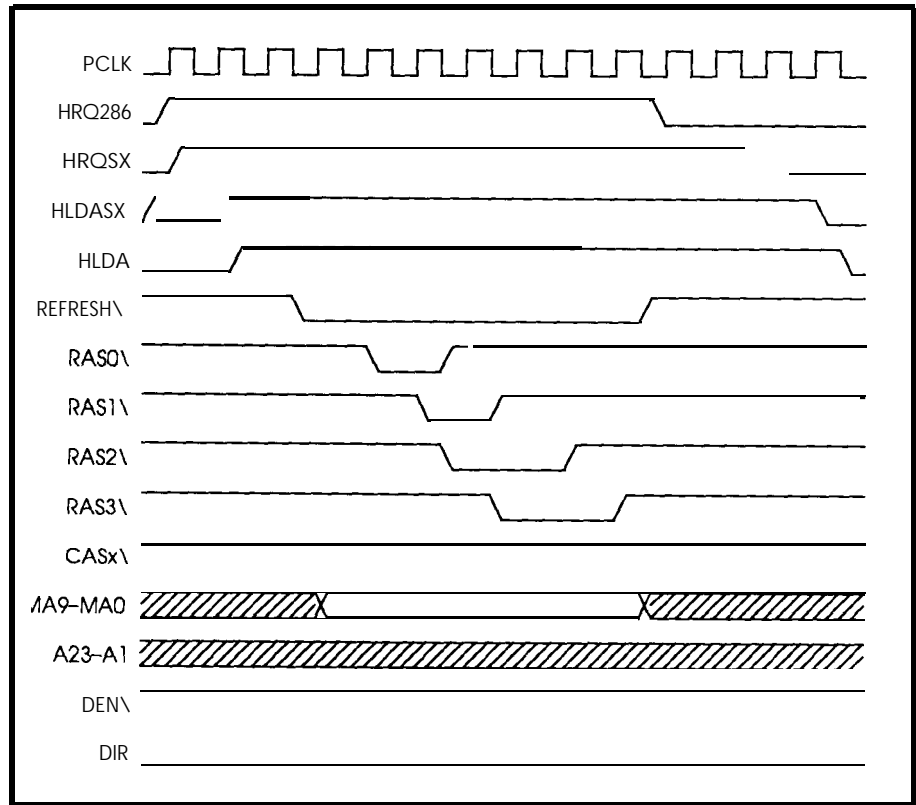


Figure 8— upon receipt of an active REFRESH \, the 82335 does a RAS-only refresh

control PAL uses HLDA active to cause addresses to flow from the SA bus to

the local address bus.

On our 80386SX computer there are three types of cycles that can occur during a HLDA:

**REFRESH**—Every 15  $\mu$ s, the 8254 Timer/Counter in the 82231 generates a refresh request. The 82231 requests a HOLD, then, upon receiving HLDA, it asserts REFRESH\ and drives a refresh address onto the XA bus. It also pulses XMEMR\ low at this time. The 82335 ignores the 82231's refresh address and generates its own refresh address instead.

The 82335 does a RAS-only refresh. Upon receiving REFRESH\ active, the 82335 puts a refresh address onto the MA bus. It pulses low for as many RASx\ lines as there are banks of DRAM present. The 82335 staggers these RASx\ outputs to reduce current surge. Figure 8 shows the REFRESH sequence.

**On-board DMA**—The 82231 receives a DMA request on one of the DRQ0-DRQ7 lines. From this, it sends a HOLD request to the CPU. Upon receiving HLDA, the 82231 activates either AEN1\ or AEN2\ to indicate that the DMA is driving an address onto the XA bus. It activates

# LOW

COMMUNICATIONS

508-485-1144

FAX 508-481-7222

BBS 508-460-9203

AD MONITORS  
SPECIALS!

## LOWBUDGETSPECIALS

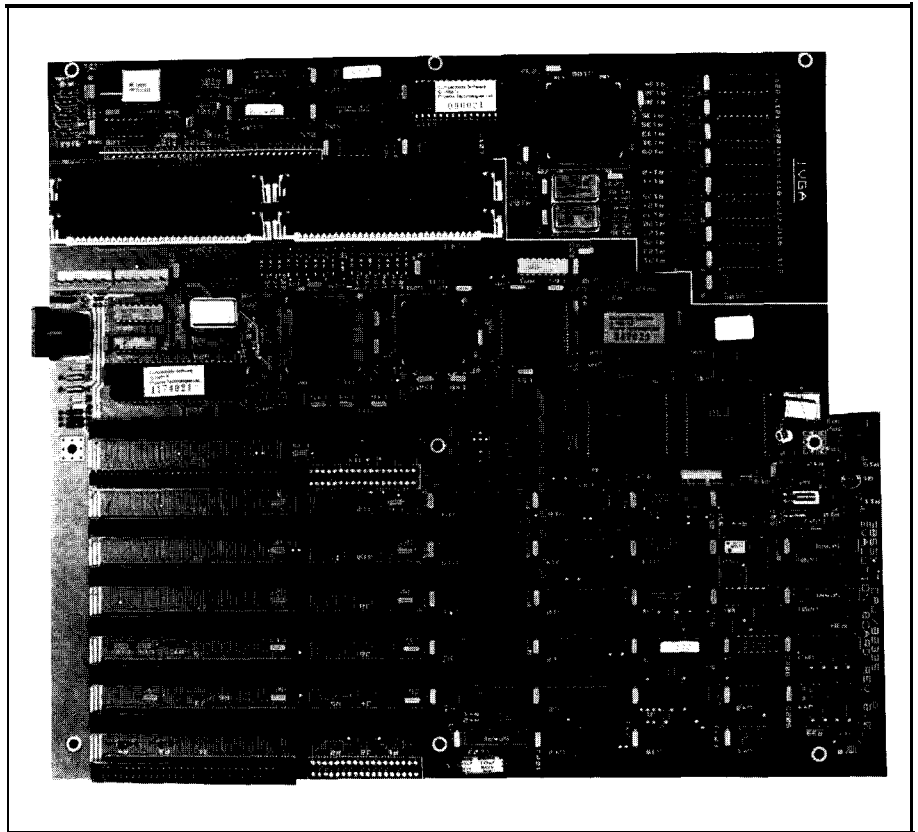
<b>19" COLOR SUPER VGA MONITOR</b>	
LIKE NEW 6 MONTH WARRANTY .....	\$800.00
USED 3 MONTH WARRANTY .....	\$600.00
USED WITHOUT CASE .....	\$300 TO \$600.00
<b>19" COLOR SONY 1280 X 1024</b> ....	<b>\$1600.00</b>
THESE SONY TRINITRONS HAVE A FULL 6 MONTH WARRANTY PARTS AND LABOR. LIMITED QUANTITY	
<b>16" 64KHZ AMI</b>	
<b>16" 64KHZ USED</b> .....	\$599.00
<b>14" Ikegami TTL VGA CHASIS</b> .....	\$249.00
NEW WITH 1 YEAR WARRANTY	
<b>19" PHILLIPS 1024X800 48 KHZ GRAY SCALE</b> .....	<b>\$350.00</b>
<b>NEW 1 YEAR WARRANTY - MAY BE ORDERED FOR VGA AT NO EXTRA CHARGE. WHEN USED IN VGA MODE THE MONITOR WILL RUN 800 X 600 X 256 GRAY SCALE OR 1024X768X16 GRAY SCALE ONLY</b>	
CALL US ABOUT OUR <b>LARGE VARIETY OF GRAPHIC CARDS !</b>	
194 Main ST. Marlboro, MA 01752	

that the DMA is driving an address onto the XA bus. It activates DMAEN\ to set the direction on the 74LS245s from the XA bus to the SA bus. Also, ACK goes high to turn off the transceiver between the SD and XD buses.

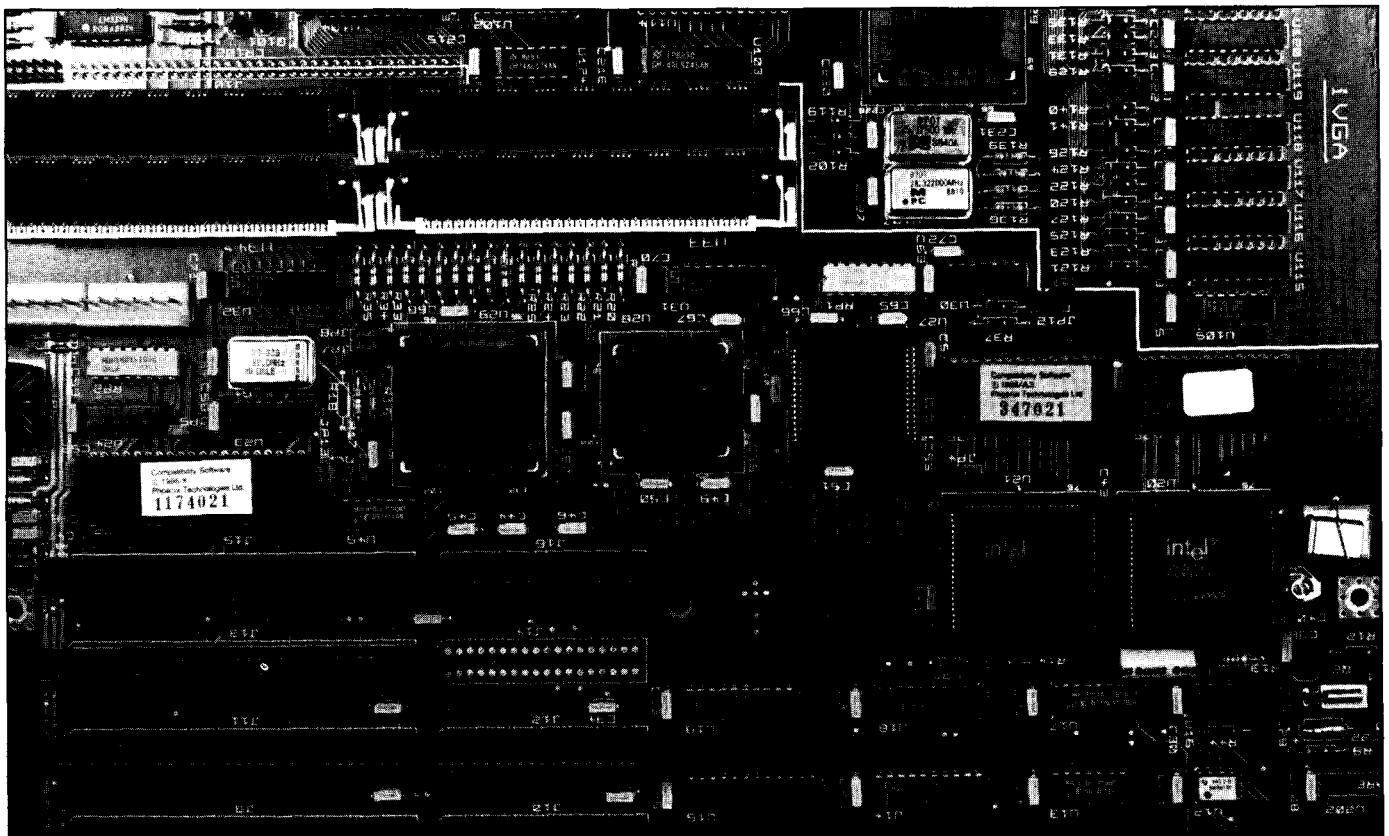
MEMR\ and IOW\ are activated at the same time for memory-to-I/O DMA cycles. IOR\ and MEMW\ are activated at the same time for I/O-to-memory DMA cycles. This means that the transfer must be between the same address on both the memory and I/O busses.

DMA cycles take ten PC/AT T-states. Since no bus cycle translation takes place, there is **no** translation time added to DMA cycles. The 8237 data sheet contains more details on DMA operation.

*Bus Master Operations*—An expansion card can also request control of the bus. The card must activate a DRQx line to cause the 82231 to activate HOLD. Upon receiving DACK, the card must activate MASTER\, causing the 82231 to relinquish control to the new bus master. The expansion card can now execute bus cycles.



**Photo 1**—The 80386SX motherboard is the same size as most AT-type motherboards, making placement in most inexpensive cases easy.



**Photo 2-A** close-up of the large chips on the 80386SX motherboard shows the 386SX, 387SX, 82235, 82230, and 82231.

Bus Master cycles are essentially the same as DMA cycles except that the Bus Master takes control of the system.

### INTR/INTA

Some time after INTR is received, the CPU will save its current state on the stack and acknowledge the interrupt request. The 386SX signifies an Interrupt Acknowledge (INTA) cycle by driving M/IO, D/C, W/R, and ADS low. All the address lines will also be low with the exception of A2. The 386SX does two INTA cycles with four idle states in between. LOCK is asserted from the beginning of the first cycle until the end of the second cycle. A2 is low in the first cycle and high in the second. Each cycle ends with the assertion of READY to the CPU.

The INTA cycle translates to M/IO286, SO, and S1 low to the 82230.

The 82230 then outputs INTA to signal an interrupt acknowledge cycle. The 82230 asserts READY to end each INTA cycle, and at the end of the second cycle places the interrupt vector onto the lower data bus. This vector points to an entry in the Interrupt Vector Table that contains the starting address of the Interrupt Service Routine (ISR) for that interrupt. The CPU then begins executing the ISR.

Acknowledgement of INTR will not occur if interrupts aren't enabled. A NMI, on the other hand, by its very nature cannot be masked out. When the CPU receives a NMI, it will not perform an INTA cycle. It will always jump directly to interrupt vector 2.

### A HIGH-INTEGRATION PLATFORM

The kernel of this system contains a 386SX CPU, a 387SX numeric coprocessor, and the local memory system. The memory system is 1 MB

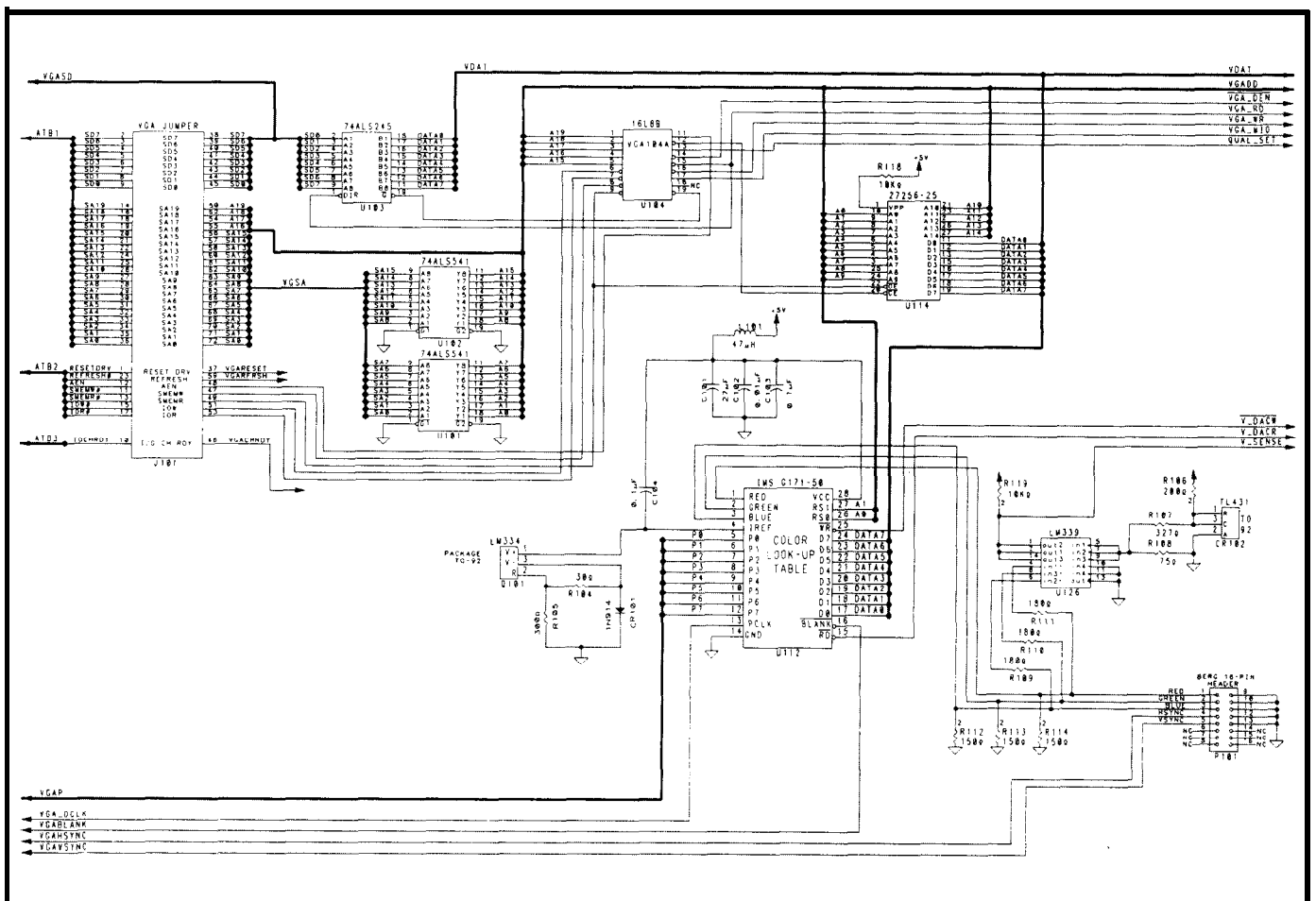
of 100-ns fast-page-mode DRAMs controlled by the 82335 Interface Device. The 82335 also connects the kernel to the PC/AT part of the system. The PC/AT part of the system contains the 82230/82231, 8042 keyboard controller, and a fully compatible ISA expansion bus.

I segmented the articles in such a way that the kernel could be used as the basis of an embedded control system or dedicated single-purpose computer. The 80386SX is a processor with many possibilities. Drop us a line to let us know how you're using it in your applications. ♦

*Daryl Rinaldi is an application engineer at Intel Corporation and works on the 386SX and associated products. In his spare time, he likes to explore California.*

### IRS

- 2 10 Very Useful
- 2 11 Moderately Useful
- 2 12 Not Useful







## FEATURE ARTICLE

A Design Contest  
project

by  
Winefred Washington

# A Low-Cost MIDI Sequencer

## Build an 8031 7-Based Stand-Alone MIDI Sequencer

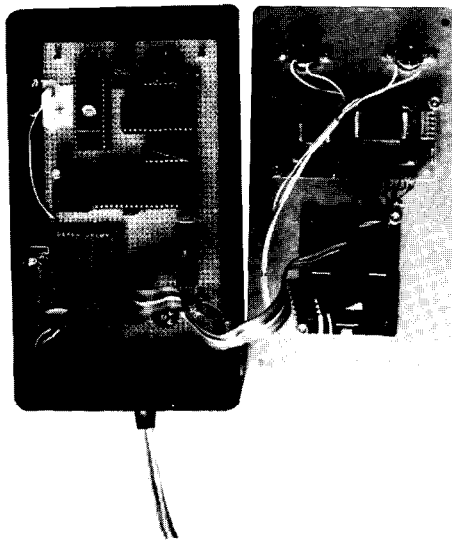
**M**IDI (Musical Instrument Digital Interface) is a serial communications standard that provides a standardized way to connect electronic instruments, computers, and peripherals. A MIDI sequencer is a device that records sequences of MIDI instructions and transmits them on demand. Since MIDI information can include pitch, attack, duration, volume, and instrumentation (among many other categories), a MIDI sequencer can record the complete data for an electronic performance and reproduce it. Many musicians use the capabilities of a sequencer to provide "back up" for live performances or recording sessions.

MIDISeq is a stand-alone micro-processor-based four-track MIDI sequencer. With MIDISeq, you can record up to four tracks of MIDI data, play them back, and save your compositions to an ordinary cassette tape recorder. MIDISeq doesn't require a PC or a MIDI adapter card. All you need is MIDISeq and a source of MIDI instructions (a synthesizer or MIDI instrument controller). It cost me around \$60 dollars, not including a few scrap parts (resistors, capacitors, etc.), to build the original MIDISeq.

### EINE KLEINE HARDWARE

MIDISeq consists of 8K of EPROM, 32K of static RAM, an LCD panel, a MIDI communications port, a switch matrix, and a simple cassette interface. The schematic is shown in Figure 1.

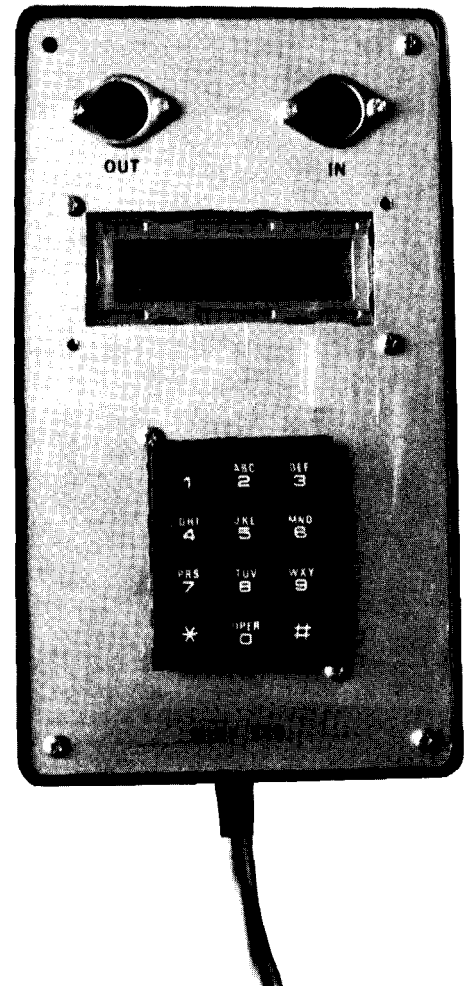
The LCD panel and switch matrix are connected to the 8031's peripheral port with the LCD operating in four-



bit interface mode. The LCD's data lines and the switch matrix's row lines share the same wires. When the 8031 is scanning the switch matrix, the LCD control lines are deasserted to ensure the LCD ignores the changes on the data lines.

The MIDI port is a 31.25-kbps serial interface. MIDI data is transmitted as one start bit, eight data bits, and one stop bit, a format which is compatible with the 8031's internal serial port. The 31.25-kbps rate is derived by dividing the 1-MHz processor clock by 32. The optocoupler is a general-purpose one obtained from Radio Shack.

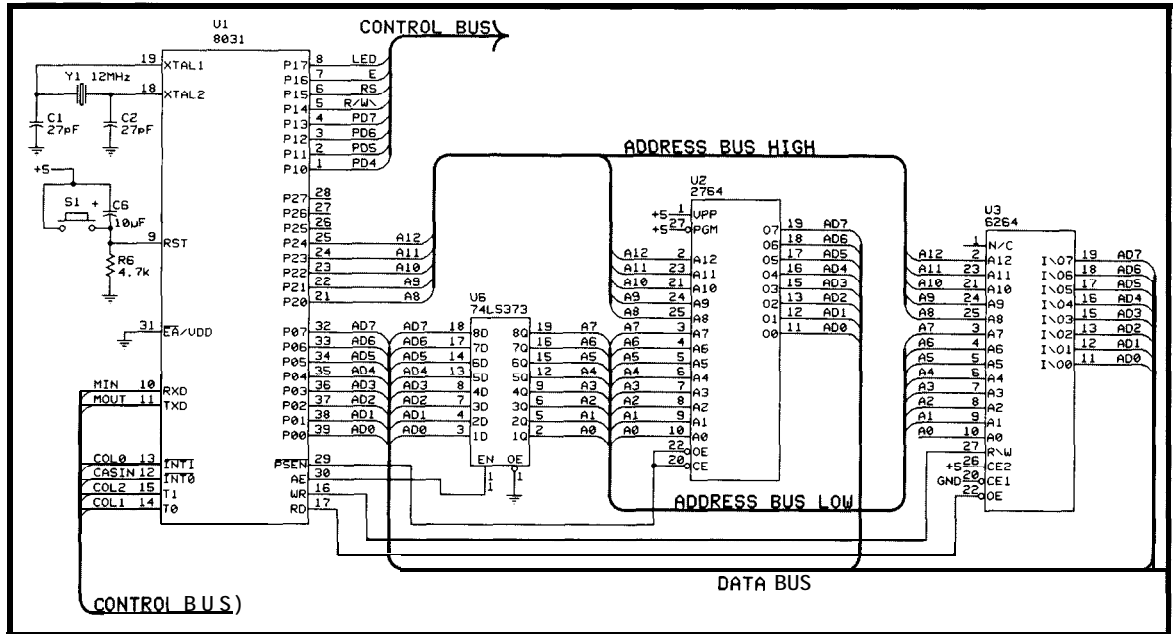
The cassette interface is tied to several spare peripheral lines. The 8031 toggles the cassette out line at the proper frequency to save the data bytes. The data transmission format is similar to RS-232: one start bit, eight data bits, and two stop bits. A "1" bit is represented by eight cycles of 2400 Hz, while a "0" bit is represented by four cycles of 1200 Hz. When loading data, the 8031 measures the width of the pulses to determine the data bits.



### THE SOUND OF SOFTWARE

The software for MIDISeq is partially written in BASIC and is compiled by BAS051, a BASIC compiler which I wrote. [Editor's Note: Software for this article is available for downloading from the Circuit Cellar BBS or on **CIRCUIT CELLAR INK On Disk #12**. See page 86 for downloading and ordering information.] I used BASIC to develop the user interface since BASIC code can be written and changed quickly. Several MIDI-specific functions were timing critical, so are in assembler.

Figure 1a--The heart of MIDISeq is an 8031 microcontroller. The circuit contains the usual 8K of RAM, 8K of EPROM, and low-address latch.



When MIDISeq is first turned on, the 8031 prints a title message and checks all the static RAM used to save MIDI data. The 8031 will display a message indicating the state of the RAM. This feature also assures that all of the address and data lines are connected properly and aids in debugging the circuit.

Next, MIDISeq asks for one of eight function codes. The two-digit function codes are:

- 00 Set **Tempo-Set** track playback speed (1-255).
- 01 Set **Track** On/Off-Enable or disable playing of selected track.
- 02 Set **Record** On/Off-Enable or disable recording.
- 03 Set **Record** Truck-Select record track 0-3.
- 04 Play Trucks-Start playing and recording of tracks.
- 05 Save **Tracks**—Save tracks to tape.
- 06 Load **Tracks**—Load tracks from tape.
- 07 Dump **Memory**—Dump the MIDI data program memory.

When a valid code is entered, the function **name** is displayed along with the current state of the function's parameters. At this point, you may enter a "#" (defined as the "Enter" key) and MIDISeq will ask for new input parameters or toggle the parameter. If you wish to abort the operation, pressing any other key will return to the function code prompt.

The Set **Tempo** function determines how fast the internal time clock

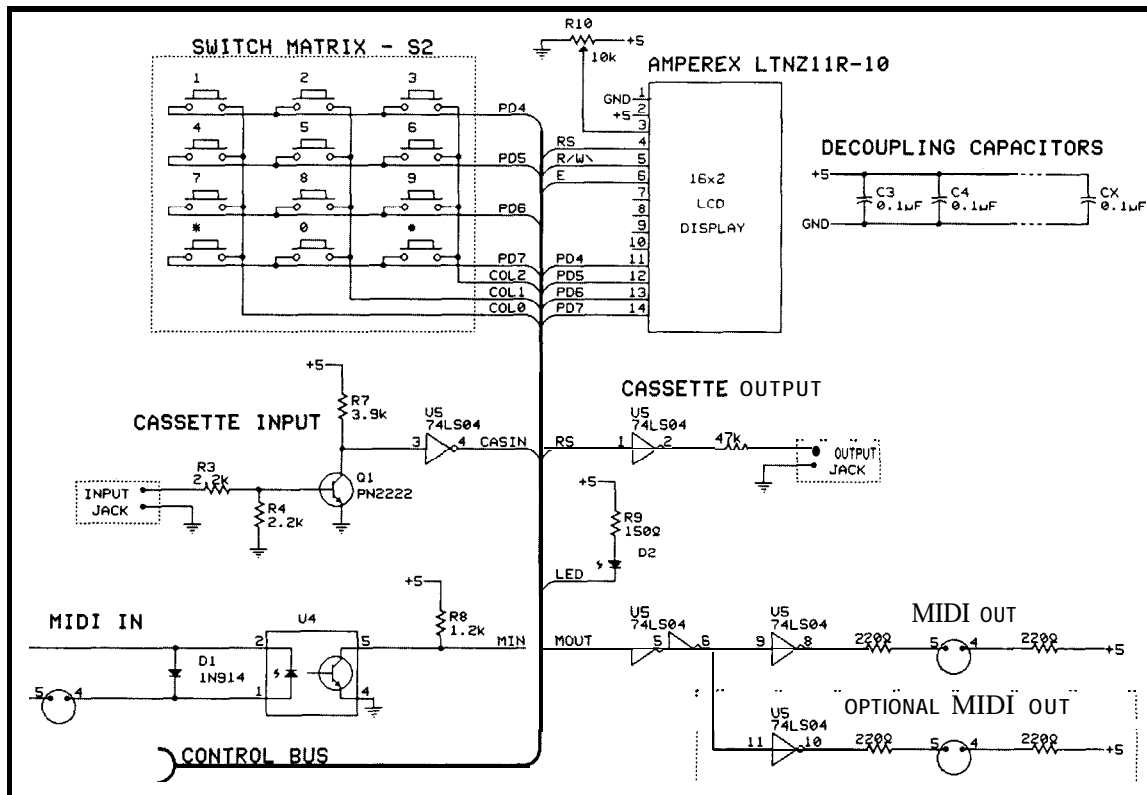


Figure 1 b--The interface portion of MIDISeq consists of a 4x4 keypad, 16x2 LCD display, cassette input and output, and MIDI input and output.

runs. The 8031's internal timer is used as the time reference when recording and playing MIDI data.

The Set **Truck On/Off** command enables or disables playing a MIDI track. Only tracks containing valid MIDI data should be enabled.

The **Record On/Off** function is a switch which enables or disables recording. This is necessary when all the data is recorded and you are ready to review your composition.

The **Record Truck** command is used to select the track for MIDI data storage. Playing of this track must be disabled during recording.

The **Play Trucks** function starts MIDISeq recording and playing MIDI data contained in its memory. Pressing any key on the keypad stops the sequencer.

The Save **Trucks** function saves the MIDI data in RAM to cassette tape. The tape interface saves data at only about 25 bytes per second, so it takes around 20 minutes to save the entire memory space.

The **Loud Trucks** command retrieves MIDI data from cassette tape. Since data is saved at about 25 bytes per second, the 8031 can only load data at the same rate.

The **Dump Memory** function displays the contents of memory used to store MIDI note on and note off data, along with a time stamp of when the data was received. When a key is pressed or released, a music keyboard transmits the event out its MIDI port. MIDISeq receives the data and stores it in memory. The storage format is as follows:

N+0—LSB of time stamp  
N+1—MSB of time stamp  
N+2—Note On/Off Command  
Byte  
N+3—Note Value  
N+4—Note Velocity

where *N* represents the event number.

MIDISeq maintains the time through a clock interrupt. When tracks are played, the events' time stamps are compared, and the notes are played when the proper time is reached.

## MAKE YOUR OWN MUSIC

I enjoyed designing and building MIDISeq. It was planned not as a competitor to the expensive professional MIDI sequencers, but as a simple, low-cost way to experiment with MIDI. The hardware works well for these criteria, and I can't think of significant enhancements that wouldn't add greatly to the cost or complexity of the **project**. On the other hand, the firmware has some serious potential for upgrades. An editor of the MIDI data stored in RAM is the

most obvious starting point for firmware upgrades. Regardless of whether you build the MIDISeq as given or use the design as the basis for bigger and better things, I hope you enjoy working with MIDI as much as I have. ❖

*Winefred Washington, Jr. is currently an engineer I/O subsystems - tems for CAD workstations. He has a BSEE from the University of Tennessee.*

## IRS

2 13 Very Useful  
2 14 Moderately Useful  
215 Not Useful

# GET TO WORK!

## A New Project

Our line of macro Cross-assemblers are easy to use and full featured, including conditional assembly and unlimited include files.

## Get It To Market--FAST

Don't wait until the hardware is finished to debug your software. Our Simulators can test your program logic before the hardware is built.

## No Source!

A minor glitch has shown up in the firmware, and you can't find the original source program. Our line of disassemblers can help you recreate the original assembly language source.

## Set To Go

Buy our developer package and the next time your boss says "get to work", you'll be ready for anything.

## Quality Solutions

PseudoCorp has been providing quality solutions for microprocessor Problems since 1985.

## BROAD RANGE OF SUPPORT

- Currently we support the following microprocessor families (with more in development):

Intel 8048	RCA 1802,05	Intel 8051	Intel 8096
Motorola 6800	Motorola 6801	Motorola 68HC11	Motorola 6805
Hitachi 6301	Motorola 6809	MOS Tech 6502	WDC 65C02
Rockwell 65C02	Intel 8080,85	Zilog 280	NSC 800
Hitachi HD64180	Motorola 68000,8	Motorola 68010	

- All products require an IBM PC or compatible.

**Cross-Assemblers as low as \$50.00**  
**Simulators as low as \$100.00**  
**Cross-Disassemblers as low as \$100.00**  
**Developer Packages as low as \$200.00**  
**(a \$50.00 Savings)**

## So What Are You Waiting For?

Call us

**PseudoCorp**

**Professional Development Products Group**

716 Thimble Shoals Blvd, Suite E  
Newport News, VA 23606

**(804) 873-1947**

## WHY BOTHER WITH MIDI?

If you are an experienced keyboard player, the attraction of MIDI has long been obvious: A single keyboard can control a number of different instruments. Furthermore, peripheral equipment lets you set up a complicated accompaniment, against which you can improvise to your heart's content. More recently, MIDI controllers based on guitars, violins, woodwinds, and even the human voice have given the power of multi-instrument control to a whole new group of musicians.

Computer users with musical interests but little musical experience are yet another group to have benefited from MIDI.

MIDI is an asynchronous serial data protocol similar, in many ways, to RS-232 and RS-423. Initially specified to run at 19.2 kbps, MIDI's rate was finally set at 31.25 kbps for reasons of speed and electrical simplicity. MIDI uses a standard protocol of 8 data bits, 1 stop bit, and no parity. Each MIDI event requires that least three bytes be sent. For example, to send the command to play a single note to a synthesizer, byte one carries

a channel identifier and command identifier, byte two is the note value, and byte three is the velocity value. Commands to change voices (instruments) or alter other playing parameters would be sent in similar form.

In addition to synthesizers and controllers, computers and MIDI peripherals may be used to complete a MIDI system. Personal computers are most often used as highly capable sequencers (more on these in a moment), music editors, or composition/transposition aids, where computer software captures MIDI signals and records the note values in standard musical notation. MIDI peripherals include patch boxes, routers, special effects boxes, and devices for adding and synchronizing SMPTE timing tracks for motion picture or video work. One of the most common peripherals is the sequencer, which acts as a performance recorder by capturing the sequence of MIDI commands rather than actual sounds.

Simple sequencers, such as the one presented in this article, are most often used to record a "track" or portion of a composition that is too complex to be played in its entirety by a single player. In a popular song, the sequencer will often be used to record the percussion, bass, and rhythm tracks, which are then played back to accompany the lead instrumentals and vocals. This is a common technique for musicians who want to make professional-sounding demo tapes, but who don't have access to a multi-track tape recorder (or large band).

More complex sequencers, of the type frequently executed in software on a personal computer, add the function of editing. Editing adds the possibility of going back to perfect a recorded track by adjusting one parameter at a time ("That note *right there* needs to be *just a little louder*"), or changing the tempo without altering the pitch or voicing of the notes. (This is the feature that can let you record at student practice tempo and play back sounding like Horowitz.) Just as a word processor can't make you another **Hemingway**, a state-of-the-art MIDI setup is no substitute for talent. It is, however, a way to take the talent you have and do more with it than before.

# **KELVIN** Electronics

7 Fairchild Ave. Plainview, NY 11803 1 (516) 349-7620

## WHY PAY RETAIL?

**TIMER - LM555** (in 100+ Qty)  
Stock No. 600021 **20 ¢** ea.

**FLUKE 77 - \$135.**  
**FLUKE 87 - \$246.**

**L.E.D. RED** (in 100+ Quantity)  
Stock No. 260020 **7 ¢** ea.

**Duracell Alkaline**  
9 Volt Battery  
**\$2.16 ea \$1.94/12+ ea.**

**Resistors 1/4 Watt 5%**  
**1 ¢** ea. (200 per Bag)

**9V Battery Snap & Holder**  
Snap **\$.15 ea. \$.10/100+**  
Holder **\$.20 ea. \$.10/100+**

**Transistor 2N2222**  
Stock No. 630041 **20 ¢** ea.  
(100+ Quantity)

**Weller Soldering Iron 25w**  
Model SP23 Stock No. 810002  
**\$8.55 ea \$8.12/12+ ea**

**Neon Bulb NE-2**  
Stock No. 260003 **12 ¢** ea.  
(100+ Quantity)

**Kester Solder 60/40 1/32"**  
**1 lb. Roll Resin Core** Stock No. 580005  
**\$9.95 ea \$8.96/4+ ea**

**SUB-MINIATURE MOMENTARY**  
**PUSH SWITCH**  
Red or Black **28 ¢** ea. (100+)

**Speaker - PM Standard**  
2", 8 ohm, .1 watt Stock No. 350009  
**\$1.25 ea. \$.88/20+ ea**

Additional Discounts Given to Substantial Quantity Purchases.  
FAX your requirements - 1 (516) 349-7830 (FAX Number)  
CALL OR WRITE FOR OUR FREE CATALOG.

## CREATIVITY WANTED

KELVIN ELECTRONICS IS A SUPPLIER OF EDUCATIONAL TECHNOLOGY KITS (ROBOTIC, MOTION CONTROL, COMMUNICATION, TRAINERS, HARDWARE AND SOFTWARE) WITH AND WITHOUT COMPUTER INTERFACES. YOUR DESIGNS COULD QUALIFY FOR PUBLICATION IN OUR CATALOG.

NOT RESPONSIBLE FOR TYPOGRAPHICAL ERRORS

## Simulated Reality

### Simulating Systems for 8051 Debugging

Writing firmware or software is much the same: both are a series of instructions that the CPU executes in sequence. Debugging firmware is unique because the target system generally does not have the full keyboard, display, hard disk, or other niceties we demand on our PCs and workstations.

The software world has vaporware, which is mythical code for real hardware. Firmware goes this one better, because you often write code for a machine that doesn't exist. Even when the target hardware is available, it usually has behavioral quirks that turn firmware debugging into an Adventure game: "you are alone in a

maze of twisty, turny instructions, any of which may do unpredictable things..."

As a result, no firmware development shop is complete without a software simulator. Although the details vary, a simulator is basically an interpreter that runs on your PC (or workstation, mainframe, or whatever), accepts executable code for the target CPU, and simulates the effects of that code on the target CPU's hardware. You can single step through the firmware instructions, observe CPU registers and I/O ports on the PC's screen, and change code and data on the fly to fix bugs or check boundary conditions.

A good simulator will maintain a record of the CPU's state prior to the current instruction, so you can back up to discover how the system got into its current predicament. This lets you track down obscure bugs long after their dirty work is done, because you can stop when you see something odd and "undo" until you catch the bug in action.

However, most simulators model only the CPU state and pay scant attention to the rest of the target system's hardware. Some simulators accept input data from a disk file and apply it to a parallel or serial input, but there are often Procrustean restrictions on timing and data format. More complex simulators allow you to write programs (albeit in a bizarre modeling language) to simulate target hardware chips and devices, but price tags are near the stratosphere in this league.

So what can you do if you really, desperately, need a target hardware simulator on a CPU simulator budget? In some cases, you can replace the missing hardware with firmware and save a bundle. Thus the title of this column: you are simulating reality for your code. Who's in charge, anyway?

Depending on the missing hardware's function, the requisite firmware can range from trivial to impossible. The simplest cases occur when the hardware generates data that can be faked with a trivial algorithm. The most difficult hardware to simulate produces complex data that must be absolutely letter perfect in both time and content.

```

;-----
; Delay for number of ticks in A

WaitTicks PROC
PUBLIC   WaitTicks

        MOV     B,A           ; save tick count
GetVar  NETTIME           ; get current time value
ADD     A,B               ; tack on delay value
MOV     B,A               ; save for loop testing

        JB     SimMode,L?dlydone ; bypass if simulating

L?dlyrun GetVar  NETTIME           ; decide if we're done
        CJNE  A,B,L?dlyrun

L?dlydone RET

WaitTicks ENDP

```

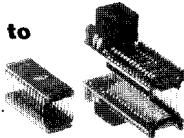
Listing 1—Timing loops can be bypassed when simulating. The "SimMode" bit is set only when the code is running in the Avocet simulator. "GetVar" is a macro that fetches a variable from external RAM; the "NETTIME" location is incremented by the 5-millisecond network timer tick.

# DEVELOPMENT TOOLS

For the IBM PC and Compatibles

## 2764 ROM EMULATOR

Appears as 2764 to target system. Connects to PC parallel port.



- Only slightly larger than an actual 2754
- Plugs into target ROM socket and connects to PC parallel port via modular telephone cable
- Accepts 8K x 8 SRAM or EEPROM (non-volatile)
- 3-state reset restarts target after downloading
- Uses HC/HCT logic for CMOS & TTL compatibility
- Loads Intel Hex, Motorola S, hex, and binary files
- Command line software can be run from batch files for automatic downloading after assembly

\$129 (without memory)    \$149 (with 120ns SRAM)    \$159 (with 250ns EEPROM)

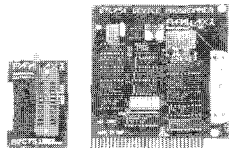
## 8051 FAMILY ASSEMBLER

- Works on all 8051 derivatives
- Supports standard Intel syntax
- Allows local labels and include files
- Labels may be up to 32 characters
- Generates assembly listings
- Outputs Intel Hex

**\$95**

Quick • Dependable • Clean Operation

## 87C751 DEVELOPMENT PACKAGE



87C751 Programmer & 8051 Family Assembler

- Programs Signetics 87C751 and 87C752 micro-controllers
- Handles EPROM, encryption key, and security bits
- Loads and saves Intel Hex, ASCII hex, and binary file formats

Dev. Package-DIP \$259 (includes choice of DIP adapter)    87C751 DIP adapter \$39  
 87C751 PLCC adapter \$79  
 Dev. Package - PLCC \$299 (includes choice of PLCC adapter)    87C752 DIP adapter \$39  
 87C752 PLCC adapter \$79

**PARALAX**

(916) 721-8217

Parallax, Inc.  
6200 Desimone Lane, #69A  
Citrus Heights, CA 95621



California residents add 6.5% sales tax  
Shipping: No charge for UPS ground,  
\$7.00 for UPS 2nd day, \$15.00 for UPS next day

Reader Service X144

```

NORMALMODE EQU 0 ;1 for "real" code
; 0 far simulation

;-----
; Delay for number of ticks in A

WaitTicks PROC
PUBLIC WaitTicks

MOV B,A ; save tick count
GetVar NETTIME ; get current time value
ADD A,B ; tack on delay value
MOV B,A ; save for loop testing

%IF NORMALMODE ; nonzero for "real" code
L?dlyrun GetVar NETTIME ; decide if we're done
CJNE A,B,L?dlyrun
%ENDIF

RET

WaitTicks ENDP
    
```

Listing 2-Conditional assembly can remove entire sections of code, but this requires two versions of the executable program: the 'real' program and the 'simulation' program. Problems may occur when code and data addresses differ between the versions.

The examples in this column are drawn from the MC-Net firmware I wrote a few months ago. Using the tricks described here, that firmware will run quite happily under the Avocet simulator I used to develop it, issuing and responding to polls, transmitting data, and listening for messages from other nodes. Indeed, I can simulate the reality of a network from the keyboard and the firmware will run blissfully unaware of the deception.

### TIME PASSES QUICKLY...

...on the real target hardware. But when a simulator is processing firmware, time passes with glacial stateliness. For example, the Avocet AVSIM51 8051 simulator runs about 1200 times slower than "real time" on a 20-MHz '386 processor. A single 1.09- $\mu$ s machine cycle takes about 1.3 ms to execute. That doesn't sound too bad, but a 50-ms delay loop takes about a minute to expire. On an 8-MHz AT,

```

;- set up serial port rate using Timer 2
; CRCx is Avocet's way of saying RCAP2x

JB SimMode,L?highrate ; always high in simulation!

MOV CRCH,#HIGH-72 ; low rate = 4800
MOV CRCL,#LOW-72
JNB RATESEL,L?setrate ; high bit -> not
; strapped to gnd

L?highrate MOV CRCH,#HIGH-18 ; high rate = 19200
MOV CRCL,#LOW-18

L?setrate MOV TH2,CRCH ; punch current count!
MOV TL2,CRCL
    
```

listing J-During simulation the MC-Net serial port runs at the higher bit rate. The SimMode bit forces the higher rate regardless of the RATESEL input bit.

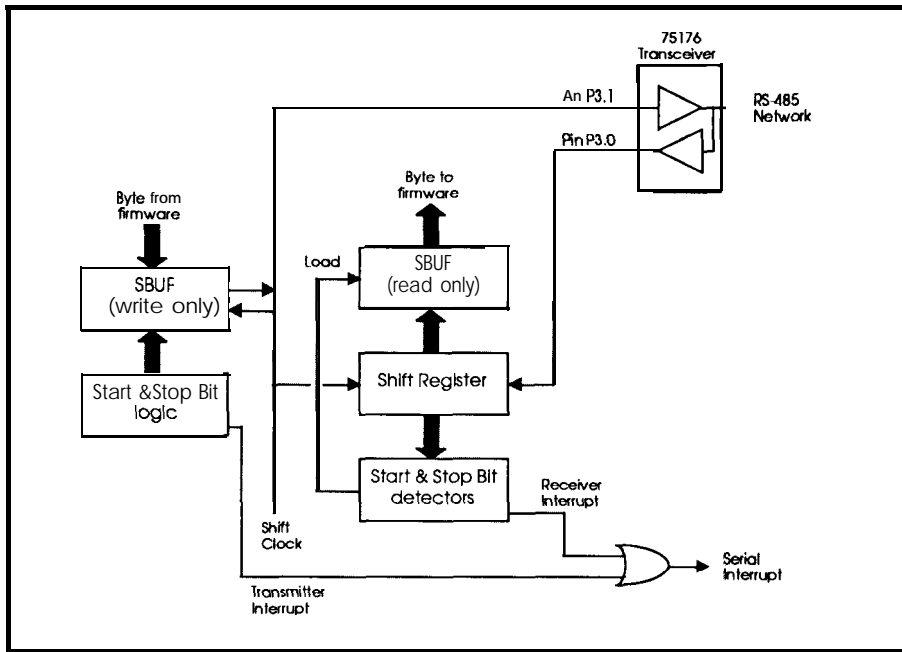


Figure 1 -The network transceiver echoes characters from the 8052 serial output back to the input. The SBUF special function register has both output and input functions.

it will take four minutes, which is more than enough to demand a better way.

Most delay loops interface with external hardware that isn't present in the simulator, so there is little value in sitting out the whole loop. Listing 1 shows a timing subroutine that ordinarily delays for multiples of the 5-ms network timer tick, but is bypassed when the code runs under the simulator.

The key is `SimMode`, a single bit in the 8051's internal data memory. The startup routine clears `SimMode` to ensure that it is OFF during normal operation. When I run the code under

the simulator, however, it is a simple matter to single-step past the CLR `SimMode` instruction and turn the bit ON. From that point the firmware bypasses all timing loops with no further attention.

Using this method has two disadvantages: it requires a bit in the 8051's (very limited) internal memory and a line of code wherever the bit is tested. Listing 2 shows how to use conditional assembly to "snip out" the delay loop; the code simply disappears from the program when the `NORMALMODE` symbol is not true.

But removing chunks of your firmware shifts the remaining code around

in memory; the program you are simulating is not the same as the program on the target system. This may have unexpected side effects. For example, because 8051 `AJMP` and `ACALL` instructions cannot cross 2K-byte page boundaries, other sections of code may cause changes in ways you don't expect. Indeed, if the excised code moves an `AJMP` to the other side of a 2K boundary, the program simply will not work, although the assembler or linker should catch this problem and issue a diagnostic message.

With that in mind, I have settled on adding the `simMode` bit and a few instructions to my programs rather than deleting blocks of code with conditional assembly. Regardless of how you do it, make sure that you don't cause more problems than you cure!

## TWEAKING CODES

Although EEPROM is gaining in popularity, most firmware reads its initial settings from a DIP switch or jumper block during the startup routine. Your simulator will certainly allow you to force that input port to a particular value, but the `SimMode` bit can come in handy here, too.

For example, MC-Net firmware on an RTC52 reads the bits of port P1 to determine the network node address, decide if the node is a net master, and pick the network bit rate. `SimMode` forces each of those decisions to a particular value so the initial settings are correct (or at least conven-

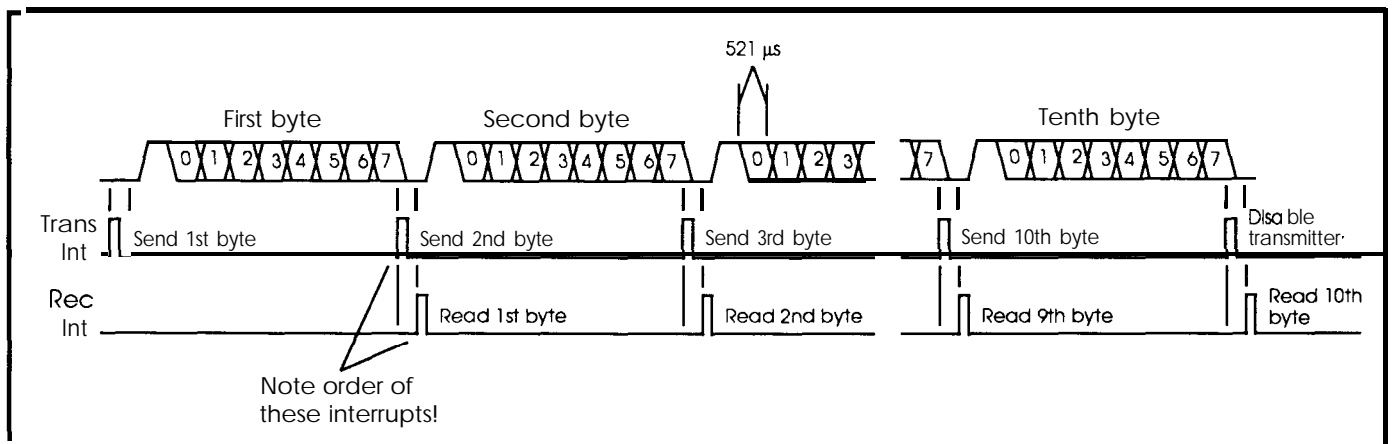
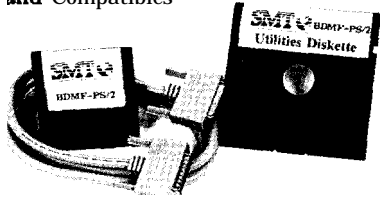


Figure P-Although the RS-485 transceiver echoes bytes sent from the serial port, the transmitter sends two bytes before the receiver returns the first one.



## THE INTERCHANGE™

Bi-directional **Data Migration Facility** for IBM **PS/2**, AT, PC, PORTABLE and Compatibles

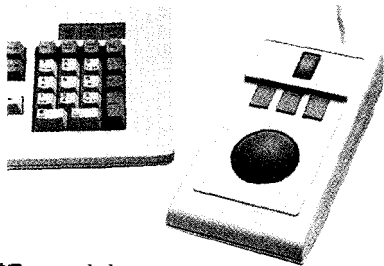


### Features:

- \*Parallel port to parallel port.
  - \*Economical method of file transfer.
  - \***Bi-Directional** file transfer easily achieved.
  - \*Supports all **PS/2 systems** (Models 30, 60, 60, and 80).
  - \*Supports IBM PC, XT, AT, Portable and 100% compatibles.
  - \*Supports 3 1/2 inch and 5 1/4 inch disk **transfers**.
  - \*Supports hard disk transfers.
  - \*Supports **RAMdisk** file transfers.
  - \*The SMT 3 Year Warranty.
- ONLY \$39.95

## FastTrap™

The pointing device of the future is here!



- \*Two and three axis pointing capability.
  - \*High resolution trackball for X and Y axis input.
  - \*High resolution **fingerwheel** for Z axis input.
  - \*Use with **IBM® PC's, XT's, AT's** and compatibles.
  - \*Three input buttons.
  - \*Full hardware emulation of Microsoft® Mouse.
  - \*Standard RS-232 serial interface.
  - \*Includes graphics drivers and menu generator.
  - \***Easy** installation.
  - \***1** year warranty.
  - \***Made** in U.S.A.
- ONLY \$149.00



**LTS/C Corp.**  
167 North Limestone Street  
Lexington, Kentucky 40507  
Tel: (606) 233-4156

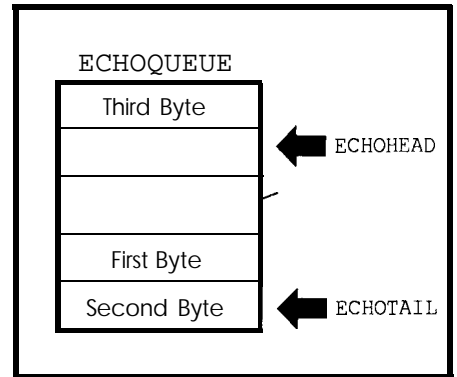
Orders (800) 872-7279  
Data (606) 252-8968 (3/12/2400 8-N-11)  
VISA, Mastercard, Discover Card,  
TeleCheck

Reader Service #134

ient) for simulation, regardless of the actual P1 value.

Listing 3 shows how the firmware sets the serial bit rate. Under normal circumstances `SimMode` is zero and the choice depends on the value read from the `RATESEL` input bit (which is P1.5 on RTC52 processors). During simulation, with `SimMode` high, `RATESEL` is ignored and the bit rate is always 19,200 bps.

Another area that should behave differently under simulation is the memory test performed immediately after the power-on reset. There is no reason to test simulated RAM (what's there to test?), so `SimMode` allows the firmware to force the "correct" regis-



**Figure 3**—A ring buffer holds outbound bytes to simulate the hardware transceiver echo. `ECHOHEAD` points to the location for the next outbound byte, `ECHOTAIL` points to the next byte to be echoed to the receiver. As shown, three bytes have been transmitted and one echoed to the receiver firmware.

```

; fetch outgoing byte & mail it off

        MOVX    A, @DPTR
        MOV     SBUF, A

; put byte into echo queue if we're simulating
; if queue has more than two entries, start the receiver...

        JNB     SimMode, L?nosim

        MOV     B, A                ; save output char

        PUSH   DPH
        PUSH   DPL

        GetVar ECHOHEAD            ; set byte into queue
        PUSH   ACC
        ADD    A, #ECHOQUEUE
        MOV    DPL, A
        MOV    A, B
        MOVX   @DPTR, A

        POP    ACC                ; tick head index
        INC   A
        ANL   A, #(ECHOQSIZE-1)
        SetVar ECHOHEAD

        GetVar ECHOLENGTH          ; tick length indicator
        INC   A
        SetVar ECHOLENGTH

        ADD    A, #-2              ; have two entries?
        JNC   L?endsim
        SETB  RI                  ; start receiver

L?endsim POP    DPL
        POP    DPH

L?nosim
    
```

**Listing 4**—The MC-Net firmware buffers outgoing data bytes in RAM during simulation.

```

MOV      A,SBUF      ; pick up the incoming byte

; fake hardware echo if we're simulating
; if drain is on, set RI to get the rest of the queue elements

JNB      SimMode,L?nosim; use faked net?
PUSH     DPH
PUSH     DPL
PUSH     B
PUSH     ACC          ; save SBUF byte

GetVar   ECHOLENGTH ; anything in queue?
JZ       L?noget
CJNE    A,#1,L?getbyte ; if 1, queue will be empty
CLR     SimEchoDrain ; done with queue, endit

L?getbyte DEC A          ; tick counter down
SetVar   ECHOLENGTH

MOV      C,SimEchoDrain ; propagate drain mode
MOV      RI,C

POP      ACC          ; discard SBUF byte

GetVar   ECHOTAIL    ; output slot index
MOV      B,A
ADD      A,#ECHOQUEUE ; get pointer to entry
MOV      DPL,A
MOVX    A,@DPTR      ; fetch that byte
PUSH     ACC

MOV      A,B          ; tick output slot
INC      A
ANL     A,#(ECHOQSIZE-1)
SetVar   ECHOTAIL

L?noget  POP ACC       ; recover SBUF or queue byte
POP      B
POP      DPL
POP      DPH

L?nosim

MOV      A,SBUF      ; pick up the incoming byte

; fake hardware echo if we're simulating
; if drain is on, set RI to get the rest of the queue elements

JNB      SimMode,L?nosim; use faked net?
PUSH     DPH
PUSH     DPL
PUSH     B
PUSH     ACC          ; save SBUF byte

GetVar   ECHOLENGTH ; anything in queue?
JZ       L?noget
CJNE    A,#1,L?getbyte ; if 1, queue will be empty
CLR     SimEchoDrain ; done with queue, endit

L?getbyte DEC A          ; tick counter down
SetVar   ECHOLENGTH

MOV      C,SimEchoDrain ; propagate drain mode
MOV      RI,C

POP      ACC          ; discard SBUF byte

GetVar   ECHOTAIL    ; output slot index
MOV      B,A
ADD      A,#ECHOQUEUE ; get pointer to entry
MOV      DPL,A
MOVX    A,@DPTR      ; fetch that byte
PUSH     ACC

MOV      A,B          ; tick output slot
INC      A
ANL     A,#(ECHOQSIZE-1)
SetVar   ECHOTAIL

L?noget  POP ACC       ; recover SBUF or queue byte
POP      B
POP      DPL
POP      DPH

L?nosim

```

listing 5—The Receiver Interrupt handler code to extract bytes from the ring buffer.

# EMBEDDED SYSTEMS DESIGNERS START USING INTEL'S 8096

★★★NOW★★★

## COMPLETE DESIGNERS TOOL-KIT UNDER

### \$500.00

#### MATE97

Single Board Computer-I 2MHz 8097 Processor, single voltage supply, RS232 serial support, 50-pin I/O header, 32K EPROM, 32K RAM

#### MON97 Debugger/Monitor EPROMs—

uses terminal or PC, download HEX, assemble, disassemble, step, dump RAM, set breakpoints, over 40 BIOS calls available.

#### SMALL C-8096 Compiler

#### MATECOM-Terminal Emulator (IBM-PC)

Complete Developers Tool-Kit with wall mount power supply and prototyping board \$495.00

#### 8096 Assembler (IBM-PC)

\$125.00

Other tools available

*Call or write today for data sheets and information.*

**B.G. Associates**  
P.O. Box 66  
Bowie, MD 207150066  
(301) 509-6748

tervalues without wasting time on the actual tests.

You might also bypass any lengthy sign-on or status messages that take minutes to trickle out the serial port. `SimMode` can bypass the code that displays the messages, or you can substitute a shorter message to exercise the code but not spend much time doing it.

Of course, you must verify the code skipped by `simMode` at least once during development! The only time you can successfully bypass code is when you know it works; shooting a

bug that crops up only when you're not simulating can be exceedingly frustrating.

#### FIXING THE MISSING

So far, the `SimMode` bit has just skipped code or made a slight modification of the normal data values. You must do more when the firmware must make up for missing hardware. Depending on what's not there, you may need a considerable amount of code. The MC-Net firmware for RTC52 boards is a good example of this, be-

cause it simulates the network RS485 transceiver hardware and network wiring.

The 75176 transceiver echoes all transmitted bytes back to the receiver as well as sending them out over the network. When the board is acting as the network master, the firmware depends on those echoed bytes to trigger outgoing messages. In simulation, of course, there is no 75176 transceiver, no connection between the serial pins, and no data echo from output to input.

To help you understand what is going on, I must explain how the 8052's serial ports work. Unlike IBM PC serial ports, everything happens on one chip and has relatively simple control and status flags. If you're familiar with how the PC hardware works, pay close attention to the differences!

Figure 1 sketches the on-chip serial port hardware and the off-chip 75176 network transceiver that produces the echoes. This is a "big picture" block diagram, so you'll have to refer to the chip and board documentation for gate-level details. The 8051 serial ports have a myriad of modes, options, switches, and features that don't apply to this discussion and thus don't show up in the figures.

Transmission starts when the firmware writes a byte into the leftmost `SBUF` special function register shown in Figure 1. The hardware shifts a start bit, eight data bits, and a trailing stop bit out the serial output pin (P3.1). At the junction between the final data bit and the stop bit, the Transmitter Interrupt flag goes active to signal the firmware that the hardware can accept a new byte in `SBUF`. If serial interrupts are enabled, the `TI` flag will generate an interrupt and vector the processor to the serial interrupt handler.

The firmware can set the `TI` flag directly to force an initial transmitter interrupt. While this level of control is foreign to PC-class machines, it provides a convenient way to "kick start" the transmitter interrupt handler. The MC-Net firmware takes advantage of this feature to simplify the code that composes and sends messages, be-




**ATTENTION  
COMPUTER  
STORES**

**Put  
Circuit Cellar INK  
to work for you.**

Circuit Cellar INK readers are engineers, programmers, consultants, and serious computer technologists. Bring them into your store by selling Circuit Cellar INK.

Circuit Cellar INK's Direct Dealer Sales Program lets you increase your store traffic-increase your sales-increase your profits with minimal risk and up-front investment.

For information on how your business can become part of the growing Circuit Cellar INK success story, write or call:



**Increased Sales =  
Increased PROFIT**

**Circuit Cellar INK  
Dealer Sales Program  
4 Park Street  
Vernon, CT 06066  
(203) 875-2199**

cause it need not be concerned with actually sending the first byte out the serial port.

On the other end, reception begins when the hardware detects a valid start bit at the serial input pin (P3.0). The next eight bits are shifted into a register and the contents of that register are loaded into the rightmost *SBUF* register in Figure 1 at the middle of the stop bit. The Receiver Interrupt flag goes active when the transfer occurs and will cause an interrupt if serial interrupts are enabled.

If the firmware doesn't read the *SBUF* holding register before the middle of the next stop bit, the new byte overwrites the old data in *SBUF*. Unlike the PC's serial port, there are no error flags to indicate that this has happened, so the firmware must ensure that overrun cannot occur under any circumstances or detect the inevitable data loss by some other means.

Notice that there are two *SBUF* registers with two different functions. When the firmware writes data into the left-hand *SBUF* register, it is loading a new byte into the transmitter shift register. When it reads the right-hand *SBUF* register, it is getting the latest received byte from the holding register. Both *SBUF* registers have the same 8051 Special Function Register address and there is no way to read back the transmitted byte or change the received byte.

Because the 75176 transceiver is echoing whatever appears at P3.1 back to P3.0, the receiver is getting an exact copy of the transmitter data. The receiver will be active whenever there is network traffic, while the transmitter is active only when the RTC52 board is sending data. In that case, both the transmitter and receiver will be processing data simultaneously and the firmware must handle the outbound and inbound messages at once.

With all that in mind, Figure 2 shows the sequence of events when the firmware sends out a ten-byte message. The MC-Net firmware is completely interrupt driven, so *TI* and *RI* invoke interrupt handlers that decide what to do next. The two handlers are at the same priority level (in fact, as shown in Figure 1, they

share a common interrupt bit and vector), so each must run to completion before the other can take control of the processor.

Although you might think that the receiver would return the same byte the transmitter just finished sending, a close look at Figure 2 reveals that the transmitter sends two bytes before the first *RI* occurs. Remember that the *TI* flag goes active at the beginning of the stop bit for each outgoing byte, while *RI* goes on at the middle of the received stop bit.

In order to simulate the transceiver hardware, the firmware must record outgoing bytes as they are deposited into the *SBUF* transmitter register, then substitute them for the bytes that normally come from the *SBUF* holding register. The Avocet simulator handles transmitter timing by turning *TI* on after the correct number of machine cycles required to send nine serial bits, but it does not mimic the actual bits at the P3.1 output pin.

There are a number of ways to store the bytes, but I settled on a traditional ring buffer. Because the Avocet simulator displays eight bytes in each line of its "dump" windows, the ring can hold up to eight bytes, even though only two are in use at any one time. Figure 3 shows the structure of the *ECHOQUEUE* buffer and the two pointers used to insert and remove bytes from the ring.

*ECHOQUEUE* starts out empty with both *ECHOHEAD* and *ECHOTAIL* pointing to the first byte in the ring. The transmitter firmware adds bytes to the ring at the *ECHOHEAD* location and increments *ECHOHEAD*. It also sets *RI* when the ring has two or more bytes. The receiver firmware activated by *RI* removes the byte at *ECHOTAIL* and ticks that pointer. Both pointers wrap from the last ring address to the first as you would expect, so messages don't have to start at the first byte in *ECHOQUEUE*.

Listing 4 shows the additional transmitter code needed to implement *ECHOQUEUE* and Listing 5 show the corresponding receiver code. Remember that the receiver cannot interrupt the transmitter code, so setting the *RI* flag forces the receiver interrupt code

Menu-driven software to monitor, display, and control your home or production system on site or from a Remote location.

### uControl Features

- Display:
    - up to 16 analog inputs
    - up to 32 discrete inputs
    - up to 32 discrete outputs
  - Sample Rate:
    - update all within 1 sec.
  - Alarms
    - Switch Discrete Output on:
      - analog threshold
      - 'trip' of discrete line
  - Password Protection
    - 4 Priorities
  - MS-Windows based display customization program.
- Price: \$175.

### uControl-Remote ADDS

- Dial up from remote to access all features
  - Automatic dial-out on Fault Condition
- Price: \$95. (requires uCONTROL)

### uControl-History ADDS

- \* Historical Plotting of any input or output vs. time.
- Price: \$95. (requires uCONTROL)

### Requirements:

**uControl.** IBM PC/XT/AT (compatible) system with 512k memory; compatible data acquisition card (inquire about boards and systems supported.) Microsoft Windows Release 2.1 or later required to run configuration program. -Remote. also Hayes compatible modems at each end.

Commands: operator Logon immediate Print pick Transmtr Out  
 Logged on with Priority 4  
 Commands: Immediate mode Calibrations set autoPrint Save Data Logoff  
 Transmitter: Console Keyboard in Control

Analog Input		A Digital Output		A Digital Input	
Temp Out	2048.00	Sprinklers	ON	Front Door	CLOSE
Temp 2nd Fl	2048.00	Burglar Alarm	OFF	Window Interior	TRIP
Temp Burnt	2048.00	Digital Out 2	ON	Digital In 2	OFF
Analog In 3	2048.00	Digital Out 3	ON	Digital In 3	OFF
Analog In 4	2048.00	Digital Out 4	ON	Digital In 4	OFF
Analog In 5	2048.00	Digital Out 5	ON	Digital In 5	OFF
Analog In 6	2048.00	Digital Out 6	ON	Digital In 6	OFF
Analog In 7	2048.00	Digital Out 7	OFF	Digital In 7	OFF
Analog In 8	2048.00	Digital Out 8	ON	Digital In 8	OFF
Analog In 9	2048.00	Digital Out 9	ON	Digital In 9	OFF
Analog Input	2048.00	Digital Out 10	ON	Digital In 10	OFF
Analog Input	2048.00	Digital Out 11	ON	Digital In 11	OFF
Analog Input	2048.00	Digital Out 12	ON	Digital In 12	OFF
Analog Input	2048.00	Digital Out 13	ON	Digital In 13	OFF
Analog Input	2048.00	Digital Out 14	ON	Digital In 14	OFF
Analog Input	2048.00	Digital Out 15	ON	Digital In 15	OFF

10 Jul 1988 8:28:13

30-day money-back guarantee

(617) 861-0181

FAX (617) 861-1850

**Unkel Software Inc.** 

62 Bridge Street, Lexington, MA 02173

Reader Service #157

to run immediately after the *TI* handler.

The code shown will work correctly as long as the transmitter is running, but after the last byte of a message there will be no *TIs* to trigger the *RIs* needed to drain the ring buffer. The code that disables the transmitter after the last outgoing byte also sets the *RI* flag to get that final byte out of the queue.

#### WHY A RING?

If you work through the code and figures, you may conclude that a ring buffer is really overkill for a two-byte queue. True enough, but there is a lesson to be learned here.

There is always a tradeoff between "quick-and-dirty" code and a more elegant solution. The former may leave you with several obscure lines of code that do something that made sense at the time, while the latter is easier to understand after a few weeks (days?) because it isn't quite so tricky.

I've used ring buffers in several applications on a variety of processors, so I don't have to puzzle out how to make one work anymore. There are only four rituals: add one byte, remove one byte, check for underflow, and check for overflow. In this case, the last check isn't needed because "it can't happen here." This beats having to figure out and debug a special solution for every situation.

Also, you can modify the ring buffer to handle longer queues and multiple sources and sinks very easily. While those problems didn't crop up here, they have in other projects!

Moral of the story: don't always go for the shortest solution if you value your time. Check your toolkit for tried and true tools before you create a new one from scratch. Then make sure it works before you make it shorter, faster, or more obscure!

One word of warning if you are adapting this code to the PC. Because both the transmitter and receiver in the 8250 chip (and its successors in the AT and PS/2 machines) are double-buffered, there can be three bytes stored in the hardware: one in the

transmitter holding register, one split between the two shift registers, and the third in the receiver holding register. You must set a flag in the transmitter termination routine to tell the receiver to drain the remaining bytes on its own, because there will be more than one byte left in the ring.

#### THE LITTLE BUG THAT WASN'T THERE

Depending on your project's hardware development schedule, you may find another use for the firmware that got your code running in the simulator: substituting for defective or missing hardware in the first few "real" boards.

If you dedicate separate bits (or bytes, depending on your processor) to control separate functions, rather than using a single variable as I did with the *SimMode* bit, you can turn individual features on and off at will. Because the code really doesn't care whether it's running on the hardware or in the simulator, you can flip the bits on to bypass real hardware that is having a bad day.

You may need to add a simple user interface to twiddle the bits while the code is running or you can compile a special version that forces a specific bit on during the setup routine. Remember to keep things straight, because there is nothing more embarrassing than asking the hardware guys to shoot a bug when your code bypasses it completely!

In any event, you won't have the excuse "But the hardware isn't ready yet" anymore.

Sorry 'bout that...+

*Ed Nisley* is a member of the Circuit Cellar INK engineering staff and enjoys making gizmos do strange and wondrous things. He is, by turns, a beekeeper, bicyclist, Registered Professional Engineer, and amateur raconteur.

#### IRS

2 16 Very Useful  
2 17 Moderately Useful  
2 18 Not Useful

## A MESSAGE TO SUBSCRIBERS

Circuit Cellar INK occasionally allows companies which have products or services of interest to our readers to conduct a mailing to our subscriber list. If you do not wish to receive information from these companies, your name can be removed from the list which we supply to them.

To have your name removed from the outside mailing list, send a written request to:

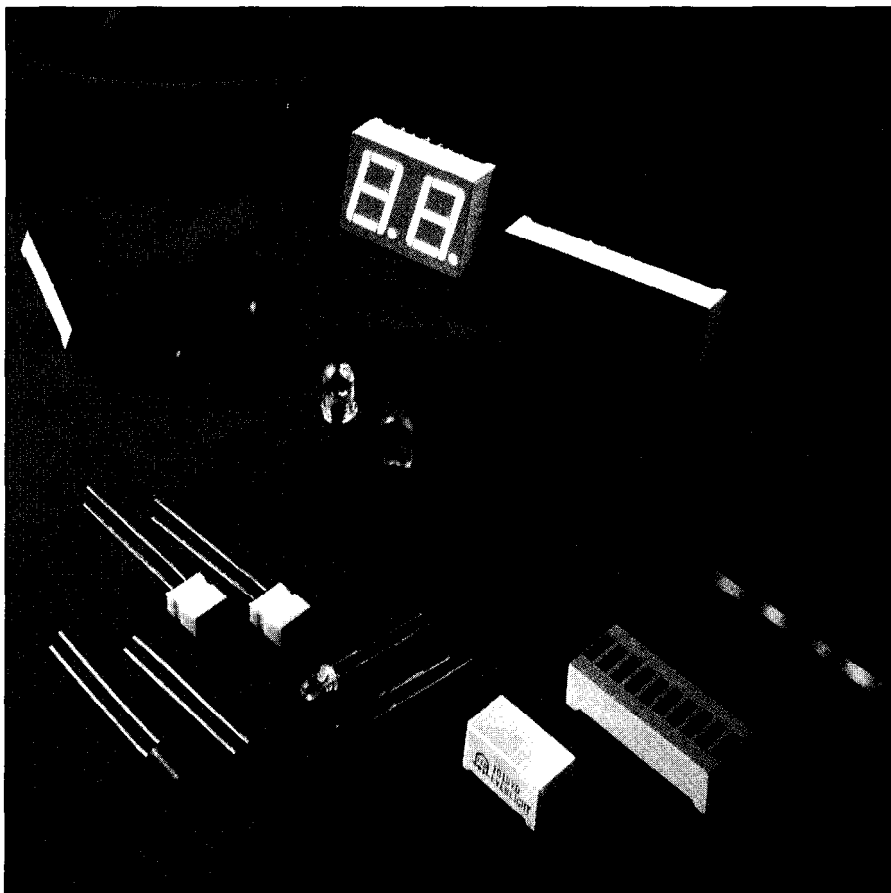
**CIRCUIT CELLAR INK  
Subscriber Service  
P. O. Box 2099  
Mahopac, NY  
01541**

# The Versatile Light-Emitting Diode

FROM  
THE  
BENCH

Jeff Bachiochi

Take a look around your environment today and notice the many uses for LEDs that we all take for granted. The first thing that we see in the morning, upon being startled out of a sound sleep, is the face of our attacker,



Four seven-segment digits daring opposition to our all-too-mechanical daily routine. Out of bed and into the shower. Feeling a bit renewed, it's a quick pause at the scale. "Oh no, those segments can't be right! Must be the battery." Off to the kitchen now for some, "Ah, smells good," coffee. An LED tattles its readiness. "Where's the TV remote? Here it is. Let's see what's on CNN this morning." The LED blinks wildly on the remote control as the TV bursts into life. The rechargeable phone, the microwave, the VCR, and almost every appliance today has an LED to indicate something. Power on, alarm, time, channel-the LED is our link to the electrical world.

LEDs offer a distinct advantage over incandescent lamps and neon bulbs. They are small in size and weight,

mechanically rugged, and operate at low voltages and currents. They have the same projected operating life as other solid-state devices: about 100,000 hours. Improvements in intensity over the years have made LEDs suitable for backlighting displays and push buttons. Status indication, whether a single LED, a row, a column, or a matrix (e.g., 5x7 array) is the largest use for LEDs. Your stereo receiver is a good example of the many configurations LEDs are used in. A single LED shows "power on" and twin bar graphs (two rows of LEDs) indicate the loudness of each channel. A digital display (seven LED-segment digits) announces the station's operating frequency and a position indicator (a column of LEDs) shows the signal strength.

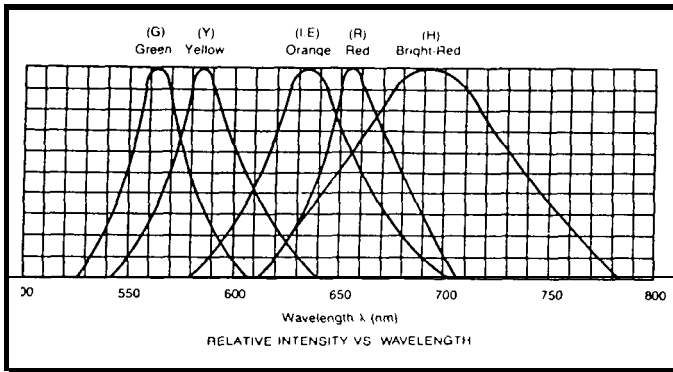


Figure 1—Human vision receives a relatively narrow spectrum of radiation between 400 and 900 nanometers. Currently available LEDs offer colors in fairly narrow ranges of the visible spectrum.

## SOME THEORY

An LED consists of a PN junction using materials which emit photons at particular energy levels when excited. When an electrical bias is applied, the recombination of electrons and holes is "radiative." The radiative photon energy is at a wavelength (in nanometers) equal to  $1240/DE$ , where DE is the approximate energy transition of the band-gap (PN junction) in electron volts for direct-transition-type materials. For example, Ga As with a band-gap energy  $E=1.43$  eV will emit at a wavelength of about 910 nm. Our eyes are sensitive to the color spectrum between 400 nm and 700 nm, so the example, which is in the infrared region, would be out of our range. That is why we cannot see the TV remote control, which uses an IR (infrared) LED to send commands to the TV. Many companies put a visible LED on the remote just to let the user know it is working. Visible LEDs are made possible using material combinations with a narrow band-gap potential. The newly developed blue LED will fill the remaining gap in the color spectrum made up of the popular red, yellow, orange, and green LEDs (see Figure 1).

## LET THERE BE LIGHT

When an LED's anode is at greater potential than the cathode (at least  $V_F$ ), it will conduct current. Current limiting of some type is needed to ensure that the maximum peak current of power is not exceeded, since doing so will cause permanent damage to the LED.

TTL devices can sink considerably more current than they can source. Therefore, if a logic device directly drives an LED (and current-limiting resistor), the device is normally configured so a low output drives the LED. Figure 2 shows several such configurations.

The following shows the guaranteed  $I_{OL}$  for similar devices in several logic families:

74-16 mA	74S—20 mA	74LS-8 mA
74L—3.6 mA	74H—20 mA	74C—3.6 mA
74AC—24 mA	74HC—4 mA	74HCT-8 mA
74AS—20 mA	74ALS-8 mA	74F—20 mA

Use a transistor to drive LEDs when more current is needed through the LED.

A single LED can indicate more than an on/off condition. Flashing the LED at different rates can indicate an analog-type output. A metronome is an example of how a flashing LED is used to indicate an analog value—in this case tempo. When we need to express more data than a single LED can convey, other methods are used. The most common is the seven-segment display. By mounting single bar-shaped diodes in the configuration of a number eight (as shown in Figure 3a), the digits zero through nine can be displayed by turning on the correct combination of diodes (sometimes including A-F to cover hexadecimal digits).

Many devices will include a decimal point or a colon to help indicate money, time, or a fractional value. In this type of package, one end of each LED is tied to a common lead. Both common-anode and common-cathode packages are available, reducing the number of pins needed by almost half.

A slight improvement on the seven-segment display adds seven additional segments. This adds the capability of displaying all the letters of the alphabet in addition to the numerals, and requires fourteen control lines (Figure 3b).

The fourteen-segment configuration is practically a matrix. A matrix, or an array of rows and columns, is the basis of a matrix printer. A 5x8 (5-column by 8-row) matrix character is printed one column at a time. A column of print hammers, with each hammer corresponding to a bit, raps the paper when enabled, leaving a line of dots on the paper. Five successive print head shifts and column prints form a character.

Figure 3c shows how a 5x8 matrix can be used to display alphanumeric information. At one control line per

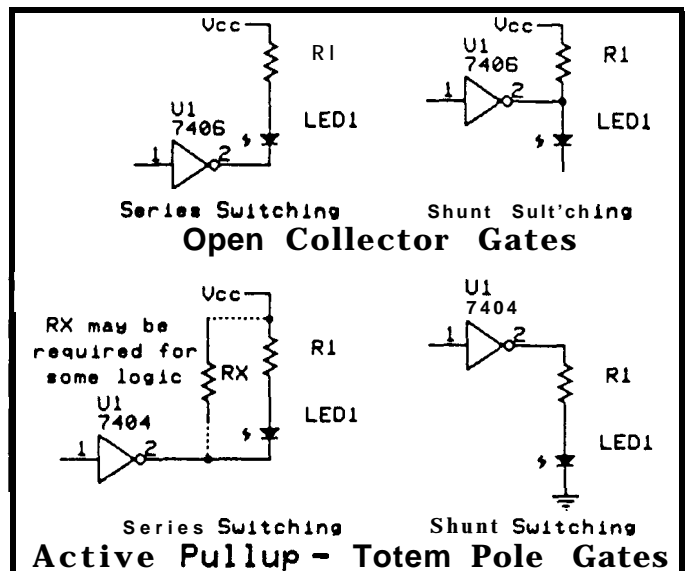
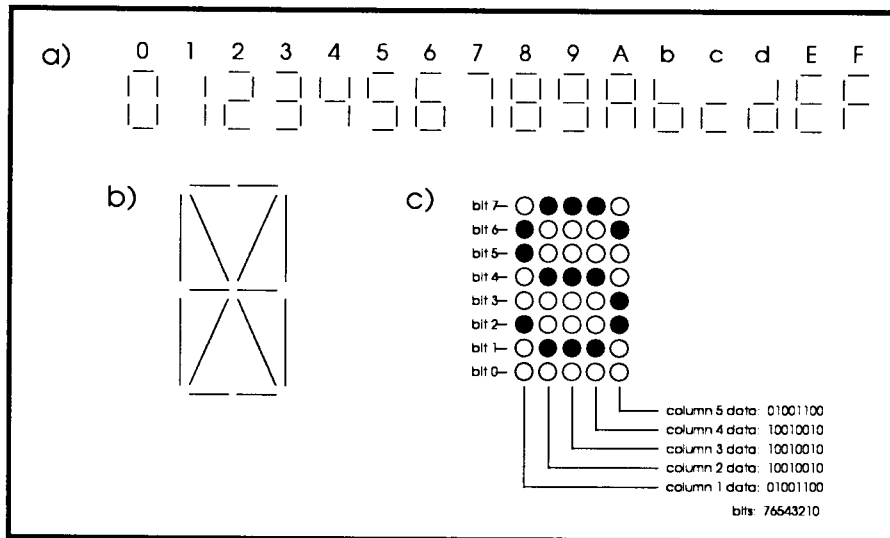


Figure 2—TTL logic devices can sink considerably more current than they can source. Therefore, if a logic device directly drives an LED (and current-limiting resistor), its output is normally configured so a low output drives the LED.



**Figure 3**—a) By mounting single bar-shaped diodes in the configuration of a number eight, the digits 0-9 and characters A-F can be displayed by turning on the correct combination of diodes. b) Adding seven more segments to a seven-segment display provides the capability of displaying all letters of the alphabet in addition to the numerals. c) A 5x8 matrix can be used to display alphanumeric information.

LED, that's 40 control lines, more than most systems have available. Learning from the printer, we could reduce the 40 down to 13 by using eight lines to carry the data bits and five lines to selectively enable the columns. Limiting the LED current to an amount a TTL gate can sink, we can produce a design using a '374 to latch each column. Each column's data is set up on the eight data bits and strobed in with one of the five column-control lines. This arrangement has each selected LED latched on for a 100% duty cycle (Figure 4).

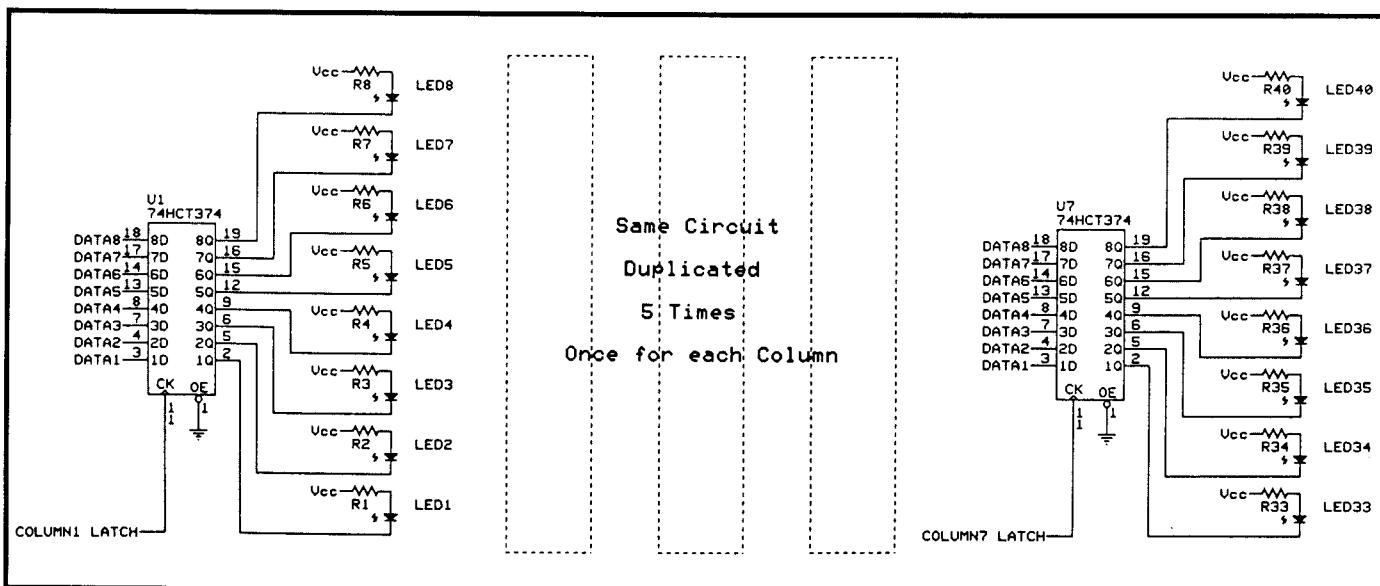
Adding high-current drivers to each LED increases the brightness of each LED, but also increases the parts count by 40 drivers.

Consider the way a TV picture is painted on the screen: dot by dot, row by row. We do not see the dots being painted because the phosphor and your eye "hold" an image. The ability of your eyes to retain images is called persistence of vision. Since your eyes can't adjust as rapidly as the images change on the screen, the images

seem to flow together and are viewed as complete pictures and not dots. "So what does this have to do with LEDs?" you ask?

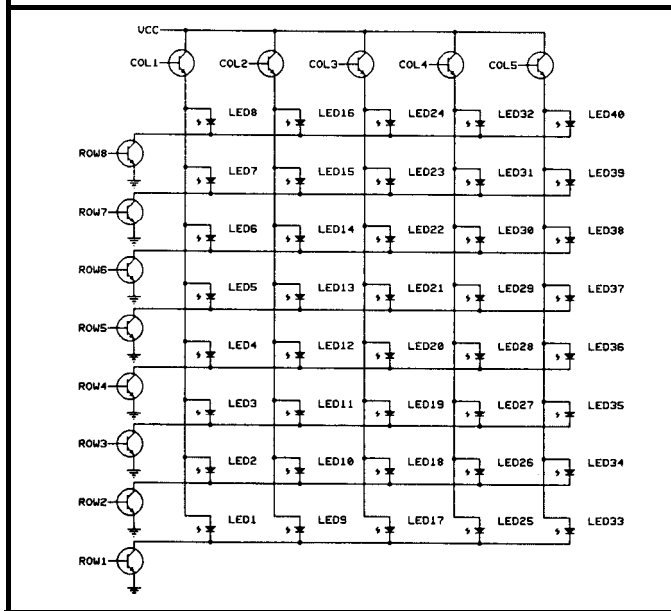
It is this persistence of vision that lets us make the next advance. Multiplexing is a method of cramming many channels of data onto a single line by timesharing. Each channel uses the single line for 1/N of the time, where N is the number of channels being multiplexed. In order for all the information to get through the single line, the multiplexer must cycle through all the channels at least as fast as the fastest data on any one channel. Demultiplexing is the stripping of each channel back out of the single line.

Referring back to Figure 4, let's remove the '374s and add eight drivers, one to each row of the first column. Tie every LED's cathode together within the same row. Tie every LED's anode together within the same column. Then drive each column with its own driver. This configuration—the columns connected to the anodes—is designated a column-anode matrix and is shown in Figure 5. If



**Figure 4**—A circuit for producing an LED matrix display. The 74HCT374 latches each column. In this circuit, each selected LED is latched on for a 100% duty cycle.

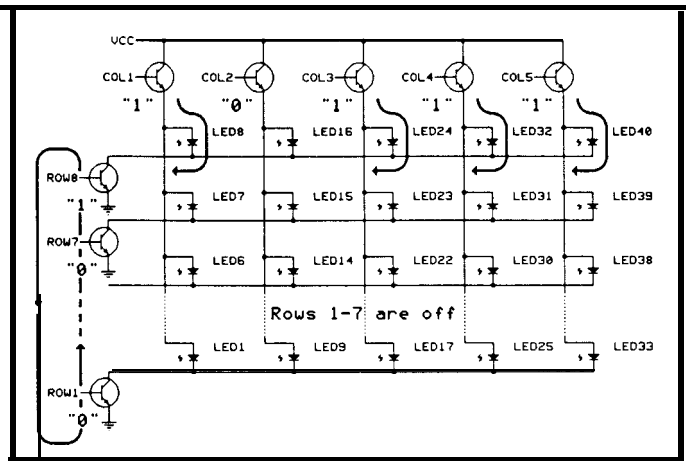




**Figure 5**—A variation on the circuit shown in Figure 4. The 74HC137s have been removed and replaced with a dedicated driver for each row. The result is designated either a column-anode matrix or a column-cathode matrix, depending on the orientation of the driver connections.

the matrix is produced with columns connecting the cathodes, it is a column-cathode device.

The above arrangement can be multiplexed by row, by column, or by both.



**Figure 6**—An expansion of the design shown in Figure 5. As each row is sequentially enabled, column data is displayed.

When multiplexed by row, 5-bit data applied to the columns will create source current for any row enabled (see Figure 6). More than one row should not be enabled at the same time or multiple rows of LEDs will light with the same data. By enabling the rows sequentially while the corresponding data is applied to the columns, all 40 LEDs will be addressed. If all seven rows can be sequentially scanned fast enough, our persistence of vision will create the illusion that all the rows are on at the same time, while in reality only one row is on at any given time. Current-limiting resistors must be placed between the five columns

## Cross-16 V2.0 Meta Assembler

Table based absolute cross-assembler using the manufacturer's assembly mnemonics.

Includes manual and MS-DOS assembler disk with tables for *all* of the following processors:

1802	64180	65C02	6801
6805	6809	68HC11	COP400
COP800	8048	8051	8085
8096	320C1X	TMS370	SUPER8
Z8	Z80	Z180	MORE...

Users can create tables for other processors!

Generates listing, symbol table and binary, Intel, or Motorola hexcode.

Free worldwide airmail shipping and handling.

Check, Money Order or P.O. **US\$99.00**  
VISA, Mastercard and Canada **CN\$119.00**

Universal Cross-Assemblers  
POB 6158, Saint John, NB  
Canada E2L 4R6  
Voice/Fax: (506)847-0681

## GIVE A GIFT OF TECHNOLOGY

Circuit Cellar INK Gift Subscriptions are available for the technologically inquisitive people in your life. You don't have to fumble with wrapping paper or fight holiday crowds, just give us a call. We'll make sure your lucky gift recipient has a full one (or two) year subscription to Circuit Cellar INK, along with a gift card announcing the giver's name and starting issue.

To make sure the gift card arrives before the holidays, call **203/875-2199** before **December 15.**

**CIRCUIT CELLAR INK**  
**THE GIFT OF TECHNOLOGY**

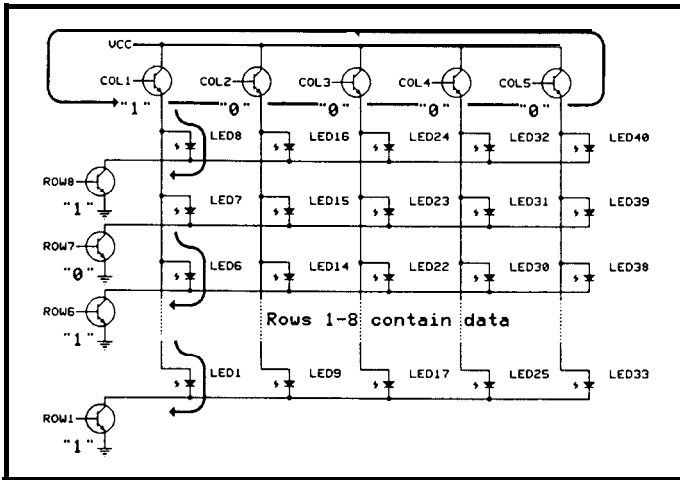


Figure 7—Still more based on Figure 5. As the columns are sequentially enabled, row data is displayed.

and the source drivers so that current through each LED can be individually controlled. If the resistors were placed between the rows and the sink drivers, the current would be divided up between all the column LEDs which are enabled. The more LEDs enabled, the lower the intensity of light from each will be.

When multiplexed by column, 8-bit data applied to the rows creates a current sink for any column enabled by the column source drivers (Figure 7). No more than one column should be enabled in this sequential form. Current-limiting resistors are placed between the rows and the sink drivers.

When multiplexed by both row and column, the rows are sequentially enabled. During the time each row is enabled, the columns are sequentially enabled. With this

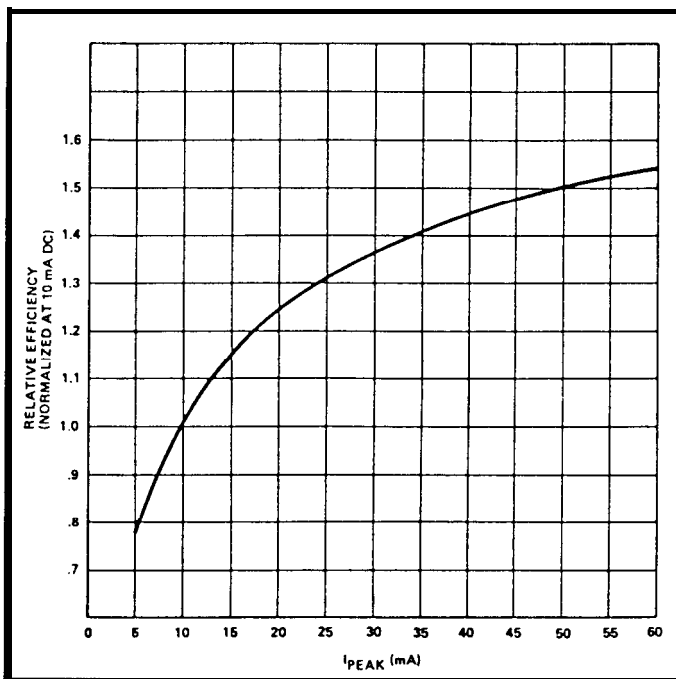


Figure 8—Efficiency goes up in relation to the current. This is shown as the relative efficiency normalized to 1.0 @ 10 mA.

form, only one LED can be enabled at a time, so the current-limiting resistors can go on either the columns or the rows.

In the multiplexed row and column modes, the duty cycle for any row or column will be  $1/N$ , where  $N$  is the number of rows or columns multiplexed ( $1/8$  for rows and  $1/5$  for columns). In the combination of rows and columns, the duty cycle for any one LED will be  $1/(R*C)$  or  $1/40$ . Assuming an LED's intensity is linearly proportional to the current flowing through it (though, in reality, it isn't), the current driving an LED that is on 12% of the time must be eight times that driving an LED which is on 100% of the time. In the combination multiplexed system where the duty cycle is  $1/40$ , an LED would need 40 times the current required for a single, 100% duty-cycle LED. Maximum peak currents are roughly five times the maximum average current, so without exceeding the maximum peak current the device, short duty cycles will produce dimly lit LEDs.

### THE MATH

LEDs made from the same junction materials can be packaged to produce different output effects. Depending on the lens arrangement, the light emitted can be focused to a rather narrow viewing angle or diffused for a wide viewing angle. The total light output from each device is the same, yet the nondiffused LED will appear to be brighter since its light is concentrated into a smaller viewing angle.

In an application where the LED is indicating a status in a latched (100% duty cycle) mode, the current-limiting resistor is figured as simply:

$$R = \frac{V_{CC} - V_F}{I_F}$$

where  $V_{CC}$  is the supply voltage,  $V_F$  is the LED forward voltage drop, and  $I_F$  is the LED forward current and is less than  $I_{Fmax}$ .

A standard T1 $\frac{3}{4}$  red LED operating at a typical current of 10 mA has a light output (luminous intensity) of about 1.1 mcd (millicandelas). The series resistor required with a 5-volt  $V_{CC}$  would be calculated as  $(5 - 1.7) / 0.01 = 330\Omega$ . In this case,  $I_{avg}$  equals  $I_F$ .

When used in a multiplexed application the formula is a bit more complicated:

$$R = \frac{V_{CC} - V_F}{I_{PEAK}}$$

It is the peak current that is the mystery. If we try to maintain the 1.1-mcd output of the nonmultiplexed example, we use the following formula to find  $I_{avg}$ :

$$I_{PEAK} * \eta_{PEAK} = \frac{I_F * \eta_F}{TCYC}$$

where  $\eta$  is the efficiency of the LED at the respective current and  $T_{Cyc}$  is the duty cycle.

Let's use a duty cycle of 1/5 (20%). Figure 8 shows how efficiency goes up in relation to the current, and is shown as the relative efficiency normalized to 1.0@10mA. Plugging in numbers,  $I_{PEAK} * \eta_{PEAK} = (10 \text{ mA} * 1.0) / 20\% = (0.01 * 1) 0.2 = 0.05$ . Using the graph, we find it takes about 35 mA at an efficiency of 1.4 to achieve the same luminous intensity as 10 mA at a 100% duty cycle. That is three and a half times the current for one fifth the time.

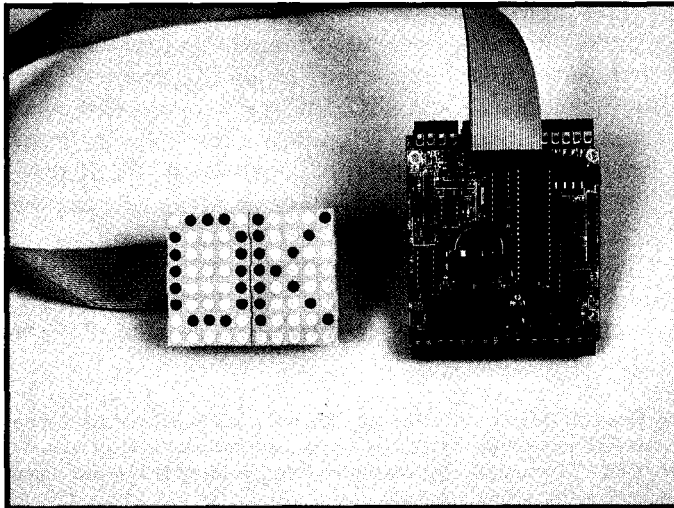
The same calculation for a duty-cycle of 1/40 is:

$$I_{PEAK} * \eta_{PEAK} = (10 \text{ mA} * 1.0) / 2.5\% = (0.01 * 1) / 0.025 = 0.4$$

Well, 400 mA is off the scale, but if we use an efficiency of 1.6 (about the maximum obtainable), the peak current equals about 250 mA. Since maximum allowable  $I_L$  is about 200 mA, we can't obtain the 1.1 mcd with a 1/40 duty-cycle.

In a multiplexed design, peak currents (or average currents, for that matter) are often limited by the drivers used in the circuit. Where individual transistors can be specially selected to provide adequate drive for any application, prepackaged drivers are a bit more conservative. Two popular prepackaged drivers are the 75491 and 75492.

The 75491 has four drivers per package and can source or sink 50 mA each. The 75492 has six drivers per package and can sink 250 mA each (600 mA total).



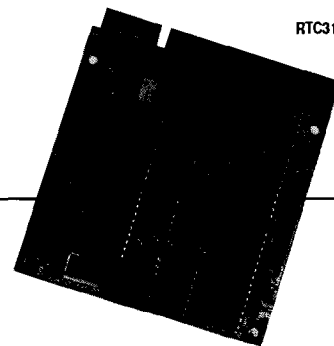
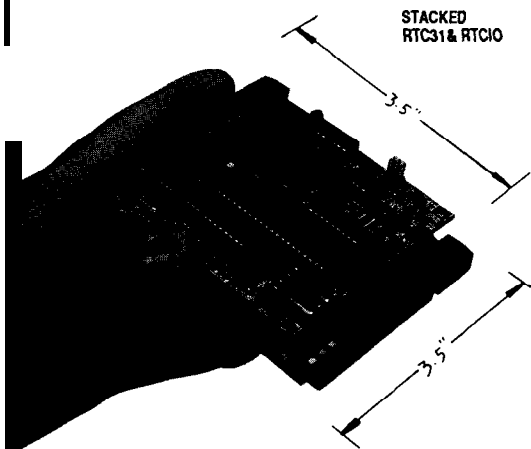
### LED MODULES

Using LEDs in an array helps to save in two ways: fewer components are necessary to drive the array and fewer actual connections are made between the drivers and the array. The circuit in Figure 9 demonstrates the ease of using LED arrays in a display application. The display is driven by a PPI (8255) which has 24 bits of I/O. The 8255 is part of the RTC52 system as presented in this column in CIRCUIT CELLAR INK #8.

## MICROMINT Introduces "Micro" Controlling!

After years of experience in manufacturing OEM controller boards and talking to customers, we think we have hit upon just the right combination of format and function to satisfy even the toughest case of "relay mentality." Realizing that not every computer/controller application warrants a Cray X/MP, Micromint offers a tiny 8031/8052-based controller board for those dedicated and cost-sensitive installations.

**New MC-NET™** software links your desktop up to 31 RTC controllers.



#### RTC31 and RTC52 Technical Specifications

- 8031 processor (RTC31) or Micromint 80C52-BASIC processor (RTC52)
- 11.05-MHz system clock
- Uses 8K or 32K memory chips
- Up to 64K bytes of RAM or EPROM
- 5-volt-only operation
- 110-19200 bps RS-232 and/or RS-485 serial port
- Use stand-alone or networked
- 12 bits of parallel I/O
- Vertical-stacking expansion bus
- Screw terminal connections
- Small 3.5"x3.5" format
- 80 mA typical operating current (RTC52)

RTC31-1	8031 Controller	\$119.00
RTC31-1	OEM 100-Quantity Price	\$79.00
RTC52-1	80C52 Controller	\$139.00
RTC52-1	OEM 100-Quantity Price	\$99.00

#### RTCIO Technical Specifications

- Three bidirectional parallel ports (24 bits)
- 8-channel, 8-bit A/D (C-V); 9,000 samples/sec
- 4-channel, 8-bit D/A (0-5V); 2- $\mu$ s response time
- Battery-backed clock/calendar and presettable time-interrupted capability
- DC to DC conversion-5-volt-only operation
- Screw terminal connections
- Small 3.5"x3.5" format

RTCIO	RTCIO board with parallel I/O and A/D converter	\$129.00
RTCIO	OEM 100-Quantity Price	\$89.00



**Micromint, Inc.**

4 Park Street, Vernon, Connecticut 06066  
Tel: (203) 871-6170 • Fax: (203) 872-2204

Available Soon!

ATC-OPT0 8-channel Optoisolated I/O Expansion Board (expected availability December, 1989)  
RTC-SIR Serial, Timer, and Infrared I/O Expansion Board (expected availability January, 1990)  
RTC-LCD LCD, Keyboard, and X-10 I/O Expansion Board (expected availability February, 1990)

This simple matrix uses two 5x8 LED display modules to create a 10x8 display. This arrangement could be used to indicate time or temperature as an alphanumeric display; as status indication of 80 individual processes; or as bar graphs displaying A/D inputs in either column or row format. By placing an overlay of your home on the display, LEDs can indicate open windows or doors or even the presence of moving objects in each **room**.

Next time, I'll discuss a bit of software used to activate the rows and columns. Also, how the hardware might be expanded to drive more modules using the same 8255. ❖

*Jeff Bachiochi (pronounced "BAH-key-AH-key") is a member of the Circuit Cellar INK engineering staff. His background includes work in both the electronic engineering and manufacturing fields. In his spare time, Jeff enjoys his family, windsurfing, and pizza.*

## REFERENCES

Optoelectronics Applications Manual  
Hewlett-Packard  
McGraw-Hill

Opto Electronic Semiconductors  
Toshiba America, Inc.

Opto Electronic Components  
GLOITE Sales Ltd.

Interface Circuits Data Book  
Texas Instruments

## IRS

2 19 Very Useful  
220 Moderately Useful  
221 Not Useful

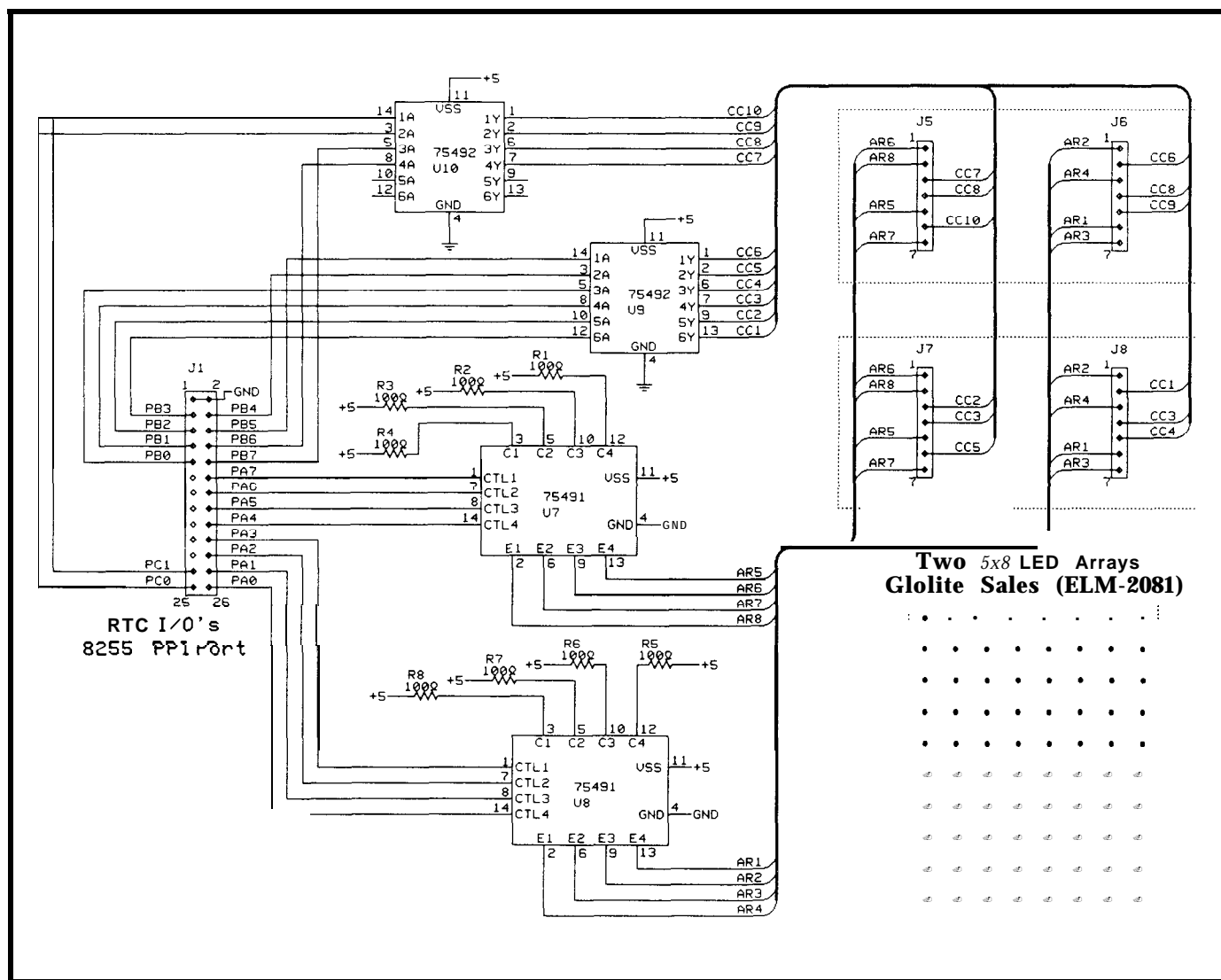


Figure 9-LED arrays are straightforward to use in most applications that have several parallel ports and a bit of processing power.

# MAXIMUM Chips

Maximum Real- World Problem Solvers

## SILICON UPDATE

Tom Cantrell

Intel says the forthcoming '586 will have four-million transistors! Without adding them up, I guess this is probably **more** transistors on a single chip than the combined total of all the chips (CPU, RAM, I/O) in an AT-compatible computer. Not to say that the '586 is a PC on a chip, but I can definitely see a time in the future when a total PC is reduced to a handful of ICs.

But, no matter how highly integrated the "big" chips get, I predict a healthy market will remain for "little" chips which fill mundane, but necessary, roles. Largely, these "little" chips serve to connect the finicky "big" chips, which require a pristine atmosphere of 1s and 0s, to the ugly real world. Examples include a myriad of analog functions (converters, filters, switches, and muxes), power supply circuits, and data communications interfaces. Even tomorrow's PC on a chip will have to be supplemented with a lowly RS-232 transceiver!

Enter Maxim (Sunnyvale, Calif.), a company which eschews the glamour and hype of "big" chips in favor of solving those nagging little problems which plague even the most elegant designs. While others concentrate on tomorrow's "Strategic Rodent Defense Initiative," Maxim concentrates on better mousetraps.

Maxim got its start in 1983 by second-sourcing chips from established analog suppliers like Analog Devices and Intersil. Now, while continuing to second-source, their product portfolio has expanded with a wide variety of proprietary products.

The Maxim products of most interest to microprocessor-based system designers fit into three groups: analog data conversion, RS-232 interfaces, and power supply circuits.

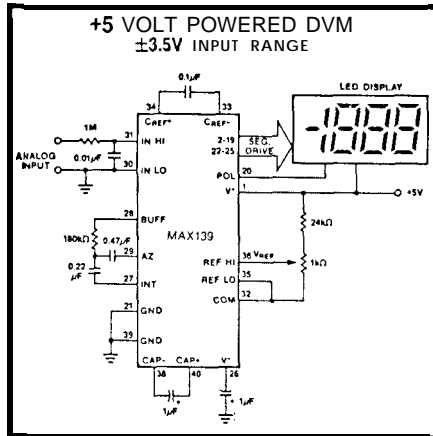


Figure 1 - Integrating A/D converters, such as the MAX 139, are suitable for use in stand-alone applications that require both A/D conversion and a built-in display.

### IT'S AN ANALOG WORLD

The base for Maxim's growth has been analog data converters—both analog to digital (A/D) and digital to analog (D/A). Maxim offers dozens of parts; many of them second-sources of existing devices. More interesting are Maxim's new, proprietary designs.

The new A/D offerings fit into three categories, according to their conversion technique.. .

*Integrating* - These are relatively slow (>20 ms) converters which include built-in display (LED or LCD) drivers. They are best suited for stand-alone applications such as a hand-held instrument or dedicated process monitor/display. Figure 1 shows the guts of such a design based on the MAX139.

*Successive Approximation (SAR)* - Maxim's implementation of this popular technique achieves good speed (3.25-10 µs) and resolution (8- or 12-bit) at reasonable cost (Figure 2). The MAX160, '161, and '162 are faster equivalents to existing chips while the '163, '164, and '167 integrate input sample/hold to reduce noise at high sampling rates.

*Half-Flash* - As the name implies, these converters are based on a scheme that combines the speed of flash conversion with the cost effectiveness of successive approxima-

SAR A/D Converters							
Part Number	Resolution	Integral Linearity	Conversion Time	Supply Voltage	Input Range	Features	Reference
MAX160	8 bits	1/2 LSB	4µs	+5V	±15V	Fast AD7574	
MAX161	8 bits/8ch	1/2 LSB	20µs	+5V	±15V	Fast AD758 1 Dual port RAM	External
MAX162	12 bits	1/2 LSB	3.25µs	+5V/-12V	+5V	Fast AD7572	Internal
MAX163	12 bits	1/2 LSB	7µs	+5V/-12V	+5V	Sample/Hold	Internal
MAX164	12 bits	1/2 LSB	7µs	+5V/-12V	±5V	Sample/Hold	Internal
MAX167	12 bits	1/2 LSB	7µs	+5V/-12V	±2.5V	Sample/Hold	Internal
MAX172	12 bits	1/2 LSB	10µs	+5V/-12V	+5V	Low cost	Internal

Figure 2-The chips in the MAX16x series use Successive Approximation to combine speed, 8- to 12-bit resolution, and reasonable cost.

### Half-Flash A/D Converters

Part Number	Resolution	Integral Linearity	Conversion Time	Supply Voltage	Input Range	Features	Reference
MAX 150	8 bits	1/2LSB	1.34µs	+5V	+5V	Track/Hold	Internal
MAX154	8 bits/4 ch	1/2LSB	2µs	+5V	+5V	Track/Hold	Internal
MAX158	8 bits/4 ch	1/2LSB	2µs	+5V	+5V	Track/Hold	Internal

Figure 3—The MAX150 series is referred to as 'Half-Flash' since they combine Flash and Successive Approximation conversion techniques.

tion. The MAX150 series of 8-bit converters, shown in Figure 3, actually contain two 4-bit flash converters of the type shown in Figure 4. The first determines the upper four bits of the result. The upper four bits are then sent to a 4-bit DAC whose output is subtracted from the original analog source. The resulting signal is then fed to the second 4-bit flash converter to determine the lower four bits of the result. This cuts conversion time to 2.0 µs or less, which is pretty fast. Indeed, you'll need a pretty speedy micro (preferably with DMA) to keep up.

On the D/A front, Maxim has thrust beyond their second-source offerings with some neat new chips: the MAX500 and '543 (Figure 5). A feature appearing on more and more I&---serial. I/O-significantly reduces the number of micro I/O lines needed to connect the '500 and '543 compared to the 10-12 lines typically required of an 8-bit parallel chip.

DACs need a voltage reference, and often the system power supply can provide it. If a different voltage or more accuracy and temperature stability is required, Maxim offers a range of chips (including the MAX670 series) which

generate precise reference voltages (1.23V, 2.5V, 5V, or 10V) from an input voltage (typically 15V, though the MAX672 and '673 support an input range from 8-13V to 40V).

### SOMETHING NEW FOR '232

Surely even the casual experimenter has paid the RS-232 dues in the form of endless fiddling with cables and handshaking lines to try to get two "RS-232-compatible" (an oxymoron, if I ever heard one) devices to talk.

You RS-232 users don't suffer alone. The hardware designer who made the box you're having trouble with also shares your grief. The designer's problem is that RS-232 calls for signal levels outside the realm of the 0-5V digital world. There must be a zillion designs that could get by with a single 5V supply, but instead must use a more

## P-C-B ARTWORK MADE EASY !

Create Printed Circuit Artwork  
on your IBM or Compatible

- \* MENU DRIVEN
- \* HELP SCREENS
- \* ADVANCED FEATURES
- \* EXTREMELY USER FRIENDLY
- \* 1X and 2X PRINTER ARTWORK
- \* 1X HP LaserJet ARTWORK

\* HP and HI PLOTTER DRIVER optional 49.00

REQUIREMENTS: IBM PC or Compatible, 384K RAM  
DOS 3.0 or later.

PCBoards - layout program 99.00  
PCRoute - auto-router 99.00  
SuperCAD - schematic pgm. 99.00  
DEMO - 10.00

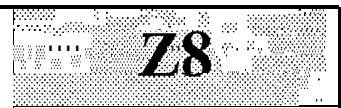
Call or write for more information

**PCBoards**

2110 14th Ave. South, Birmingham, AL 35205  
(205) 933-1122

Reader Service #145

## NEW WICEZ8 In-Circuit Emulator



WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC WYTEC

This NEW WICEZ8, containing the new **Z86C12** CMOS ICE chip, is a complete In-Circuit Emulator for the entire **Z8** microcontroller family, including the **86C91** and the **86C21** with 256 internal registers.

- Comes with window-oriented, user-friendly PC driver software for the IBM PC, XT, AT or compatibles and permits real-time emulation up to 20 MHz.
  - Unique display windows monitor 30 programmable memory locations or registers, 17 bytes of stack and is automatically updated when the user program is stopped or single-stepped.
  - Addresses up to 128K bytes of memory.
  - Provides 32K emulation RAM mappable in 2K, 4K, 8K, 16K, or 32K blocks.
  - Provides 8 hardware breakpoints which can always be displayed on the window.
  - Symbolic debugger reads symbol files in the 2500 A.D., Microtek and Zax formats directly.
  - On-line assembler, disassembler, memory/register exam, compare, fill, move, search and modification.
- 038.4K baud rate for fast upload/download of files in Tektronix Hex, Intel Hex and Motorola S record formats. (Download an 8K hex file in 5 seconds. )

Regular Price: \$1475 Introductory Price: \$995  
**30 day Money Back Guarantee**

**WYTEC**

(708) 894-1440  
**WYTEC COMPANY**

Suite 140, 185C E. Lake Street, Bloomingdale, IL60108

Reader Service #159

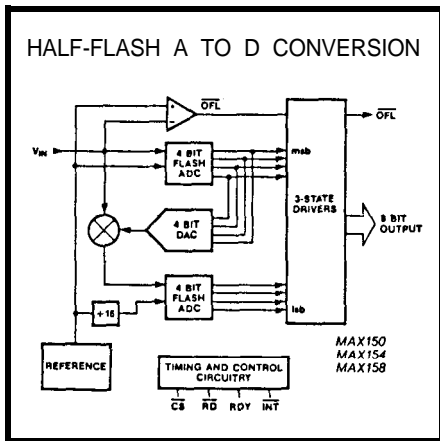


Figure 4—Two 4-bit converters are combined in the 8-bit MAX 150 series of A/D converters. Conversion time runs to as little as 2.0  $\mu$ s.

expensive three-output supply to feed the RS-232 transceiver.

I originally became aware of Maxim when they introduced the MAX232: an RS-232 transceiver which runs on

### Intel Introduces "Superscalar" 80960CA

In the last Silicon Update, I discussed the Intel 80960KA, 'KB, and 'MC. Since then, Intel has announced the newest member of the family: the 80960CA.

As described at a September 12, 1989 press conference, the 'CA is quite similar on the surface to the existing 'KA. Major functional upgrades from 'KA are the inclusion of a four-channel DMA controller, a fancier bus controller driving a newly demultiplexed bus, an improved interrupt controller, and 1.5K bytes of on-chip data RAM (supplementing the 1 KB code cache offered on all '960s). Notably, the new RAM supports a multiregister set banking scheme to speed context switches and subroutine calls.

Perhaps more important than the new functions are improved implementation of the core architecture for higher speed. You remember that the claim to fame of the earlier '960s was on-chip parallelism and bandwidth. Now, the 'CA pushes further in this direction.

According to Intel, the 'CA can execute three instructions in a single 33-ns clock cycle using the "scoreboarding" technique (described in the last Update) to avoid out-of-order-execution hazards. In marketese, Intel says "Superscalar" is the word to describe highly parallel multi-instruction-per-clock machines. The concept is also known as VLIW (Very Long Instruction Word) in other circles.

The key is to maximize bandwidth, which is essentially clock rate times bus width. Since the clock rate is limited by process and system design considerations, multiple and wide (the "L" in VLIW) buses are used. For instance, the '960CA internal data paths (code cache, data RAM, etc.) are 128 bits wide. With ten separate on-chip buses (608 bits in total) and a 33-MHz clock rate, the gross on-chip bandwidth is 2.5 GB/s! This is what really separates the '960CA from lesser competitors.

The 'CA also includes branch prediction to take advantage of the fact that conditional branches go the same way most of the time. The processor can start the branch even before the condition is evaluated. Usually (i.e., loop iteration), the "prediction" will be right, and the processor is ready to go as soon as the condition is evaluated. When the prediction is finally wrong (i.e., loop exit), only a small penalty (pipeline break) is paid.

Intel had an elaborate benchmark setup involving 'CA, 'KA, MC68030, and AMD29000SBCs to back their claims of "5x" faster and 30-66-100 (VAX-Native-Burst) MIPS. Forgetting the hype; I'll agree that the 80960CA is the hottest rod in town.

Until now, the '960 has mainly been confined to very-high-end applications (that tends to happen with \$200 chips) such as avionics and medical imaging. Now I hear aggressive pricing (especially on the 'KA) is winning commercial designs such as laser printers and LAN controllers.

The embedded-control-oriented '960 may not have the glamour of its PC ('486) and workstation ('860) siblings, but it is sure a nice chip.

+5V only. As shown in Figure 6, the additional voltages (+10V and -10V) are generated on the chip itself. What a lifesaver!

Now, the MAX200 series ('230-'252) includes dozens of offerings (Figure 7). Besides solving the basic voltage conversion problem, each chip in the series is tailored to serve specific applications. Need maximum channels in the smallest possible board space? Try the MAX235 or '245 (these chips don't require the fairly large external capacitors that the others do). Building a battery-powered design which takes advantage of the Maxim CMOS technology? Choose one of the chips with "shutdown" capability to reduce power consumption when the RS-232 port is not active. Want to harden your system against potentially hazardous voltages (something to consider in a factory or

D/A Converters				
Part Number	Type	Resolution	Setting Time ( $\mu$ s)	Power (V)
MAX500	Quad. Voltage Out. Serial In	8-bit	5.0	+12V to +15V
MAX543	Serial Input. 8-pin pkg	12-bit	1.0	+5V

Figure 5—The MAX150 and MAX543 have D/A and serial communications combined in a single chip to reduce the total number of I/O lines needed for an application.

outdoor setting)? Choose the MAX252 which features 1500V isolation.

Some systems may actually need a non-5V supply for something besides RS-232 (typically an A/D or D/A). If the MAX230 series doesn't kill off the market for those crazy three-output supplies (you know, 5V@100A,  $\pm$ 12V@10mA), perhaps some of the Maxim power supply circuits will.

### DIGITAL POWER

Even in the analog world, power supply design approaches being a "black art." Needless to say, digital designers stay away from this stuff and simply view the

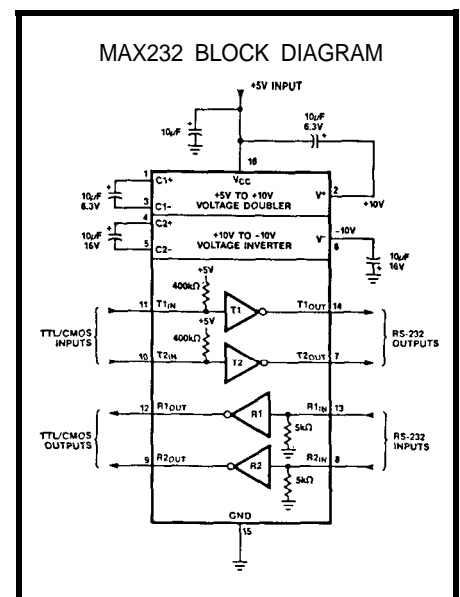


Figure 6—The MAX232 has been used in several projects presented in Circuit Cellar INK.

The MAX232 Family							
Part Number	Power Supply	No. of RS-232 Drivers	No. of RS-232 Receivers	Ext. Caps.	Shut-down	Receivers Output 3-State	Pins
MAX230	+5V	5	0	4	Y		20
MAX231	+5V and +7.5V to 13.2V	2	2	2	-		14
MAX232	+5V	2	2	4			16
MAX232A	+5V	2	2	4			16
MAX233	+5V	2	2	0	-		20
MAX234	+5V	4	0	4			16
MAX235		5	5	0	Y		24
MAX236	+5V	4	3	4	Y		24
MAX237	+5V	5	3	4			24
MAX238	+5V	4	4	4	-		24
MAX239	+5V and +7.5V to 13.2V	3	4	2	-		24
MAX240	+5V	5	5	4	Y		44
MAX241	+5V	4	5	4	Y		28
MAX242	+5V	2	2	4	Y	Y	18
MAX243	+5V	2	2	4	-		16
MAX244	+5V	8	10	4	-		44
MAX245	+5V	8	10	0	Y	Y	40
MAX246	+5V	8	10	0	Y	Y	40
MAX247	+5V	8	9	0	Y	Y	40
MAX248	+5V	8	8	4	Y		44
MAX249	+5V	6	10	4	Y		44
MAX250	+5V	2	2		Y	Y	14
MAX251	+5V	2	2		Y	Y	14
MAX252	+5V	2	2	0	Y	Y	40

Figure 7—The MAX200 series provides RS-232 channels with custom features for specific applications.

power supply as a connector on their schematic. This is OK when you can buy a power supply that meets your specs.

But what if you need something a little different: battery operation, memory back-up, or an extra voltage or two. Check out the MAX600 series of converters, regulators, and supervisory circuits.

Perhaps you want to power your hand-held micro-based instrument, which contains an A/D requiring -5V, from a 6V battery. First you need a MAX667 lowdropout voltage regulator. The 'low dropout' feature means the regulator will continue to provide +5V output even as the input drops (due to battery drain) as low as 5.12V. Normal regulators need at least a 1-2V difference between the

back-up mode and the microprocessor shut off in an orderly manner.

Enter the MAX690 series supervisory circuits (Figure 8). These chips handle all the housekeeping for power-related tasks including power-up reset, power-fail reset (keeps the micro from crashing), CMOS RAM protect/battery switchover, and watchdog timer.

As I said back in the beginning of this column, the glamour no doubt goes to the high-powered microprocessors. When it comes time to solve real-world problems, though, I'm glad that the engineers at Maxim have been hard at work on the little chips that help to hold it all together. ❖

Supervisory Power Supply Circuits								
μP Reset, Power Fail Detector, Battery Switchover, and Watchdog Timer								
Part Number	Pins	Reset Level (volts)	Supply Current (mA)	Reset Delay (ms)	Battery Switch	RAM Protect	Low In	Line Out
MAX690	8	4.65	4	50	Yes	No	No	No
MAX691	16	4.65	4	50	Yes	Yes	No	Yes
MAX692	8	4.40	4	50	Yes	No	No	No
MAX693	16	4.40	4	50	Yes	Yes	No	Yes
MAX694	8	4.65	4	200	Yes	No	No	No
MAX695	16	4.65	4	200	Yes	Yes	No	Yes
MAX696	16	Adj.	4	50	Yes	No	Yes	Yes
MAX697	16	Adj.	.16	50	No	Yes	Yes	Yes

Figure 8—If you are designing an application where power consumption and conservation are critical, the MAX690 series provides full housekeeping for power-related tasks.

input and output to maintain output regulation.

Now take the +5V from the '667 (up to 250 mA; enough for pretty sophisticated systems thanks to CMOS) and feed it to a MAX660 negative voltage converter. Voilà! Out comes the -5V for the A/D.

Don't overlook the fact that the Maxim power conversion chips can be used in different ways to serve a wide range of applications. For instance, a MAX641 can be used to generate +5V@40mA from a 1.5V battery. With different wiring and external components, the same '641 can generate +5V@0.5A from -48V commonly found in telecom applications. A MAX680, normally used to output ±10V from a +5V input, can be hooked up "in reverse" to extract 5V@5mA from RS-232 inputs—great for low-power RS-232 plug-ins like a mouse or joystick.

What if you want your instrument to include a CMOS SRAM to store data samples? This introduces a big problem: How to avoid trashing the SRAM data when the battery does fail. The memory must be put into lowest voltage/power

For more information:  
Maxim Integrated Products, Inc.  
120 San Gabriel Drive  
Sunnyvale, CA 94086  
(408) 737-7600

Tom Cantrell holds a B.A. in economics and an M.B.A. from UCLA. He owns and operates Microfuture Inc., and has been in Silicon Valley for 10 years involved in chip, board, and system design and marketing.

IRS

222 Very Useful  
223 Moderately Useful  
224 Not Useful



## ADVERTISER'S INDEX

Reader Service Number	Advertiser	Page Number
101	<b>Alpha Products</b>	18
102	<b>Andratech</b>	36
103	<b>Avocet</b>	C2
104	Berry Computers	7
105	BG Associates	57
106	Big Bang Software	33
107	Binary Technologies	81
108	Cabbage Cases	84
109	California Network Controls	87
110	Catenary Systems	29
.	Ciarcia Design Works	87
111	Circuit Cellar	61
112	Circuit Cellar	61
113	Circuit Cellar	87
114	Computerwise	6
115	Cottage Resources	26
116	Covox Inc.	48
117	<b>Dycor</b> Industrial	40
118	Electronic Energy Control	30
119	Engineers Collaborative	77
120	Express Circuits	36
121	F&W Communications	44
122	G-Tek	C4
123	<b>Gott</b> Electronics	11
124	Grammar Engine	85
125	Hazelwood	81
126	Innotec	80
127	Introl Corp	19
128	Kelvin	52
129	Kolod Research, Inc.	6
130	Laboratory Microsystems	76
131	Lear Com Company	87
132	Logical Systems	25
133	LS Electronics	87
134	<b>LTS/C Corp</b>	56
135	Meredith Instruments	87
136	Micro Resources	33
137	Micromint	C3
138	Micromint	17
139	Micromint	48
140	Micromint	67
141	Ming Engineering	77
142	NOHAU Corp	36
143	Paradigm Systems	76
144	<b>Parallax, Inc.</b>	54
145	PC Boards	70
146	<b>PseudoCorp</b>	51
147	R & D Electronics	84
148	Real Time Devices	31
.	Ryle Design	87
149	Silicon Alley	38
150	Skunk Creek	85
151	Soft Advances	13
152	Tardis Systems	48
153	Thinking Tools	80
154	<b>Timeline</b>	4
.	Tinney	25
155	Traxel Laboratories Inc.	25
156	Universal Cross-Assemblers	65
157	Unkel Software	59
158	<b>Witts</b> Associates	87
159	Wytech Co.	70

HAJAR ASSOCIATES  
NATIONAL ADVERTISING SALES  
REPRESENTATIVES

## NORTHEAST

Lisa D'Ambrosia  
Tel: (617) 769-8950  
Fax: (617) 769-8982

## MID-ATLANTIC

Barbara Best  
Tel: (201) 741-7744  
Fax: (201) 741-6823

## SOUTHEAST

Christa Collins  
Tel: (305) 966-3939  
Fax: (305) 985-8457

## MIDWEST

Nanette Traetow  
Tel: (708) 789-3080  
Fax: (708) 789-3082

## WEST COAST

Barbara Jones & Shelley Rainey  
Tel: (714) 540-3554  
Fax: (714) 540-7103

SUBSCRIPTION  
PROBLEMS?

If you have problems with your subscription (delayed or missing issues, change of address, or questions on renewals), call the Circuit Cellar INK Subscriber Service Line at (914) 628-0885 or write:

**Circuit Cellar INK**  
**Subscriber Service Dept.**  
**P.O. Box 2099**  
**Mahopac, NY 10541**

# Least-Squares Curve Fitting

SOFTWARE  
by DESIGN

Jack

Ganssle

In the August/September issue, I discussed using convolutions to smooth noisy real-world raw data. Judging by the amount of mail I've received, this subject seems to be of considerable interest to INK readers.

The method I presented was based on using convolutions over very small intervals to smooth and/or differentiate an input waveform. The technique of Least Squares was used to force the noisy signal to match a reasonable polynomial curve over the interval.

The technique of least squares is often used to fit experimental data to a polynomial. If you don't know the formula that relates a measured value to a desired output, it is often possible to "calibrate" the instrument by measuring many different samples of known values and then computing a polynomial that represents the relation between the input and output. Confusing? Perhaps, but this technique forms the basis of many instruments.

I once worked at a company that used different wavelengths of infrared light to measure the amount of protein in wheat. No "magic" formula exists to compute protein in this manner, but it turned out that over a small range (say, 11 to 19% protein), a linear relation exists between the measured light and percent protein. In other words, if  $f_1$ ,  $f_2$ , and  $f_3$  are responses at each of three frequencies:

$$\text{Protein} = k_0 + k_1 \cdot f_1 + k_2 \cdot f_2 + k_3 \cdot f_3$$

Unfortunately, the coefficients  $k_0$  to  $k_3$  are not known, and tend to vary from instrument to instrument. By running dozens of wheat samples with known protein concentrations and measuring the  $f_1$ ,  $f_2$ , and  $f_3$  values, we could compute the  $k$  coefficients by doing a least-squares fit of the data to the known protein levels.

The empirical data obtained during the instrument's calibration is used as a predictor for future measurements; the calibration determines the  $k$  coefficients, and then the instrument uses these coefficients and the equation to measure unknown samples.

Another, similar application is precision temperature monitors. The response of some thermistors is not linear—temperature **may** not be directly related to the thermistor's resistance. Some electronic thermometers operate by forcing the user to present known temperatures (typically through a specific chemical state change) to the instrument

and letting the unit calibrate itself using a least-squares process.

In effect, this self-calibration procedure is heuristic—the instrument learns from its mistakes. On initial power-up, the device is a blank slate, with the potential of measuring some parameter, but without the detailed "knowledge" of the world (represented by coefficients of an equation) needed to do the job. Calibration to known samples provides these coefficients. In situations where it is impossible (or not practical) to know an absolute transfer function between a signal's raw output and the value being computed, using a least-squares predictor is often a good approach.

Of course, this concept of calibrating an instrument and then using the calibration to predict new data is not the most common use of least squares. Scientists use it to fit curves through noisy data, computing a polynomial to fit the data and then plotting the polynomial itself (which must be smooth), rather than the raw data points. This lets them describe their data in a dramatic graph that is easy to interpret.

## LEAST SQUARES

All curve fitting techniques attempt to find a formula that relates one or more inputs to a single output. In other words, a function  $f$  is found such that:

$$g = f(v_1, v_2, v_3, \dots)$$

where  $g$  is the output and  $v_1$  to  $v_n$  are input signals.

The form of the function must be assumed; generally, a polynomial is selected since practically any function can be approximated (at least over some interval) by a polynomial.

Note that this is quite different than the polynomial approximations to trigonometric functions I presented in the last issue. Those formulas were derivations from series expansions that ultimately, if carried on to infinity, would yield exact solutions. Least squares is used when fitting noisy, inexact, real-world sampled data to a value. The noisy input data precludes an exact solution.

By definition, least squares constructs a function that minimizes the sum-square error between the measured

samples and the "real" value of the sample. To be more exact, it minimizes the function:

$$\sum_{i=1}^m (x(i) - x(i)_{comp})^2$$

where  $m$  is the number of samples in the calibration,  $x(i)$  are the predicted values, and  $x(i)_{comp}$  are the computed  $x$  values.

Least squares operates by recognizing the impossibility of an exact fit; it tries to keep the average error as small as possible.

Dozens of different approaches to curve fitting exist. Least squares is by far the most common. Another is the Chebyshev approximation. Chebyshev minimizes maximum error; no point is allowed to exceed some maximum value. This doesn't mean Chebyshev is more accurate than least squares, since typically the average error will be higher.

The coefficients of the polynomial can be computed by setting the sum-square formula to 0 and figuring the partial derivatives of it with respect to each of the terms. Once this is done, a matrix of equations is obtained. These are referred to as "normal" equations, and represent a system of formulas that must be solved for simultaneously.

In school, we all learned how to solve simultaneous equations by substitution. Computers are not particularly adept at this, so a number of approaches have been developed to expedite digital solutions of normal equations. The "Gauss Jordan" method is probably most commonly used, and is the one presented here.

Listing 1 shows a BASIC program that computes the normal equations used in fitting a least-squares curve to experimental data. It then uses a Gauss Jordan elimination to solve the matrix for the coefficients.

The program solves for a formula of the form:

$$y = k_0 + k_1 \cdot x_1 + k_2 \cdot x_2 + k_3 \cdot x_3$$

In other words, the program finds the  $k$  values that match the experimental data  $x_1$ ,  $x_2$ , and  $x_3$  to the known  $y$  data.

In the program, variable  $p$  is the number of terms in the equation. If the maximum coefficient is 3 (as above), then  $p$  will be 3. Variable  $n$  is the number of experimental samples being used to create the formula.

This program assumes that the array  $f$  will be loaded with the experimental data before it is invoked. Array elements are arranged in the following order:

	$x_1$	$x_2$	$x_3$	$y$
Sample 1:	$f(1,2)$	$f(1,3)$	$f(1,4)$	$f(1,5)$
Sample 2:	$f(2,2)$	$f(2,3)$	$f(2,4)$	$f(2,5)$
Sample 3:	$f(3,2)$	$f(3,3)$	$f(3,4)$	$f(3,5)$
etc.				

```

10 dim a(p+2,p+2)
20 for i=1 to pt2
30   for k=1 to p+2
40     a(i,k)=0
50   next k
60 next i
100 ' Build Normal Matrix
110 for t=1 to n
120   f(t,1)=1
130   for j=1 to p+2
140     for k=1 to p+2
150       a(j,k)=a(j,k)+f(t,j)*f(t,k)
160     next k
170   next j
180 next t
1000 ' Do a Gauss Jordan elimination
1001 k=1
1010 j=k
1020 s=a(k,k)
1030 a(k,j)=a(k,j)/s
1040 j=j+1
1050 if j <= p+2 then 1030
1060 if k >= p+1 then 2000
1070 i=k+1
1080 j=k
1090 s=a(i,k)
1100 a(i,j)=a(i,j)-s*a(k,j)
1110 j=j+1
1120 if j <= p+2 then 1100
1130 i=i+1
1140 if i <= p+1 then 1080
1150 k=k+1
1160 goto 1010
2000 if k <= 1 then 3000
2010 i=k-1
2020 j=k
2030 s=a(i,k)
2040 a(i,j)=a(i,j)-s*a(k,j)
2050 j=j+1
2060 if j <= p+2 then 2040
2070 i=i-1
2080 if i >= 1 then 2020
2090 k=k-1
2100 goto 2000
3000 stop

```

listing 1 -The least-squares fit can be readily programmed in BASIC. In the above program,  $n$ =number of experimental samples,  $p$ =number of terms in equation, and  $f$ =experimental data (dimensioned  $f(n,p+2)$ ).

Note that there is nothing in the first column of the array; the program fills it with ones during the computation.

When the program terminates, the coefficients  $k_0$  to  $k_n$  are stored in the "rightmost" row of the  $a$  matrix. In other words,  $a(1, p+2)$  is  $k_0$ ,  $a(2, p+2)$  is  $k_1$ , and so on.

Note that **this program** solves for a linear combination of terms ( $k_0 + k_1 \cdot x_1 + k_2 \cdot x_2 + \dots$ ) and not a polynomial combination of them.  $x_1$  to  $x_n$  are assumed to be independent variables: for example, absorptions at different wavelengths. If a polynomial fit were used, only one independent term (corresponding to a single physical parameter), raised to incremental powers ( $k_0 + k_1 \cdot x + k_2 \cdot x^2 + k_3 \cdot x^3$ ), would be used. The program can be modified to fit polynomial data by replacing the input data with the terms raised to the appropriate powers.

The concept of combining independent data into a single output is important. Many physical properties are

a function of several unrelated parameter-ven color is the combination of the three primary hues.

**WOW WELL DOES IT WORK?**

Least squares can fit the data remarkably well, but not perfectly. It is always important to evaluate just how well the computation proceeded. The great peril of computing is that the algorithm becomes a black box whose inner operations are hidden; it may not be easy to get a quick sanity check on its operation. Always apply some sort of independent test to your programs!

Fortunately, the science of statistics provides us with a great range of "goodness of fit" measures. Different applications require different yardsticks, but certainly one of the easiest to comprehend is Standard Deviation, which is defined by:

$$S.D. = \sqrt{\frac{\sum_{i=1}^n (x(i) - x(i)_{comp})^2}{n-1}}$$

**CAVEATS**

No mathematical technique, especially one as complex as least squares, can be indiscriminately applied to

Number of terms	Hn
1	1
2	8.3 x 10 <sup>-2</sup>
3	4.6 x 10 <sup>-4</sup>
4	1.7 x 10 <sup>-6</sup>
5	3.7 x 10 <sup>-12</sup>
6	5.4 x 10 <sup>-18</sup>
7	4.8 x 10 <sup>-25</sup>
8	2.1 x 10 <sup>-33</sup>
9	9.1 x 10 <sup>-47</sup>

Figure 1—The Hilbert determinant gives a relative idea of the difficulty of getting a good solution as a function of the number of coefficients.

any situation. Care must be taken to be sure the input data and results are truly meaningful.

The algorithmic nature of least squares (in a computer) tends to mask the details of its operation. Least squares only works well if the number of samples used in finding the coefficients is much larger than the number of coefficients. I've seen people try to solve for five unknowns (coefficients) with only four or five samples. With smooth, easy-to-fit data, 20 samples would be more reasonable. Noisy data requires more samples.

Once the coefficients are found, if the resulting equation is used to predict new values, be sure the values lie within the calibration range. For example, if the set of samples used to calibrate the instrument ranged from 10 to 20 picofarads (if capacitance is being computed), then

**We got you covered...**

... with complete embedded system support for popular PC compilers on Intel 80x 86 and NEC V-Series microprocessors. Now you can develop an embedded application with your choice of compiler—with full support for source level debuggers and in-circuit emulators.

Paradigm LOCATE supports Borland, C, Pascal, BASIC & Modula-2 compilers from Microsoft, Borland and others.

- comprehensive user's manual
- compiler startup examples
- free technical support
- math coprocessor emulation support
- extensible MS-DOS emulator
- full Intel MMX output

Lack of an emulator got you down? Not a problem with our Turbo Debugger interface option. Now you can debug your target hardware from the comfort of your PC.

**Paradigm LOCATE \$295.00**  
**Turbo Debugger Interface \$195.00**



Satisfaction Guaranteed  
 Orders: (800) 537-5043  
 Technical Support: (508) 478-0499  
 BIX: join paradigm  
 Visa/Mastercard/C.O.D. Accepted

**Paradigm Systems**  
 P.O. Box 152 Milford, Massachusetts 01757  
 Turbo Debugger is a registered trademark of Borland International.

**Total control with LMI FORTH™**

**For Programming Professionals:**  
 an expanding family of compatible, high-performance compilers for microcomputers

**For Development:**

Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics, floating point, native code generation

**For Applications: Forth-83 Metacompiler**

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11, 34010, V25, RTX-2000
- No license fee or royalty for compiled applications



Laboratory Microsystems Incorporated  
 Post Office Box 10430, Marina del Rey, CA 90295  
 Phone Credit Card Orders to: (213) 306-7412  
 FAX: (273) 301-0761

don't try to use the equation to solve for 30 picofarads. The fit is often quite poor outside of the calibration range of the instrument.

Finally, be careful of the number of terms used in the equation. More terms yield a set of Normal equations that is harder to solve. Small errors due to noise or even the resolution of the computer's floating-point package can give wildly incorrect values. (The Standard Deviation will flag these problems). The Hilbert determinant gives a relative idea of the difficulty of getting a good solution as a function of the number of coefficients. Figure 1 lists the Hilbert determinant for one to nine terms-with nine terms the equation is about  $10^{44}$  times more difficult to solve than with one term. If you need a lot of terms, then the use of orthogonal functions will often yield a solution that is easier to solve, and thus more accurate and dependable. Orthogonal functions are beyond the scope of this article, but are described in the references.

#### MORE OF AN ART...

Sometimes using least squares is more of an art than a science. Picking the right number of terms requires knowledge of the physics of the measurement and some experimentation. The form of the equation (linear, polynomial, or even a combination of complex functions) must also be selected, generally using some a priori knowledge of the system.

Least squares can be a bit complicated to apply, especially for the non-mathematically inclined. It is worth the trouble, especially since the results are easy to test. If the same calibration sample set is run through the instrument with the computed  $k$  values, and crazy results are obtained, there is a flaw in the code or in the selection of terms. ❖

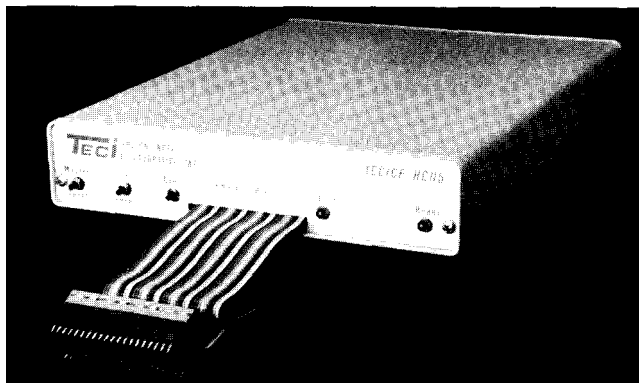
#### REFERENCES

**Numerical Methods for Scientists and Engineers**,  
R.W. Hamming, McGraw-Hill.  
**Digital Computation and Numerical Methods**,  
Southworth and Deleuw, McGraw-Hill.

*Jack Ganssle is president of Softaid, a vendor of microprocessor development tools. When not busy pushing electrons around, he sails up and down the East Coast on his 35-foot sloop.*

#### IRS

225 Very Useful  
226 Moderately Useful  
227 Not Useful



### 68HC05 IN-CIRCUIT EMULATOR

The **TECICE-HC05** is a low-cost real time emulator for the Motorola 68HC05 family of single chip microcomputers. Any host computer with serial port and terminal emulation software can be used with **TECICE-HC05**. Base price is \$1,195.00. Complete development system software is available for MS-DOS computers including the Byte Craft Limited C6805 Code Development System which includes a 6805 C compiler with Integrated Development Environment.

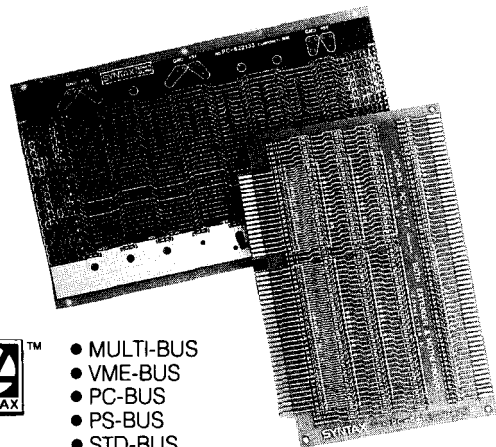
**TECI** THE ENGINEERS  
COLLABORATIVE, INC.

RR#3, BOX 8C • Barton, Vermont 05822  
Phone (802) 525-3458 • Fax (802) 525-3451

Reader Service #119

## PROTOTYPING SOLUTION

We offer the best quality  
**SYNTAX FR-4** prototyping PCB at the best price!  
All from the stock



- MULTI-BUS
- VME-BUS
- PC-BUS
- PS-BUS
- STD-BUS
- S1 00-BUS
- TRANS BOARD
- UNI BOARD

• Please Call Today for Catalog  
'Custom Design are Available  
'Dealer program Offered

Manufacturing & Marketing

**MING**™  
Engineering & Products, Inc.

977 South Meridian Ave., Alhambra, CA 91903  
Tel: (818) 570-0058 • Fax: (818) 578-8748

Reader Service #141

# New X-1 O-Compatible Products Hit the Market

## Domestic Automation

Ken Davidson

“**O**pen the pod bay doors, please, HAL.” I know. It’s a worn out cliché too often used to describe the ultimate in home (or environment) control, but I couldn’t resist. It does show, however, man’s long-time fascination with automating his environment, making more time for leisure and devoting less time to work and toil.

As members of the forefront of computing technology, we’ve found that the typical *CIRCUIT CELLAR INK* reader has a great deal of interest in home and building automation. Many of you have automated your homes to some degree, whether through the purchase of an off-the-shelf control system or, more likely, through the design and implementation of a system from the ground up. Domestic Automation is a new column that will appear regularly in *CIRCUIT CELLAR INK* and will be devoted to keeping you at the forefront of new home automation technologies and techniques.

### CEBus

Of course, the big news in the home automation arena these days is CEBus, the Consumer Electronics Bus. I did a complete technical overview of CEBus in the August/September ‘89 issue (#10) of *CIRCUIT CELLAR INK*, but let me quickly fill in those who may have missed that article.

CEBus is actually a complete home network description based on the OSI/ISO seven-layer network model. Devices—which can include TVs, dishwashers, lamps, door locks, draperies, hot water heaters, and so on—are connected to the network via one of several available media. Currently proposed media include power line (PLBus), twisted pair (TPBus), coax (CXBus), infrared (SRBus), radio (RFBus), and fiber optic (FOBus).

As you go up through the layers of the network model, you slowly work from the specifics of how a message gets from one device to another to more general terms concerning just what the message means. Each layer is responsible for just one aspect of the network communication; it relies on the other layers to fill in the gaps. For example, the data link layer only cares about getting a packet from one node to another without errors. It doesn’t care what the information in the packet means (a higher layer takes care of that) or how it physically gets from here to there (a lower level takes care of that).

At the top network layer is CAL, or Common Application Language. CAL is based on tables maintained by EIA and contains commands for stereos, TVs, lamps, clocks, dishwashers, washing machines, telephones, and the list goes on. Presumably, all lamp controllers will respond to the same on, off, dim, and bright commands no matter who the manufacturer is, making universal control possible. Similarly, all stereos will respond to the same volume or tuning commands, all VCRs to the same tape control commands, and so on. Of course, each device will have a unique unit number, so individual control is possible, but having universal commands available makes it easy to put together a very powerful and flexible home control system with separate components from a whole host of manufacturers.

By using a layered network model, it’s possible to make the upper layers in all devices identical, with only minor differences depending on which physical medium is being used. For example, the TV and VCR may communicate over coax, but the lights are on the power line. Bridges make it possible for devices on different media to communicate without ever knowing that they are on different media. A very common example of a bridge EIA expects to see is inside the television. A hand-held infrared

---

---

**Speaking of official releases, EIA is scheduled to release the first working specifications as we go to press (mid-November).**

---

---

remote will send CEBus commands to the TV. The TV will then retransmit those commands over coax to VCRs and other devices on the CXBus, while it also retransmits them onto the power line for any devices connected to the PLBus.

CEBus also makes two-way communication between devices possible. Instead of sending a command to the door lock to secure the **door and** just hope that it gets there, we can wait for a confirmation from the lock that it actually was successful in getting the job done. If there is a problem getting the door closed, or the network connection to the lock has somehow been disrupted, we'll get an error message and can go investigate further.

Enough about how CEBus works. If you want all the details, I'd suggest digging out issue #10 and curling up next to the fireplace. Let's get into CEBus' current status.

First of all, "CEBus" is really just a working name for the standard. EIA expects to have a new name for it when it is officially released, but nobody has a clue as to what that name will be yet. After five years of calling it CEBus, I would think members would have a difficult time referring to as something else, but we'll have to see.

Speaking of official releases, EIA is scheduled to release the first working specifications as we go to press (mid-November). While these won't be the cast-in-stone final specifications, they will be final enough for interested parties to begin product development and are intended to be the subject of debate among those in the industry who might have differing opinions about how things "should" be done. After an initial "comment period," the final specification will be released.

Expected to be released are the PLBus physical layer, all the higher layers (data link, network, presentation, and application layers), and CAL. The other physical layers will be released in the future **one** at a time as they **are** ready. I plan to be at that meeting **and** will have a full report in the next issue. In the meantime, there will likely be some discussion on the Circuit Cellar BBS (203/871-1988) about what happened at the meeting, so give us a call and jump right in.

I did make it to the August meeting of the CEBus committee in Boston and was both pleased and disillusioned by what I saw. After having worked on the CEBus spec for almost five years now, and with the expectation of releasing PLBus and CAL in November, I had expected committee discussions to focus around last-minute minor details and finishing touches related to those two areas of the specification. Instead, I was greeted with presentations by committee members proposing entirely new methods and concepts for both the high-level language layer and the physical layer. Fortunately, common sense prevailed and some useful work was done toward having something ready for the November release based primarily on past work, but some major issues still needed resolving, especially in the PLBus camp. Things should be interesting in the next few months.

x-10

The folks at X-10 haven't exactly been sitting around, either. Veteran INK readers will remember the articles I've done in past issues about computer interface modules and an IR gateway from X-10 (Issues #3, #5, and #9). The latest

Radio Shack and Heathkit catalogs yield some interesting new products.

Radio Shack will be carrying the infamous momentary contact module. Those who closely follow X-10's products will remember hearing about the momentary contact module quite a while ago. X-10 didn't seem to be able to get their act together enough to get the thing made. Now it may actually become reality. Judging from pictures and descriptions (no, I haven't been able to get one, either), the unit will close a pair of dry contacts and/or sound a built-

---

---

The most interesting addition to both Heath and Radio Shack's product lines is a complete home security system based on X-10 modules and RF transmitters and receivers.

---

---

in speaker when addressed, depending on how a switch on the module is set. Another switch controls whether the closure is momentary or latched. We've heard from a number of people on the BBS who have modified appliance modules for similar operation, but we'll finally have a unit that does it for real. One real-world use for such a module that many people have suggested is in triggering a garage door opener, where a momentary push of a button is what's needed.

Along these same lines, Heath lists a "Sensor Chime" module that does nothing but sound a "pleasant chime" when activated. Might be nice to use in conjunction with a home control system that also functions as an alarm system. When the system has been armed, it could sound a chime to signal that you have sixty seconds to vacate the area. By the same token, upon returning home, the chime could signal a warning that you only have thirty seconds to get the alarm disabled before the big guns are brought to bear.

Another unit that Radio Shack is selling for the first time this year (which Heath started selling several months ago) is a pair of flood lights with a built-in motion detector. Before you say, "Big deal, those are all over the place," realize that this one also transmits X-10 commands over the power line so lights inside the house can also be turned on in response to outside motion. Used in conjunction with the two-way computer interface module I discussed in issue #5 (the TW523), your home control computer can also be made aware that there is motion outside and that it

should be alert for possible trouble (or can trigger some other response such as a voice from a speaker).

A unit similar to the IR gateway mini controller I discussed in issue #9 is being sold by Heath that has a photosensor in it. When it gets dark out, the controller sends an "on" command onto the power line (presumably to a lamp module). When it gets light out, the controller sends an off command. Like above, this controller could be used with the TW523 module and a home control computer to tell the computer when it's dark out and when it's light again.

The most interesting addition to both Heath and Radio Shack's product lines is a complete home security system based on X-10 modules and RF transmitters and receivers. There is a master unit that has an X-10 transceiver, RF receiver, speaker, and indicator lights on it. Attached to doors and windows are sensors that send signals onto the power line when they are tripped. Finally, there is a hand-held RF transmitter used to enable and disable the alarm system. With the system disabled, a soft chime sounds when one of the sensors is tripped and the LEDs on the master unit show the sensors' status. When the alarm is enabled and is tripped, a loud siren sounds and lights are flashed for four minutes. Let's just hope those power glitches that like to occasionally wreak havoc with X-10 modules don't wake your neighbors (and you) in the middle of the night some time.

And the future...

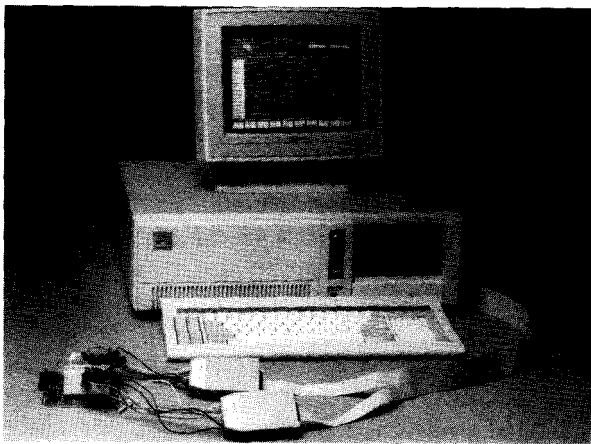
That about does it for this issue. Future plans include complete coverage of the November release of the CEBus working specification, any new CEBus-based products that may be slated for release to coincide with the release of the spec, a nifty hand-held infrared remote control that should have a good deal of appeal to the home automation experimenter, and anything else you want to see. Drop me a letter or send me a message on the Circuit Cellar BBS with ideas, suggestions, or pointers to new products that have caught your eye. Plus, if you've implemented some form of automation in your home that you think others would enjoy reading about, tell me about that too.✚

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

## IRS

228 Very Useful  
229 Moderately Useful  
230 Not Useful

### ID160/161



\*100MHz for \$745

- 50 MHz Sampling Speed • Multi-level triggering
- 8K trace buffer • 32-channel capability
- Event Timer/Counter • Performance Histograms
- Hardcopy output • Disassembles B-bit micros
- and much more ! • Satisfaction Guaranteed



INNOTECH DESIGN, INC.  
P.O. Box 3304  
Cerritos, CA 90703-3304  
Tel: 714-527-8540 FAX: 714-527-1812

Reader Service #126

Create Professional **Quality** Circuit **Diagrams** with

# MacSchematic

MacSchematic is a library of over 800 Electrical and Electronics symbols for professional quality circuit diagrams on the Mac.

Symbols are in both PICT format for MacDraw,

MacDraft, or other CAD/Draw programs and in MacDraw II libraries,

MacSchematic symbols are object oriented for superior printer, plotter and laser output. They can be rescaled without losing their high resolution.

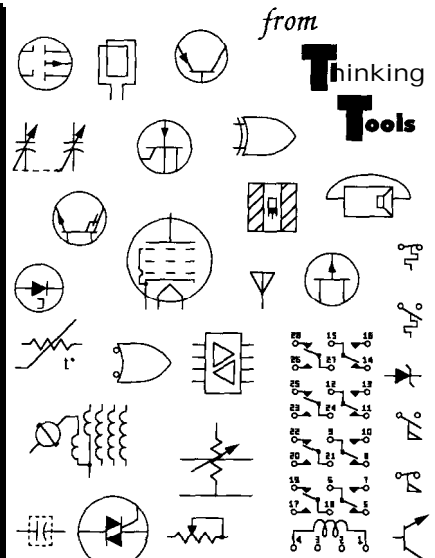
MacSchematic symbols are designed to snap to grid so their external connections fall on grid points and lines connect to symbols "on the dot".

Includes symbols from ANSI Standard Y32.2:

- Analog Components
- Gates/Digital Devices
- Industrial Wiring
- Ladder Diagrams

Not Copy Protected

All Macs



Symbols shown at 50% Reduction

MacSchematic: \$80  
Demo + Manual: \$10  
plus \$5 shipping/handling

To Order  
Send Check, MO, or PO:  
Thinking Tools  
2411-M Linden Ave  
Baltimore, MD 21227

For Information  
or Visa/MC Order  
Call (301) 383-6490

Reader Service #153



## 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

*Topics that we'll be covering in this issue's ConnecTime include emergency cut-off switches, waveform conversion, and switched-capacitor filter design. In this first discussion, we find that hardware isn't always the basis for all message threads on the Circuit Cellar BBS; software rears its ugly (?) head at times, too.*

### Msg#:22701

From: MARVIN GOLDBERG To: ALL USERS

I have written a program to display a large graphic file (230K). This program is written using Turbo C v 2.0 and is being used on PS/2 Model 50 with VGA. The following function receives a line of data at time from the file and then displays the image on the screen, one line at a time. It performs as expected except that it is very slow. I have the feeling that this function will perform faster if written in assembler language. Is it possible to replace any or all of this function with in-line assembler code using the Turbo Assembler? If so, I would be grateful to see what this code looks like since I am not too handy with programming assembler code.

```
void put_line(line_no,buf) int line_no;
char *buf;
{
    register int i;
    unsigned int offset; /* x coordinate of data in line */
    i=512; /* size of buffer */
    offset=0;
    while(i-->0)
    {
        putpixel(offset++,line_no,*buf++);
    }
}
```

### Msg#:22702

From: STEVE SAMPSON To: MARVIN GOLDBERG

You might try using the `_read()` function, and also `setvbuf()` and make the buffers larger, say 1024 for a start. Before you go off into assembler. The problem as I see it is that the disk access is going to be the slowest item. It might be better to drink in a compressed file completely and then uncompress and display the result. I don't think assembler will help with disk access speed.

### Msg#:22717

From: NATHAN ENGLE To: STEVE SAMPSON

Another thing to do is to use the "++" construct as a prefix rather than as a postfix (i.e., ++i rather than i++). It wouldn't necessarily save that much in CPU time, but it does evaluate faster since the evaluated value of ++i is the same as the value of i AFTER incrementing (which is already in the primary evaluation register). When you use i++ you make the compiler backtrack and put the old value in that register (just because of the way expressions evaluate in C).

A good optimizer can spot places where you compute something and never use it, so a lot of crud like that disappears when you turn the optimization switches on maximum.

### Msg#:22754

From: ED NISLEY To: MARVIN GOLDBERG

One thing you can do is combine the loop counter and pixel offset into a single variable. Right now you're doing two arithmetic operations per iteration; cut one out!

Next, you can use a block move (of the memcopy flavor) to flash a whole line into the buffer given the starting address. That gets rid of the C-level iteration and is really going to be the biggest win.

Next, snag a 32K buffer (using `malloc`) and suck in huge gulps of the file. If the pels are all contiguous in video memory (which they will be if you're in VGA native mode and not doing windows or some such), use `memcpy` (or some such) to block move the whole buffer.

I suspect that'll give you a 100x speedup. ..time it and report back!

*Anyone who has ever worked with an alarm system knows that there is a fine line between a system that is so sensitive that false alarms become a problem and one that isn't sensitive enough to pick up trouble. The same applies for remote emergency cut-off switches, as the next message thread shows.*

**Msg#:24119**

From: ROBERT MEUSHAW To: ALL USERS

I am need of a wireless remote control transmitter and receiver which can be used in a factory environment to send an emergency cutoff signal to a machine. The control action must be very reliable (since it is for emergency use) and it should have a range of at least 100–150 feet. I have considered the use of RF, infrared, and ultrasonic. Obviously, they must work in a noisy (electrical and sound) environment.

For simplicity, it seems that infrared has an advantage, but it may be too directional and short range for this application. I am considering using a Motorola encoder (MC145026) to provide data to the transmitter, and using a Motorola decoder (MC145027) to process the received data. The transmitter must be small enough to keep in a pocket. The receiver size is not critical, and it can be mounted almost anywhere, including high above the machine. For RF controls I would prefer preassembled modules which can be integrated with other machine controls, since RF devices generally seem tricky to build and use reliably. However, I would consider construction articles if they are relatively fool-proof. I am mainly interested in simplicity of design and in construction. Any suggestions would be greatly appreciated.

**Msg#:24132**

From: DALE NASSAR To: ROBERT MEUSHAW

I would feel a little shaky using IR for an emergency cut-off at 150 feet. You may want to consider using a DTMF signal sent over one of those 49-MHz EM transceivers. The transmitter could be hard wired so that the push of a momentary button sent the signal as well as turned on the transmitter. The things are about the size of a cigarette pack and are about \$50 each at Radio Shack.

**Msg#:24136**

From: BOB PADDOCK To: ROBERT MEUSHAW

For a fail-safe system, you should look for the loss of a signal, instead of trying to detect a signal. It is far better to have a few erroneous shut downs for whatever reason (receiver died, transmitter battery when dead, the service center took the antenna off because it was in the way of servicing the receiver, etc.) than to hit the \*PANIC\* button and have nothing at all happen, while watching the machine grind one of your coworkers to bits. (Paranoid, perhaps, but it just might be me standing between the wall and the machine.)

**Msg#:24140**

From: KEN DAVIDSON To: ROBERT MEUSHAW

I don't think you're paranoid, Bob. As I was reading the original message and the first reply, I was thinking exactly what you were. I would try setting up a low-power RF transmitter that transmitted continuously, with the press of a button cutting off the transmission. After having worked with a number of IR-based devices, I wouldn't trust my life to IR, and certainly not at 150 feet. I think the biggest problem to solve at that point is the power source. You'd likely have to put the unit on a charger each night.

**Msg#:24142**

From: JEFF BACHIOCHI To: ROBERT MEUSHAW

Ken's suggestion of a constantly transmitted OK is a good one. It is much easier to shut a transmission off than rely on one getting through! Although it prevents more than **one** user from stopping a life-threatening situation.

**Msg#:24220**

From: ROBERT MEUSHAW To: JEFF BACHIOCHI

The machine that I'm interested in stopping has several normal controls for that purpose. Generally what happens is that the machine gets jammed and must be cleared. The remote shutoff is just a convenience that the operators might use if they are not next to the machine when it jams. For this reason, I'm not sure I'd want a loss of carrier to stop the machine. The nuisance factor might keep operators from wanting the system at all.

**Msg#:24219**

From: ROBERT MEUSHAW To: DALE NASSAR

Thanks for the reply. I tried one of those **cheapo** transmitters from Radio Shack, and the range was insufficient. Do you know of any others that might work?

**Msg#:24233**

From: DALE NASSAR To: ROBERT MEUSHAW

You may have tried the "toy" versions. The better ones that I have used have easily given me a range of >500 feet. When you said "emergency cutoff," I may have wrongly assumed that the situation was not life threatening. Because you were considering IR, I assumed that you just wanted a secondary cutoff.

---

*Even the most esoteric request will often get a response or a pointer to where an answer can be found. Here, a user is looking for a waveform output.*

**Msg#:23187**

From: MICHAEL CAVANAUGH To: ALL USERS

I am in need of a chip to convert a DC voltage (either 6 or 18 VDC) to two sinusoidal waveforms as an output. The two outputs must be 90 degrees out of phase (sin and cos) and 16 volts peak-to-peak ( $\pm 8$  volts). They must also be at 250 Hz. A 6-VDC square wave at 250 Hz is available as an input to clock the output. Any ideas will be greatly appreciated.

**Msg#:23290**

From: DALE NASSAR To: MICHAEL CAVANAUGH

The square wave you have along with the fixed frequency can make things simple for you. If you filter the square wave with a

low-pass filter you can obtain the fundamental component which is a sine wave of the same frequency. If you pass this through a simple op-amp integrator (inverting), the output will be a cosine wave (same frequency). If this output is then passed through another inverting integrator, the output is the negative sine (you can pass this through an op-amp inverter to get sine). Use a quad op-amp such as the TL084. I did a one-page article in *Radio-Electronics* describing how to filter out the fundamental sine from a square wave (April 1988, page 77). It also described how to make the frequency and amplitude controlled independently, each with a single pot-a feature I found to be hard to obtain in a simple circuit, which is why I wrote the article.

**Msg#:23483**

From: MICHAEL CAVANAUGH To: DALE NASSAR

Sounds like a good idea. If I run the square wave through a low-pass filter, what should be the cut-off frequencies? I will use the final waveform outputs to provide excitation to several resolvers, so I may need to use some type of power op-amp to drive this type of load. I will play with your idea on the breadboard and see what shakes out.

**Msg#:23670**

From: DALE NASSAR To: MICHAEL CAVANAUGH

The fundamental sinusoidal that you want to filter from the square wave (it is SQUARE??) is of the same frequency as the

square wave. Op-amp filters will work, but I have found switched-capacitor filters less troublesome. For high-input-impedance buffers you can try an op-amp configured as a direct-input voltage follower.

*Speaking of switched-capacitor filters, the next discussion actually covers two topics. The first deals with switched-capacitor filter design, while the second addresses a problem most electronics experimenters face at one time or another: where to find hard-to-find parts.*

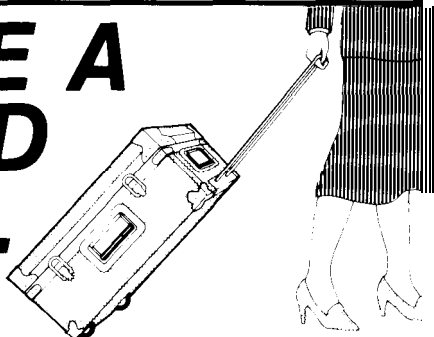
**Msg#:23175**

From: MATTHEW TAYLOR To: ALL USERS

I am considering using a switched-capacitor filter (like the MF10) as the low-pass filter for a 12-bit A/D converter. Has anyone worked with these filters? Is clock noise a problem? Does anyone know of any higher-order switched-capacitor filters?

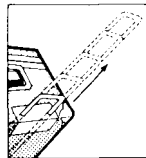
Also, as a student, I have never dealt with any parts distributors such as Arrow, Hamilton/Avnet, Pioneer, etc. Do they frown on doing business with individuals? I realize they have minimum orders, but could anyone please elaborate on what I should expect? I am trying to get some parts that Jameco, JDR, and Digi-Key don't carry.

# TAKE A LOAD OFF...



A rugged CABBAGE CASE? lined with plenty of foam for your equipment can **TAKE A LOAD OFF YOUR MIND** when you've got to travel.

**TAKE A LOAD OFF YOUR BACK** with our exclusive tilt-wheels and extension handle option.



**UNLOAD ON US!**

Call or write to tell us about your shipping or carrying problems  
**WE HAVE SOLUTIONS!**



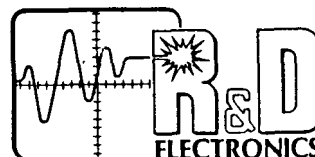
CABBAGE CASES, INC.  
1 166-C STEELWOOD ROAD  
COLUMBUS, OHIO 43212-1356  
(614) 486-2495 (800) 888-2495

## R&D ELECTRONIC SURPLUS, INC.

Has been in business since 1946 selling NEW surplus electronics and electromechanical parts.

Send for our FREE 40 page catalogue detailing:

Batteries	LEDs
Cables	Lugs
Capacitors	MOVs
Clocks	Ni-Cads
Connectors	Power Devices
Digital Timer/Controllers	many Power Supplies
Diodes	Relays
Displays	SemiConductors galore
Enclosures	Stepper Motors & Driver ICs
Fans	Speakers
Filters	Many Switches
Heat-Shrinkable tubing	Telephones and components
Heatsinks	Transformers
Integrated Circuits	Ultrasonic Transducer Board
Lamps & Lights	Zeners
	& many more items



1224 Prospect Avenue  
Playhouse Square  
Cleveland, OH 44115

Telephone: (216) 621-1121 • Fax: (216) 621-8628

Msg#:23192

From: PAUL SHUBEL To: MATTHEW TAYLOR

Arrow puts out a catalog for mail-order electronic parts. The minimum order is \$25.00; they take credit cards. I have ordered from them with no problem. The nice thing about buying chips from them is that you know that they will be factory fresh. Their order hot line is 1-800-932-7769.

Their catalog contains chips and all different electronic parts and components. They will probably send you a catalog if you call and ask. A while ago they promoted this catalog through the bingo cards. To anyone else interested, they actually carry the 80% family and most TI chips including the elusive TMS70xx. Ditto for Allied Electronics, phone 1-800-433-5700.

Msg#:23289

From: JAMES D STEWART To: MATTHEW TAYLOR

The MF10 will work great as a low-pass filter on an A/D or D/A. I'm using them on two of my designs in this way. Just make sure that the sample frequency and the filter clock frequency are derived from the same oscillator. If you use two different oscillators, you will get horrible beat frequencies in the audio. In terms of dealing with the big parts guys, tell them you are a little start-up company and that you will pay COD. They will treat you OK. Also, don't be afraid to ask them for samples and data sheets.

Msg#:23313

From: MATTHEW TAYLOR To: JAMES D STEWART

I was hoping the MF10 would work. I don't really like wiring up high-order filters: boring and tedious. Not that carrying 16 address lines throughout a wire-wrapped project is any joy. :-)

Are your designs digitizing audio? I've built several audio processors using 8-bit data converters and a Signetics compander (to help with the dynamic range). The results were less than thrilling, but the designs worked. I've since learned a lot about real world designs that NONE of my EE books (or professors) have ever mentioned. This next design will \*NOT\* have the analog and digital portions of the circuit living on the same wire-wrap board!

If you have time, I would love any advice you could offer on keeping things quiet in an analog/digital system. How do you test your designs? I am planning on doing the analog section on a breadboard with a ground plane and probably wire-wrapping the digital section. Sound OK?

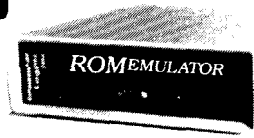
Msg#:23333

From: JAMES D STEWART To: MATTHEW TAYLOR

My designs digitize audio to 10 bits, run the digital data through an OKI ADPCM processor chip, and store the resulting four bits

## Still Blasting ROMs?

Develop and test ROM code in minutes without leaving your keyboard — with the GEI ROMemulator.



Works with any micro (8, 16, 32 bit)

ROMemulator need not be located near host PC

Works with most PCs, Unix systems and Macs

Daisy-chain up to 8 ROMs worth from one serial port

Target can be Z80, 80286, 68020, etc.

Doesn't use a host slot — loads over a RS232 link

Loads most popular hex data formats

Source code for host software is included

Ask us about faster (than 150ns) ROMemulators and custom cables for most non-27xxx ROM sizes.

**115.2K BAUD RATE UPGRADE NOW AVAILABLE**



**Grammar Engine, Inc**  
3314 Morse Road  
Columbus, Ohio 43231  
614/471-1113

VISA and MasterCard Accepted  
Dealer Inquiries Welcome

In Pacific and Mountain timezones:  
415/595-2252

## Files Lost in the Z-Environment?

NSURVEY™ is the file name cataloging program for the Z-System where a file can be in any of 32 directories.

NSURVEY keeps a database of file name, disk identification, directory, length and CRC for every file. NSURVEY allows searching the database on any field; wild card characters permitted.

NSURVEY edits the database to allow removing old disks.

NSURVEY allows a list of exclusion names to specify what files, or class of files, are not added to database.

NSURVEY can operate from a control file to allow automating unique-to-the-directory operations.

NSURVEY comes with 24k of documentation in both print and .HLP formats.

NSURVEY combines all updating, searching, and editing functions into one program that totally manages the file name database.

**For CP/M 2.2, CP/M 3, ZCPR3, Z-Systems.**

**\$28.95 + \$4.00 S+H**

**Skunk Creek Computing Services**  
1985 Kohler Drive, Boulder, CO 80303

of data in RAM or EPROM. I digitize at 8 kHz and filter at about 3.5 kHz on both the A/D and D/A. I have been told by my customers that my speech quality is the best they have ever heard for the applications they are doing (security and process control). As far as keeping things quiet on a wire-wrap board, it is possible, but not easy. Physically separate the digital and analog and put the A/D in the middle. Part of your problem may be the limited range of an 8-bit conversion. I have found that for the quality that I and my customers will live with, I need at least 10 bits and 8-kHz samples. One of my cards lives in an IBM PC and has a low-impedance microphone input, and no noise problems. I ran the analog stuff off of  $\pm 5$  volts that was regulated on the board with LM78L05/LM79L05s from the PC's  $\pm 12$  volts. The cleanest possible design would have the A/D converter separated by optoisolators from the digital side. Hope this is of some use to you. Good luck.

**Msg#:23354**

From: MATTHEW TAYLOR To: JAMES D STEWART

Thanks for the reply. My analog design skills are a little weak at this point (as far as guarding against noise).

One of my projects was a real-time pitch shifter that sampled at 100 kHz and stored the sample in RAM. A separate circuit would read the data out of RAM at a different clock speed (50-200 kHz) and a send it straight to the D/A converter. Sticking a sine wave into the device yielded a sine wave on the output, but it was really

fuzzy. A scope on the A/D convertor showed a LOT of noise that wasn't present when all the digital stuff was unplugged.

This next project, I'm using 12-bit converters and skipping the compander circuitry. I was using a Signetics NE570. Its internal op-amp was similar to a 741.

If you are ever wondering what someone CAN do with 8-bit audio, go to a professional music store and listen to an (audio) digital delay made by DigiTech. Many of their older products were 8-bit and sounded fantastic! I was a little disappointed to open one of their units and see the pre- and postdigital section. Very similar to mine (including the '570).

*The Circuit Cellar BBS runs on a 10-MHz Micromint OEM-286 IBM PC/AT-compatible computer using the multiline version of The Bread Board System (TBBS 2.1M) and currently has four modems connected. We invite you to call and exchange ideas with other Circuit Cellar readers. It is available 24 hours a day and can be reached at (203) 871-1 988. Set your modem for 8 data bits, 1 stop bit, and either 300, 1200, or 2400 bps.*

*Complete your reference library —  
you cannot afford to miss an issue of  
Circuit Cellar INK*

### AVAILABLE BACK ISSUES

January/February 1988—Issue # 1 SOLD OUT  
March/April 1988—Issue #2 SOLD OUT  
May/June 1988—Issue #3 SOLD OUT  
July/August 1988—Issue #4 SOLD OUT  
September/October 1988—Issue #5  
November/December 1988—Issue #6

*The above issues are available as a Bound Offset Reprint of the  
First Year of Circuit Cellar INK see page 81 for details.*

January/February 1989—Issue #7  
April/May 1989—Issue #8  
June/July 1989—Issue #9 SOLD OUT  
August/September—Issue #10  
October/November—Issue #11

**To order Circuit Cellar INK back issues, send  
\$4.00 check (Visa, MasterCard accepted) for each  
issue to:**

Circuit Cellar INK  
Back Issue Sales  
4 Park Street  
Vernon, CT 06066  
or call (203) 8752199

IRS

231 Very Useful  
232 Moderately Useful  
233 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 (December/January 1989/90) includes all public messages posted during September and October, 1989.

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.

## ...And Everything in its Place

Certainly I should be the last guy in the world that would have any complaints about current technology and involvement of computers in our lives. I mean, after all, aren't I Mr. Embedded Controller?

I didn't get upset when I found myself "negotiating" proper cooking times with a 4-bit computer-controlled microwave oven. My blood pressure doesn't even go up a notch when the new FM stereo WON'T let me listen to my favorite station because ITS computer-controlled programming has decided that the low signal strength of my station isn't worth locking onto! ("Having a mind of its own" takes on a new meaning with today's technology).

For what it's worth, however, I am coming to a slow burn over the computer control systems in cars. Have you looked under the hood of a car recently? Surely it is no longer the province of a grease-covered mechanic but, are we to substitute a plumber, electrical engineer, physicist, or a computer programmer when maintenance is required?

My latest hot rod is high-tech German steel. The car is a wonderful evolution in performance and control but, like most new cars, it can be a nightmare of pushbuttons and computers. When you get in it the first time you spend about 2 hours programming the "central computer" with the time, mileage alert thresholds, seat positions, radio stations, wiper speeds, mirror positions, and assorted other electrical stuff. The one item I dared not program was a multi-digit alarm code which enabled the ignition system when you entered the car. Entering the wrong code put the car out of commission for a half hour. (I had already lost the argument with my home stereo so I was gun shy).

Coexistence with this marvel of technology was fine until I decide to take the car for a spin and the battery was dead. The car had been sitting in the garage for 3 weeks and the battery was dead! I charged the battery, reprogrammed everything, and then shut the car off. Just for curiosity I lifted a battery lead and there was a spark! But the car was off! A check with an ammeter showed that the computer was drawing 300mA with the car off. Give me a break!

To add insult to injury, the "central computer" decided to hiccup. While I was on my way to the dealer to "discuss" this constant current drain discovery, I was passing through a construction area. To this day I will argue that the guy frantically waving the flag meant that I should get through as fast as I could, but by then it was too late. You remember those movies where the car sail through the air? I did that over a pile of wooden concrete forms!

The landing was hard but my German steel survived with flying colors (quite exhilarating actually). Unfortunately, the engine started missing and I started losing power. Thinking the worst, I limped into the dealership and pulled into one of the service bays. Three technicians, responding to the emergency, dragged sophisticated diagnostic equipment and electronic probes to the car. My car soon looked like an octopus of electronic cabling.

I related the travel incident to the technician monitoring the central diagnostic console. He shook his head and pointed at the display, "Ze computa is ded! Four cylinda is verking! Four cylinda is capute!" The parts manager piped in, "I can get you a new ignition control computer in about a week... Cost about \$3000! Are you gonna need a rental?"

As I was about to fall on the floor in grief, the service manager showed up. He silently walked around the sputtering car, looked at the diagnostic displays, and looked under the hood at the badly vibrating engine. Next he picked up a piece of scrap wood and walked back to the car. To our amazement he took the wood and hit under the passenger-side dashboard. The engine jumped RPM for an instant and then sputtered. The service manager bent down under the dash, took the piece of wood and jammed it up behind the glovebox. Instantly the engine roared to life and the red flashing lights on the diagnostic panels went to green. He just smiled and said, "Loose connector." I just felt like hugging the car.

Even though I made it home safely, I was left with a dilemma. Does living with a high-tech car mean having an extension cord hanging out of the engine block and a 2x4 braced against the computer? Is all this computer stuff now so complicated that service personnel think only to replace the whole system rather than understand and fix a problem?

The success of microprocessors in toasters and appliances is that they demand very little understanding from users as to what they are rather than how they perform. The most successfully implemented embedded control computer is one that functions flawlessly and is totally ignored. At one time they were made with springs and levers. Today they use microprocessors. Perhaps the next evolutionary stage will have something suitable for cars.

