

CIRCUIT CELLAR

INK®

# THE COMPUTER APPLICATIONS JOURNAL

October 1994 — Issue #51

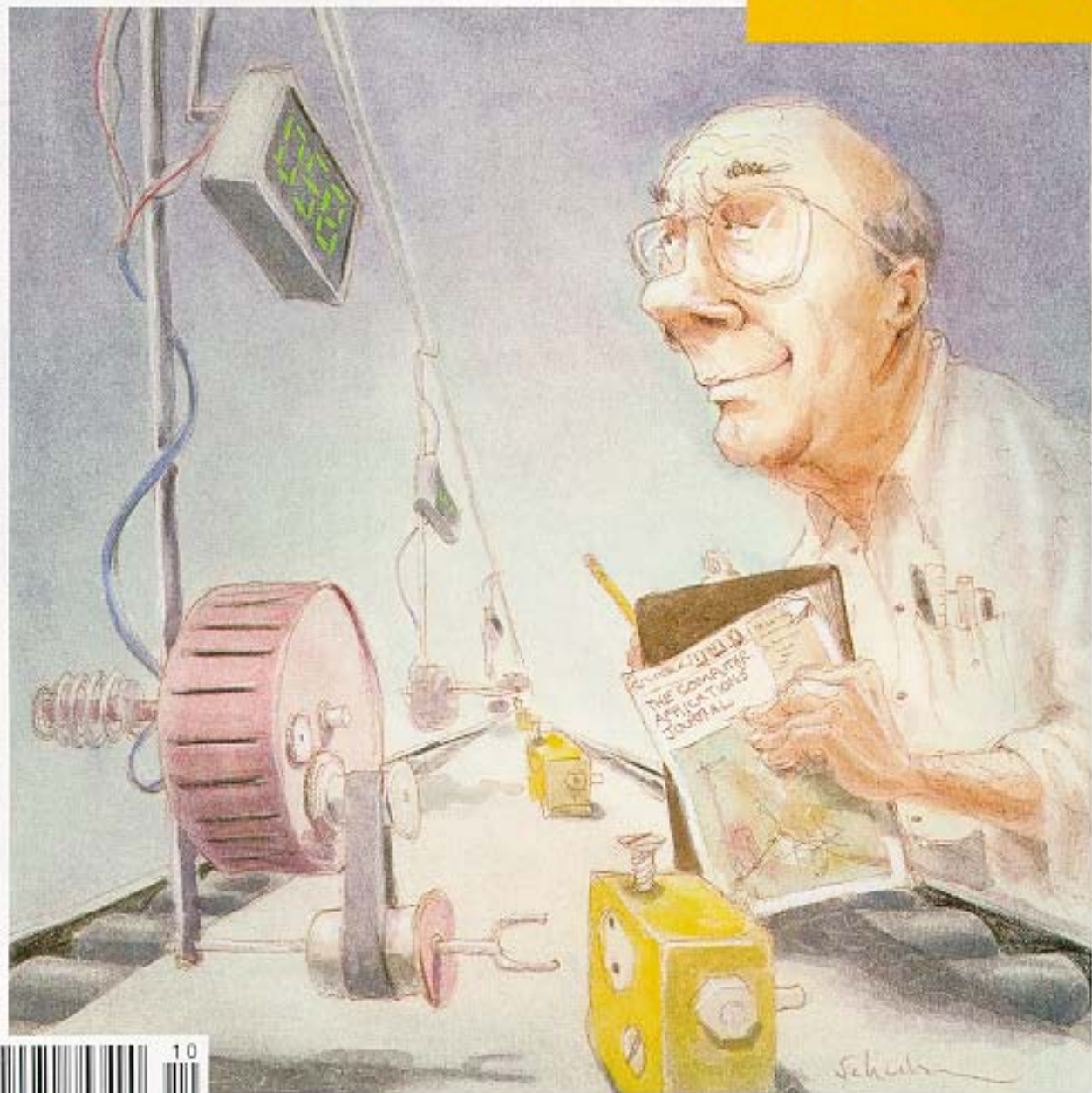
## INDUSTRIAL CONTROL

Servo Motor Control  
Techniques

Serial Data Line Monitor

ARM Processor Explored

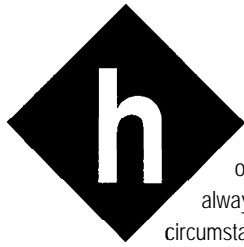
Recycling Old Floppy Drives



\$3.95 U.S.  
\$4.95 Canada

# EDITOR'S INK

## Breaking Old Habits



How often is it in industry that people stick to the old ways out of sheer comfort? "Because we've always done it this way" may have its place in some circumstances, but it has no business rearing its ugly head when a process or operation could be streamlined or improved simply by a liberal sprinkling of new technology.

For example, when this magazine was first started, article records were kept on index cards in a little file box. Cover sheets for in-house article folders and proofs were filled out by hand. Author contact notes were kept in the editors head. The system continued in this manner for far too long simply because that's the way it always had been done.

When I took over the task of handling author contacts, I developed a database with complete author information and all contacts I've had with the authors. I also programmed macros to automatically print out all in-house paperwork and form letters. Spending an afternoon computerizing something that should have been automated from day one has saved me countless hours of trying to manually organize an ever-growing task. I can instantly access upcoming issues to find out what's on tap. Appropriate fires can be lit in plenty of time.

Industrial automation needn't necessarily be heavy machinery and robotic arms. Anything that saves a company time, improves quality, and increases productivity certainly fits the mold.

While our first feature article this month doesn't deal with heavy-duty "industrial" motors, the servos and the interface described can be used in many light industrial settings. Just don't let the servo motors' normal application fool you.

Next, anybody who has ever tried to connect one or more devices serially knows the frustration of trying deal with the various "standards" out there. The Analyst 2 data line monitor is a hand-held diagnostic tool that can analyze and help debug all manner of serial connections wherever you're trying to make them.

While many of our articles deal with 8-bit processors, the world of embedded control certainly isn't limited to the little tykes. The 32-bit ARM RISC processor, originally developed in England, is finding its way into more and more embedded applications (such as Apple's Newton). The third feature article is an introduction to the ARM and its capabilities.

Finally, we have a neat little project that performs a rather mundane task: telling the time-of-day: However, this clock is unique in that it's intended to be used by those who are blind or visually impaired and need to know the time without using a mechanical voice.

In our columns, Ed continues his journey through the protected land with another look at the disk boot process; Jeff takes a trip to the junk heap with an eye toward salvaging parts from old floppy disk drives; Tom checks a novel new approach to using flash memory for mass storage on existing PCs; and John finishes up his DS80C320-based ec.32 embedded controller board.

CIRCUIT CELLAR **INK**®

## THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR  
Steve Ciarcia

EDITOR-IN-CHIEF  
Ken Davidson

TECHNICAL EDITOR  
Janice Marinelli

ENGINEERING STAFF  
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR  
Tom Cantrell

CONTRIBUTING EDITORS  
John Dybowski & Russ Reiss

NEW PRODUCTS EDITOR  
Harv Weiner

ART DIRECTOR  
Lisa Ferry

GRAPHIC ARTIST  
Joseph Quinlan

CONTRIBUTORS:  
Jon Elson  
Tim McDonough  
Frank Kuechmann  
Pellervo Kaskinen

PUBLISHER  
Daniel Rodrigues

PUBLISHER'S ASSISTANT  
Sue Hodge

CIRCULATION COORDINATOR  
Rose Mansella

CIRCULATION ASSISTANT  
Barbara Maleski

CIRCULATION CONSULTANT  
Gregory Spitzfaden

BUSINESS MANAGER  
Jeannette Walters

ADVERTISING COORDINATOR  
Dan Gorsky

CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published CircuitryCellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second class postage paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U.S.A. and possessions \$21.95, Canada/Mexico \$31.95, all other countries \$49.95. All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank. Direct subscription orders and subscription related questions to The Computer Applications Journal Subscriptions, P O Box 696, Holmes, PA 19043.9613 or call (600) 269.6301. POSTMASTER, Please send address changes to The Computer Applications Journal, Circulation Dept P O Box 696, Holmes, PA 19043.9613

Cover Illustration by Bob Schuchman  
PRINTED IN THE UNITED STATES

### HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

**NORTHEAST & MID-ATLANTIC**  
**Barbara Best**  
(908) 741-7744  
Fax: (908) 741-6823

**SOUTHEAST**  
**Christa Collins**  
(305) 966-3939  
Fax: (305) 9858457

**WEST COAST**  
**Barbara Jones & Shelley Rainey**  
(714) 540-3554  
Fax: (714) 540-7103

**MIDWEST**  
**Nanette Traetow**  
(708) 789-3080  
Fax: (708) 789-3082

Circuit Cellar BBS—24 Hrs. 300/1200/2400/9600/11.4k bps, 6 bits, no parity, 1 stop bit, (203) 871-1988; 24001 1600 bps Courier HST. (203) 871-0549

All programs and schematics in *Circuit Cellar INK* have been carefully reviewed and 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 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*.

Entire contents copyright © 1994 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.

- 14 The Juggler's Delight: PIC-based Controller for up to Eight Servos  
*by Scott Edwards*
- 22 Analyst 2 Data Line Monitor  
*by Bill Payne*
- 36 A RISC Designer's New Right ARM/Designing with the ARM Processor  
*by Art Sobel*
- 48 Feeling Out a Braille Digital Clock  
*by Wayne Thompson*
- 56  Firmware Furnace  
Journey to the Protected Land:  
Booting into Protected Mode  
*Ed Nisley*
- 64  From the Bench  
Celebrate National Cannibalism Week/Take Your Old Floppy Drives to Lunch  
*Jeff Bachiochi*
- 70  Silicon Update  
Flash of Inspiration  
*Tom Cantrell*
- 76  Embedded Techniques  
ec.32 Wrap Up  
*John Dybowski*

# INSIDE ISSUE 51

- 2 Editor's INK  
Ken Davidson  
Breaking Old Habits
- 6 Reader's INK  
Letters to the Editor
- 8 New Product News  
edited by Hat-v Weiner

- 84 **ConnectTime**  
Excerpts from  
the Circuit Cellar BBS  
conducted by  
Ken Davidson
- 96 **Steve's Own INK**  
Steve Ciarcia  
Of Patentable Value?
- 81 **Advertiser's Index**

# READER'S INK

## Precise Confusion

I just finished reading my article "Get Precise with the Précis A/D Converter" in the August issue. Now I'm writing to halt the spread of confusion (from which I obviously suffer) regarding byte swapping and Intel 80x86 I/O operations.

In the article, I made the statement, "By contrast, the Motorola 680xx architecture is big endian with respect to bytes, but little endian with respect to bits. However, it is perhaps less well known that 80x86 processors follow the Motorola scheme as far as I/O operations are concerned." The reason it is not well known is because the statement is wrong!

Although the code presented in the article is correct and works properly, my explanation of the I/O scheme was wrong. The correct explanation is that the architects of the 80x86 were consistent in their use of the little-endian scheme for both memory and I/O accesses. That is, the lower numeric address always contains the least-significant byte of a 16-bit word.

J. Conrad Hubert, St. Paul, MN

## Contacting Circuit Cellar

We at the Computer Applications Journal encourage communication between our readers and our staff, so have made every effort to make contacting us easy. We prefer electronic communications, but feel free to use any of the following:

Mail: Letters to the Editor may be sent to: Editor, The Computer Applications Journal, 4 Park St., Vernon, CT 06066.

Phone: Direct all subscription inquiries to (800) 269-6301.

Contact our editorial offices at (203) 8752199.

Fax: All faxes may be sent to (203) 872-2204.

BBS: All of our editors and regular authors frequent the Circuit Cellar BBS and are available to answer questions. Call (203) 871-1988 with your modem (300-14.4k bps, 8N1).

Internet: Electronic mail may also be sent to our editors and regular authors via the Internet. To determine a particular person's Internet address, use their name as it appears in the masthead or by-line, insert a period between their first and last names, and append "@circellar.com" to the end. For example, to send Internet E-mail to Jeff Bachiochi, address it to jeff.bachiochi@circellar.com. For more information, send E-mail to info@circellar.com.

TIRED OF WAITING FOR THE PROMPT ?

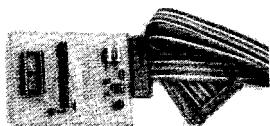
Speed up with a ROM DRIVE! Boots DOS and programs instantly. Also used to replace mechanical drive completely in controllers or diskless workstations. The only perfect protection from viruses. Easy to install half-size card.

VVDISK1 128k 575  
VVDISK2 720k 1150  
VVDISK3 288m 1195

**\$75**

Quantity discounts!

## DOS IN ROM!



## \$95 EPROM PROGRAMMER

- Super Fast Programming
- Easier to use than others
- Does 2764127080 (8 Meg)

WORLD'S SMALLEST PC !!!

ROBOTS ALARMS RECORDERS DOS

THREE EASY STEPS: **\$27** 1K QTY  
1. Develop on PC  
2. Download to SBC **\$95** SGL QTY  
3. Burn into EPROM

-2 PARALLEL -LCD INTERFACE  
-3 SERIAL -KEYBOARD INPUT  
-PC TYPE BUS -REAL TIME CLK  
-BIOS OPTION -BATTERY OR SV

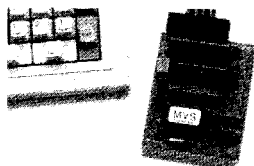
FREE SHIPPING IN U.S.

5 YEAR LIMITED WARRANTY



Box B50  
Merrimack, NH  
(508) 792 9507

## 8088 SINGLE BOARD COMPUTER



## BCC52 BASIC-52 COMPUTER/CONTROLLER

The BCC52 controller continues to be Micromint's best selling single-board computer. Its cost-effective architecture needs only a power supply and terminal to become a complete development system or single-board solution in an end-use system. The BCC52 is programmable in BASIC-52, (a fast, full floating point interpreted BASIC), or assembly language.

The BCC52 contains five RAM/ROM sockets, an "intelligent" 27641128 EPROM programmer, three b-bit parallel ports, an auto-baud rate detect serial console port, a serial printer port, and much more.

### PROCESSOR

- 80C528-bit CMOS processor w/BASIC-52
- Three 16-bit counter/timers
- Six interrupts
- Much more!

### Input/Output

- Console RS232 -autobaud detect
- Lineprinter RS-232
- Three 8-bit parallel ports
- EXPANDABLE!
- Compatible with 12 BCC expansion boards

### MEMORY

- 48K RAM/ROM, expandable
- Five on-board memory sockets
- Either 8K or 16K EPROM

To Order Call 1-800-635-3355

Tel: (203) 871-6170

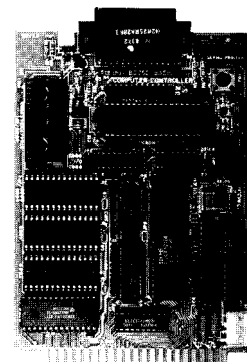
Fax: (203) 872-2204

BCC52	Controller board with BASIC-52 and 8K RAM	\$189.00	single Qty
BCC52C	Low-power CMOS version of the BCC52	\$199.00	
BCC52I	-40°C to +85°C industrial temperature version	\$294.00	
BCC52CX	Low-power CMOS, expanded BCC52 w/32K RAM	\$259.00	

CALL FOR OEM PRICING



MICROMINT, INC. 4 Park Street, Vernon, CT 06066  
in Europe: (44) 0285-658122 • in Canada, (514) 336-9426 • in Australia: (3) 467.7194  
Distributor Inquiries Welcome!



# NEW PRODUCT NEWS

Edited by Harv Weiner



## SERIAL EEPROM DESIGNER KIT

Microchip Technology is shipping a comprehensive **Serial EEPROM Designer's Kit** to assist engineers in creating robust and reliable embedded systems which use serial EEPROMs. The Kit provides all the hardware, PC-based software tools, and on-line reference materials

required by a design engineer to create, model, integrate, debug, and test systems incorporating serial EEPROMs.

The Kit includes SEEVAL, an advanced serial EEPROM programming and evaluation system. SEEVAL enables the designer to quickly learn serial EEPROM operational theory and usage, load an EEPROM with known data prior to a

test, modify part or all of an EEPROM array, save or restore data to or from a disk, and program special device features. SEEVAL accelerates the development and integration process.

Also

included in the Kit is a Total Endurance Model which models a serial EEPROM within a specific application environment and analyzes the impact of factors such as density, temperature, voltage, write modes, bytes per cycle, and cycles per day on erase/write endurance. A Serial EEPROM Handbook includes data sheets, application notes, and

qualification reports. Support materials such as a power supply, RS-232 cable, and chip samples are also included.

The Serial EEPROM Designer's Kit sells for \$149.

Microchip Technology, Inc.  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
(602) 786-7200  
Fax: (602) 899-9210

#500

## MACHINE VISION/IMAGE ANALYSIS

The **OVG-1** from Advanced Micro Systems is a compact, high-resolution graphics display generator capable of superimposing lines, circles, crosses, boxes [icons], and text on standard composite video signals such as those generated by closed-circuit cameras. High-resolution capability with digital position read-out, gives detail measurement and display in applications such as quality control, robotics, pattern recognition, pick and place, and targeting systems.

The OVG-1 accepts up to three composite video inputs such as TV cameras, VCR, demodulated received signals, or any source of RS-170, composite, interlaced video. Two independent video-overlay memories, each with a full resolution of 700 x 640 elements, are standard. In normal operation, only one is displayed at a given time, but both may be displayed to create a composite image. In addition, a terminal display option offers the capability of displaying ASCII alphanumeric characters with 35- or 70-column resolution.

Nine separate images may be stored in the image definition memory for recall and display on demand. Each image consists of a set of up to nine individually programmable icons. The nonvolatile memory feature supports powerup without the need to reprogram or calibrate.

The OVG-1 is capable of being interfaced to a host computer via the RS-232 serial port at 9600 bps. Single-line ASCII commands are used to set display parameters or read out data.

Pricing for the OVG-1 starts at under \$750.

Advanced Micro Systems, Inc.

2 Townsend West • Nashua, NH 03063-1277 • (603) 882-1447 • Fax: (603) 881-7600

#501

# NEW PRODUCT NEWS

## DMM WITH RS-232 PORT

Prairie Digital has introduced a low-cost, full-function True RMS (TRMS) Digital Multimeter with a serial port for both data output and control input. The **Model 150-02** features an RS-232 serial interface that offers remote control of the front panel from a host computer.

The Model 150-02 features TRMS AC voltage, TRMS AC current, DC current and voltage, resistance, frequency, continuity, and diode measurement capability. A full-scale count of 3999 (3¾ digits) results in twice the resolution of comparable 1999 (3%digit) meters. Remote RS-232 commands enable the host computer to request readings (with or without header information), change ranges (or select autorange), select AC/DC (ideal for power supply testing), enter or exit hold mode, and reinitialize. An optoisolated, 3-wire RS-232 interface offers over 2,000 V of isolation between the host PC and the signal being measured.

The Model 150-02 can be connected to any computer having an RS-232 port, and the unit comes with a PC-compatible cable and PC-DOS software. The data-logging software logs readings to the screen,

printer, or disk file. The interval is programmable, and readings can be time and date stamped.

Another program included with the 150-02 demonstrates how easily the unit can be controlled and read by a host computer. Source code is included for both programs.

The Model 150-02 sells for \$179 and is shipped complete with PC-compatible data-logging and demonstration software, PC cable, test leads, battery, and manual.

Prairie Digital, Inc.  
846 17th St.  
Industrial Park  
Prairie du Sac, WI 53578  
(608) 643-8599  
Fax: (608) 643-6754

#502

## MACINTOSH INTERFACE

The Mac65 is a new, low-cost interface that connects a Macintosh computer to external resources. The Mac65 plugs into the Macintosh SCSI port to achieve fast, real-time data transfer.

The interface accommodates four digital sensors (sampled at 10,000 times/s) and three analog sensors (sampled 100,000 times/s for one analog channel). Digital sensors include a photogate, laser switch, pulley-timing system, ultrasonic motion sensor, and Geiger counter. Analog sensors measure voltage, force, light, temperature, sound, pH, magnetic field strength, light absorbency (colorimetry), and heart rate.

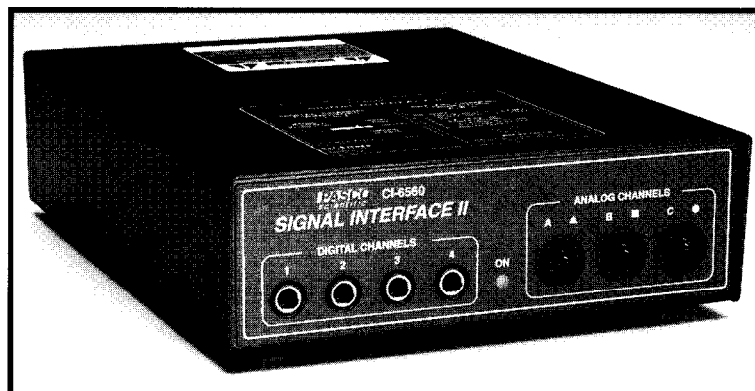
With the Mac65, a Macintosh can emulate lab equipment such as a digital voltmeter, electronic stopwatch, light meter, radiation counter, and a triple-trace storage oscilloscope. By combining an optional Power Amplifier with the Mac65, the Macintosh becomes a power supply ( $\pm 10$  V, supplying up to 1 A) or a function generator (up to 5 kHz).

The **Mac65 Science Workshop** software manipulates data using a variety of features. It displays up to three data sets on a single graph and offers one-click autoscaling to fit data into available graph space. Statistics tools include linear fit, integration, histogram, fast Fourier transform, mean, maximum, and so on. A smart cursor identifies points on a graph.

The Mac65 comes bundled with a floppy disk containing 50 science experiments, making it perfect for a school science lab.

Pasco Scientific  
10101 Foothills Blvd.  
P.O. Box 619011  
Roseville, CA 95678-9011  
(916) 786-3800  
Fax: (916) 786-8905  
Internet: [sales@pasco.com](mailto:sales@pasco.com)

#503



# NEW PRODUCT NEWS

## FAST KEY BOARD

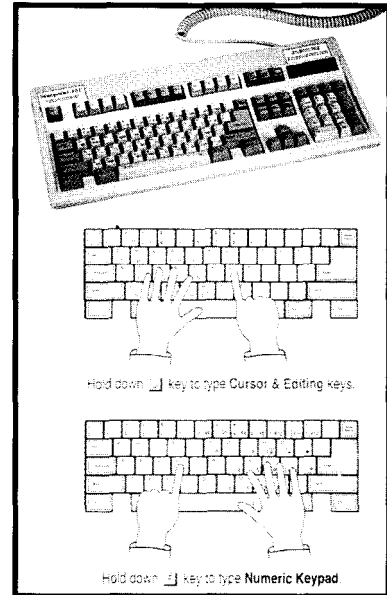
With **Starpoint-101**, a new computer keyboard marketed by Jefferson Computer Products, typists can perform all keyboard operations while keeping both hands on the letter keys in the traditional touch-typing position.

The Starpoint- is easy to operate and does not interfere with normal typing. If you hold down the "J" key with your right hand, the letter keys under your left hand instantly become cursor and editing keys. Hold the "F" key, and the letter keys under your right hand become a numeric keypad. This flexibility eliminates the need to lift your hands from the letter keys, reduces wrist and hand stress, and increases keyboard speed and productivity.

For Windows users, the keyboard also features a new kind of pointing device. The space bar can be used to open and select menu items from Menu Mouse, a pull-down menu. This keyboard mouse is much faster and easier than reaching for a regular mouse (or using the ALT key) and does not interfere with any other mouse or pointing device.

The Starpoint- is the first implementation of KeyStar, Jefferson's keyboard algorithm. The patented algorithm is available for licensing and is particularly suitable for notebook computers, subnotebook computers, and small-footprint-keyboard applications.

The Starpoint- keyboard sells for \$129.95 and is compatible with all PC, XT, AT, and PS/2 computers.



Jefferson Computer Products, Inc.

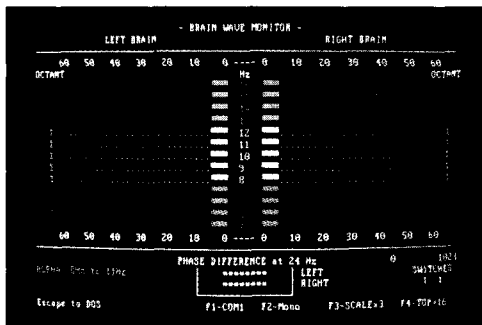
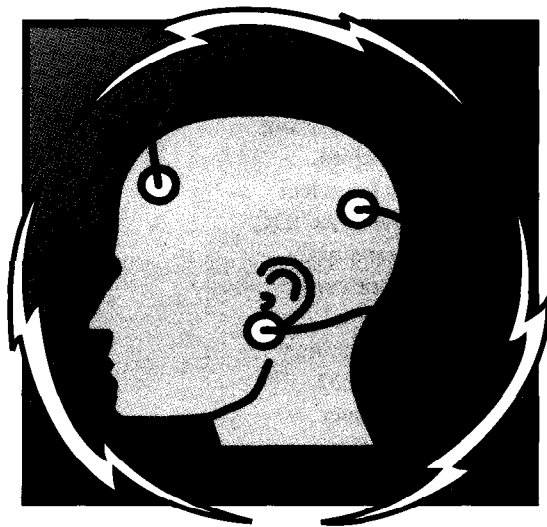
23454 25th Ave. S. • Seattle, WA 98198 • (206) 824-1111 • Fax: (206) 824-0941 • Internet: cwolfi@delphi.com

#504

# HAL-4

## EEG Biofeedback Brainwave Analyzer

The HAL-4 kit is a complete battery-operated a-channel electroencephalograph (EEG) which measures a mere 6" x 7". HAL is sensitive enough to even distinguish different conscious states-between concentrated mental activity and pleasant daydreaming. HAL gathers all relevant alpha, beta, and theta brainwave signals within the range of 4-20 Hz and presents it in a serial digitized format that can be easily recorded or analyzed. HAL's operation is straightforward. It samples four channels of analog brainwave data 64 times per second and transmits this digitized data serially to a PC at 4800 bps. There, using a Fast Fourier Transform to determine frequency, amplitude, and phase components, the results are graphically displayed in real time for each side of the brain.



**HAL-4 KIT..... NEW PACKAGE PRICE - \$279 +SHIPPING**  
Contains HAL-4 PCB and all circuit components, source code on PC diskette, serial connection cable, and four extra sets of disposable electrodes.

to order the HAL-4 Kit or to receive a catalog,  
CALL: (203) 8752751 OR FAX: (203) 8752204

CIRCUIT CELLAR KITS • 4 PARK STREET  
SUITE 12 • VERNON • CT 06066

• The Circuit Cellar Hemispheric Activation Level detector is presented as an engineering example of the design techniques used in acquiring brainwave signals. This Hemispheric Activation Level detector is not a medically approved device, no medical claims are made for this device, and it should not be used for medical diagnostic purposes. Furthermore, safe use requires HAL be battery operated only!

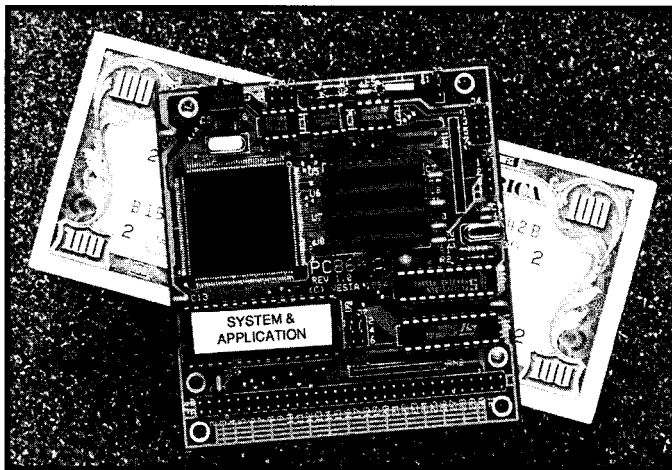
# NEW PRODUCT NEWS

## LOW COST PC

The PC86 from Vesta Technology is a low-cost PC/XT on a PC/104 bus board. It is an ideal engine for embedded applications requiring low power and DOS compatibility.

The PC86 supports the entire XT standard including keyboard input, 512 KB or 2 MB of DRAM, and extended BIOS-ROM space. Low-power operation (100 mA at 5 V) and small size (3.6" x 3.8") are key features of this PC/104 bus master. However, processing power is not compromised as the 8086 core communicates with 16-bit-wide DRAM at an 8-MHz clock rate, a performance matching an AT.

Enhancements to the XT standard are a real-time clock, an E<sup>2</sup>PROM, and an SPI/I<sup>2</sup>C bus interface for communication to offboard, low-power, synchronous serial peripherals. The BIOS, ROMed DOS, and user



application are all stored in a single EPROM (384 KB maximum), which can be mapped to upper memory in four selectable regions.

Support products include keypad-to-XT-keyboard protocol converter, PC-to-alphanumeric LCD controller, flash-memory module, and ISA-bus adapter.

Price for the PC86 is \$199 or \$99 at 1000 quantity. Price for the LCD86 or KEY86 is \$29, Flash86 is \$69, and Backplane86 is \$59.

**Vesta Technology, Inc.**  
7100 W. 44th Ave., Ste. 101  
Wheat Ridge, CO 80033  
(303) 422-8088 • Fax: (303) 422-9800

#505

## VIDEO MODEM

The CIS-100 Video Modem from Communications Specialists sends and receives broadcast-quality, single-frame color video over any narrow-band communication channel. Media such as private two-way radio (with or without repeater), SMR trunked, cellular or IMTS phone, satellite, or standard dial-up landline can be used.

Any NTSC-compatible video input devices such as video cameras, camcorders, VCRs, electronic still-image cameras, LD players, photo-CD players, or document cameras can be used. The output device can be displayed on an NTSC-compatible video monitor or several thousand single frames of video can be stored on any VCR. Video printers can generate hard copy if desired.

The CIS-100 can send single-frame video on command or at predetermined intervals from unattended remote locations. The video can also be sent to a remote answering machine for later playback and viewing or for retrieval via remote control from anywhere.

A telephone adapter cable interfaces the CIS-100 with a telephone for sending and receiving single-frame video. When video is sending, the handset is disconnected and automatically reconnected when video transmission is complete. A two-way radio adapter enables a radio to send and receive single-frame color

video. This cable can also be used with cellular phones.

The CIS-100 is 5.5" x 7" x 1.5" and requires 12 VDC at 400 mA. A mobile mounting bracket and 12-VDC power supply are included.

The CIS-100 sells for \$749.95. The telephone adapter cable sells for \$49.95 and the two-way adapter cable sells for \$29.95.

**Communications Specialists, Inc.**  
426 West Taft Ave. • Orange, CA 92665-4296  
(714) 998-3021 • Fax: (714) 974-3420

#506



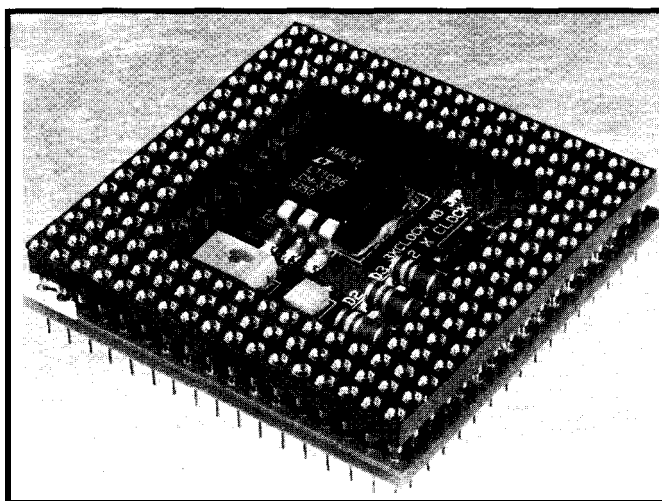


# NEW PRODUCT NEWS

## 486-UPGRADE MODULE

The **Aries Upgrade Socket** enables users of the 5-V i486DX/DX2 CPU to upgrade to the increased performance of Intel's high-speed, 3.3-V DX4 without changing motherboards.

The DX4 IC can run at 2x, 2.5x, or 3x the external clock circuitry and is available in speeds of 75, 83, and 100 MHz. It consumes relatively little power and has the same package pinout and footprint as the existing 486DX/DX2 CPUs. However, the DX4 requires a 3.3-V power supply. It cannot be used



on existing 5-V systems because of heat problems.

The Aries Upgrade Sockets, available for PGA or SQFP package CPUs, enable the DX4 to be used with the existing mother-

board. The modules regulate the voltage so that the 3.3-V chip can be used with the existing 5-V supply.

The 57-486DX2U Upgrade Socket for PGA packages permits placement

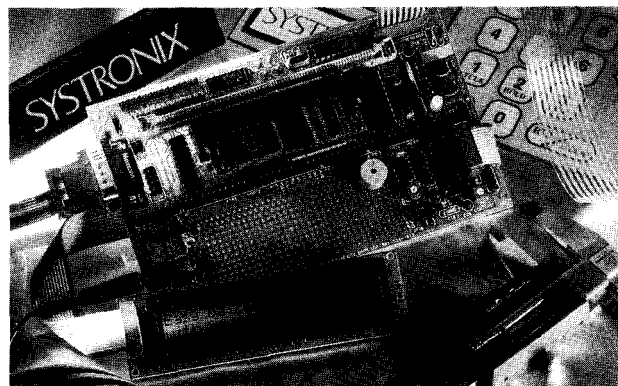
of the DX4 chip into the socket and does not require any soldering. The 57-486DX3U Upgrade Adapter for SQFP packages converts the SQFP package into a leaded PGA package by soldering the Intel package to the Aries adapter.

Pricing for the Upgrade Socket or Adapter is \$24.50 in 100 quantities.

Aries Electronics, Inc.  
P.O. Box 130  
Frenchtown, NJ 08825  
(908) 996-6841  
Fax: (908) 996-3891

#507

## NEW! UNIVERSAL DALLAS DEVELOPMENT SYSTEM from \$199!



- It's a complete single board computer!
- One board accommodates any 40 DIP DS5000, 40 SIMM DS2250, 40 SIMM DS2252, or 72 SIMM DS2251, 8051 superset processor! Snap one out, snap another in.
- Programs via PC serial port. Program lock & encrypt.
- LCD interface, keypad decoder, RS232 serial port, 8-bit ADC, four 300 mA 12V relay driver outputs.
- Power with 5VDC regulated or 6-13 VDC unregulated
- Large prototyping area, processor pins routed to headers
- Optional enclosures, keypads, LCDs, everything you need
- BC151 Pro BASIC Compiler w/50+ Dallas keywords \$399

**SYSTRONIX**® TEL: 801.534.1017 FAX: 801.534.1019  
555 South 300 East, Salt Lake City UT, USA 84111

### C COMPILERS CROSS ASSEMBLERS DEBUGGERS

68HC08	8051/52
6809	8080/8085
68HC11	8086/166
68HC16	8096/1196

Low Cost!! PC based cross development packages which include EVERYTHING you need to develop C and assembly language software for your choice of CPU.

- MICRO-C compiler, optimizer, and related utilities.
- Cross Assembler and related utilities.
- Hand coded (efficient ASM) standard library (source included).
- Resident monitor/debugger (source included)\*
- Includes text editor, telecomm software and many other utilities.

\*68HC08 and 68HC16 kits do not include monitor/debugger.

**Each Kit: \$99.95 + s&h (please specify CPU)**

**SPECIAL!!** *Super Developer's Kit*  
Includes all 8 kits above, plus additional assemblers for 6800, 6801/6803, and 6502. **Reg. \$400.00 NOW \$300.00**  
(Ask for SDK Special)

**Dunfield Development Systems**  
P.O. Box 31044 Nepean, Ont. K2B8S8  
CANADA  
Tel/BBS: 613-256-5820 Fax: 613-256-5821

# NEW PRODUCT NEWS

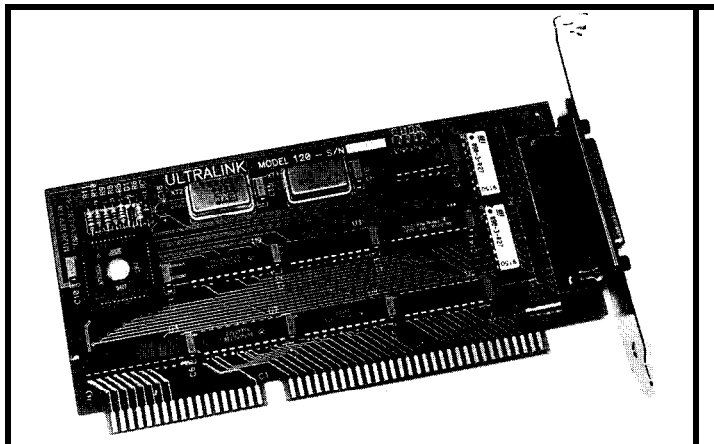
## SLOTLESS ISA BUS EXPANDER

Ultralink announces a simple, sophisticated expansion system for users of laptops, Micro Channel, and other non-ISA personal-computer platforms as well as PC users who are out of motherboard expansion slots. The **Model 120** consists of a PC card and cable which connect a passive ISA backplane to a PC's parallel printer port. Through this link, a wide range of data acquisition, control, and peripheral I/O cards available for the ISA bus can be installed without using internal expansion slots.

The Model 120 operates with Centronics-compatible and enhanced parallel-printer adapters at data rates up to 100 kbps. Both 8- and 16-bit ISA I/O data transfers are supported. Up to 16 ISA I/O cards can be installed in the passive backplane. The unit incorporates logic which expands the address space and interrupt levels available to the expansion backplane.

The Model 120 accommodates insertion or removal of cards from the expansion backplane without opening or shutting down the PC. As well, the expansion backplane is active only during data transfers with resident I/O cards. This reduces backplane electrical noise for sensitive data acquisition applications.

The Ultralink Model 120 sells for \$159.



Ultralink, Inc. • P.O. Box 1809 • Minden, NV 89423-1809 • (702) 782-9758 • Fax: (702) 782-2128

#508

## EXPRESS CIRCUITS

MANUFACTURERS OF PROTOTYPE PRINTED CIRCUITS FROM YOUR CAD DESIGNS

TURN AROUND TIMES AVAILABLE FROM 24 HRS — 2 WEEKS

### Special Support For:

- TANGO. PCB
- TANGO SERIES II
- TANGO PLUS
- PROTEL AUTOTRAX
- PROTEL EASYTRAX
- smARTWORK
- HiWIRE-Plus
- HiWIRE II
- EE DESIGNER I
- EE DESIGNER III
- ALL GERBER FORMATS

- FULL TIME MODEM
- GERBER PHOTO PLOTTING

WE CAN NOW WORK FROM  
YOUR EXISTING ARTWORK BY  
SCANNING. CALL FOR  
DETAILS!

*Express*  
*Circuits*

1150 Foster Street • P.O. Box 58  
Industrial Park Road  
Wilkesboro, NC 28697

Quotes:  
1-800-426-5396  
Phone: (910) 667-2100  
Fax: (910) 667-0487

## FEATURES

**14** The Juggler's Delight:  
PIC-based Controller for  
up to Eight Servos

**22** Analyst 2 Data Line  
Monitor

**36** A RISC Designer's  
New Right ARM

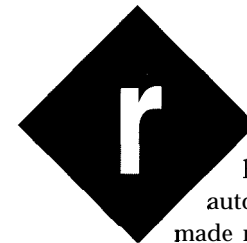
**48** Feeling Out a  
Braille Digital Clock

## FEATURE ARTICLE

Scott Edwards

# The Juggler's Delight: PIC-based Controller for up to Eight Servos

The engineer who looks beyond his or her own discipline for solutions to problems is far more successful than one who wears blinders. Here, motors designed for the R/C field find a more general-purpose use.



robotics and home/shop automation have made motion control a hot topic. You can't turn a page in a journal like this one without bumping into an H-bridge driver, shaft encoder, or control-loop algorithm. Of course, the really messy details of gearboxes, linkages, and pitiless real-world physics are left to the reader.

Wouldn't it be great to find a modular solution to small motion-control problems? Something cheap, lightweight, standardized, mass-produced, reliable.. Then if you could talk to a bunch of these miracle positioners over a standard serial port, you'd really have something.

This article will show you how to use hobby servos-the kind that steer radio-controlled models-in conjunction with a NC-based controller to provide up to 256 channels of motion control from a single serial port. Applications for the servo/controller combination include robotics, movie or theme-park special effects, test equipment, and home automation.

### THE BENEFITS

The servos I use here are typically used in radio-controlled airplanes.

However, their use doesn't have to be so limited. Their benefits include:

- \*Light weight-an obvious requirement for R/C aircraft.

- \*High power-aircraft modelers are building large aircraft these days, up to 1/4 scale. It takes muscle to throw a 15' biplane into a loop. R/C cars and sailboats also place considerable demands on a servo's mechanical power.

- Reliability-a typical R/C model costs over \$1000 in materials and hundreds of hours of painstaking labor; their builders are serious about reliability.

- \*Ruggedness-servos and electronics frequently survive a crash and are reinstalled in the next model. Even if the model doesn't crash, servos operate in an environment of temperature extremes, wicked solvents, and jarring shock and vibration.

- \*Simplicity-most R/Cers are not electronically sophisticated. They buy and assemble electronic modules—they expect them to work with a minimum of fuss.

- \*Versatility-servo systems can operate a model's control surfaces, throttle, retractable landing gear, bomb-bay doors, smoke generators, and myriad other gadgets. As a result, off-the-shelf mechanical linkages are available to support almost any conceivable kind of rotary or linear motion. Servos are available in general- and special-purpose configurations, as well.

- Low cost-mass production drives down the cost of servos, while keen competition between mail-order dealers and hobby shops keeps mark-ups paper-thin. A recent survey of ads in *Radio Control Modeler* magazine turned up a deal on decent midrange servos: 8 for \$99.95.

## HOW SERVOS WORK

Let me explain how servos work from a black-box standpoint. There are three leads: two for power and one for

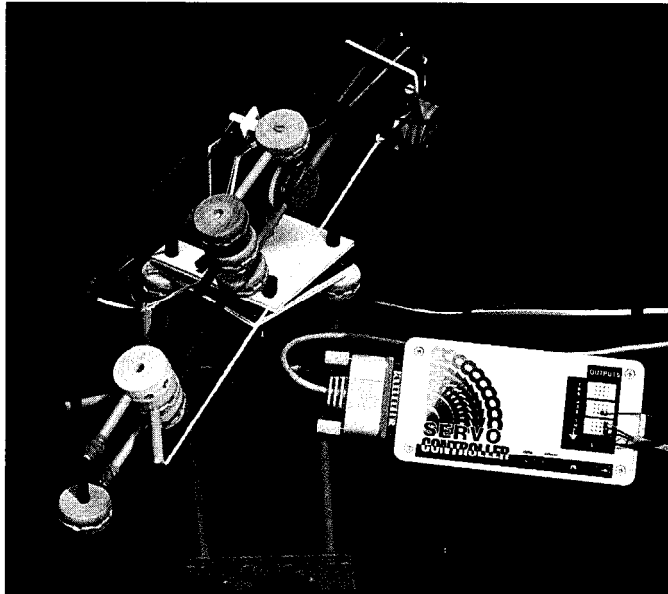


Photo 1-Tinkertoys are used to illustrate the capabilities of R/C servos in a three-axis control system.

signal. Connect the power leads to a hefty source of 4.8 VDC (this source can be the unregulated output of four dry-cell or NiCd batteries). Connect the signal lead to a source of variable-width pulses ranging from 1 to 2 ms in duration and repeating every 12–20 ms. The servo's output shaft will rotate to a position proportional to the input pulse width. Although servos' exact pulse-width-to-position relationships are not standardized, an input train of 1.5ms pulses generally moves the servo to the center of its output range (see Figure 1).

The servo works by comparing the width of input pulses to the width of pulses generated by an internal timer. The timer's period is controlled by a pot coupled to the servo's output shaft. The difference between the input and internal pulse widths serves as an error signal. Servo logic determines what direction to turn the motor to reduce the error signal, and turns on the appropriate

output drivers. The motor turns, changing the position of the feedback pot. When the next input pulse arrives, the servo performs the comparison again.

At some point, the motor's position reduces the error signal to an acceptable level, usually close to a 5- $\mu$ s difference between input and internal pulses. This corresponds to a fraction of a degree of servo travel. When the error is in this range, known as the *guard band*, the servo turns off its motor drivers. If it did not, the electronics would continue to try to cancel out the minuscule

error, and would drive the motor back and forth in a motion known as *hunting*.

There are two ways to look at this kind of control scheme. From the controller's standpoint, it's an open-loop system. There's no feedback from the servo to the pulse-generating controller. On the local level, however, it's a closed-loop system. The servo electronics are constantly trying to eliminate the difference between the commanded and actual position.

This split personality is a very convenient characteristic. The servo requires a minimum of attention from the controller, but still actively resists

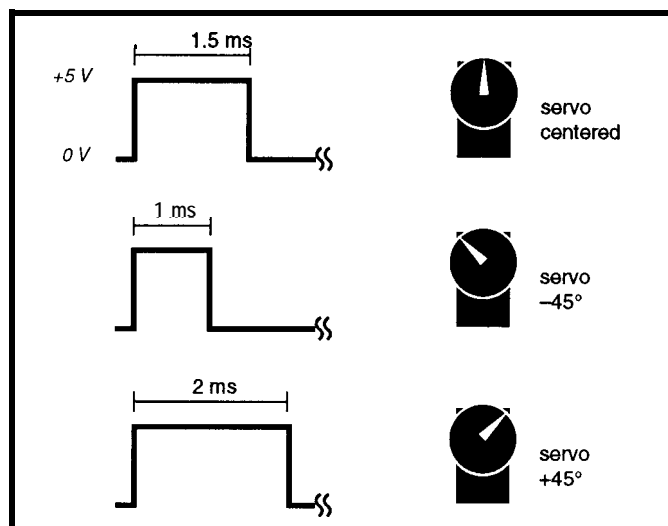


Figure 1-A train of positive pulses, repeated at a 40- to 100-Hz rate, controls servo position. Typical servos have a total range of 90°, but "retract" servos travel 180° or more. Most servos have position resolution of better than 0.5°.

and corrects for external influences that might try to move the servo away from its commanded position.

Although servos are almost ideal positioners in stock form, they are also easy to modify for special applications. You can alter the pot feedback circuit to change the servo's range of travel. Remove the pot linkage and any mechanical stops, and you have a gearhead motor whose direction of rotation is controlled by a servo pulse train. Apply analog signals to the feedback circuit, and you can mix the effects of the digital remote-control signal with a local analog adjustment such as temperature compensation. Drive a jackscrew mechanism and use a slide (linear-motion) pot for feedback, and you have a low-speed, high-torque linear actuator. You get the idea.

## THE SERIAL SERVO CONTROLLER

Any micro worth its salt can generate pulse-width-modulated signals for servo control. I have used a variety of servo-driver programs to demonstrate Parallax's BASIC Stamp



Photo 2—The servo arm in action shows what the controller can do.

SBC in articles and application notes. The idea of something mechanical moving with power and precision under the control of the tiny Stamp really impresses people. The truth is, however, nothing could be simpler.

Although generating the servo signals in software is attractive from a minimalist standpoint, it's not always the best way to go. Your computer or controller may have business to attend to other than the simple-minded generation of a bunch of pulses. With that in mind, I designed a Serial Servo Controller (SSC) that uses a pair of PICs to receive commands over a serial link and convert them into servo-control signals. The SSC is serially

addressable, with a three-bit code representing servos 0-7, and a five-bit code representing controllers 0-31. This allows you to control up to 256 servos from a single serial port. The controller outputs servo pulses from 1 to 2 ms in duration with a resolution of 4  $\mu$ s. Since the guard band for most servos is 5  $\mu$ s, this gives you maximum position resolution with only a 1-byte position value.

The data format includes extra bytes that help the controller synchronize a stream of serial commands. This lets you add and subtract controllers to and from a working system without major foul-ups. The controller supports data rates of 4800, 9600, and 19,200 bps. It is always ready to accept new data; there's no handshaking or buffering involved. The maximum delay between receipt of new position data and change in the appropriate servo's control pulse width is one servo frame—approximately 20 ms.

Figure 2 is a schematic of the controller. It bucks the trend of PIC applications presented here in *Computer Applications Journal*. Instead of

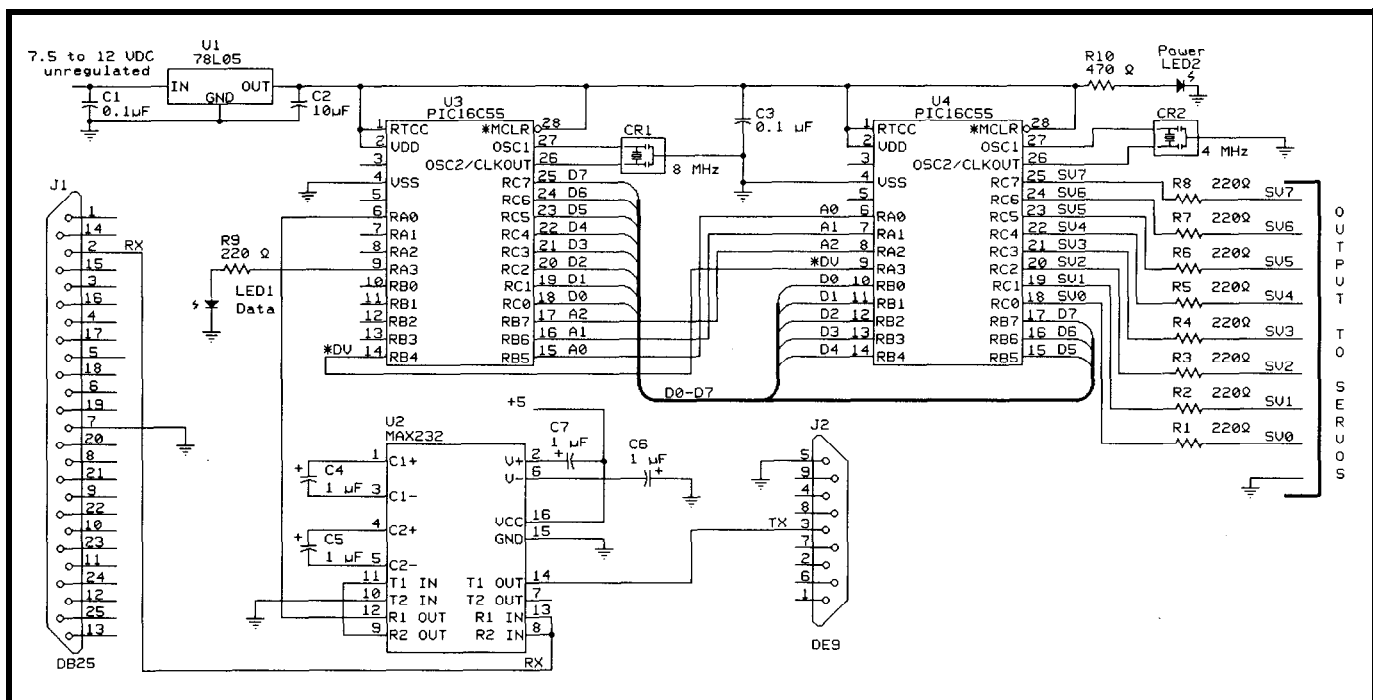


Figure 2—By using two PICs to share the workload (one for serial communications and another for motor control), code development is simplified.

## Servo Specs and Modifications

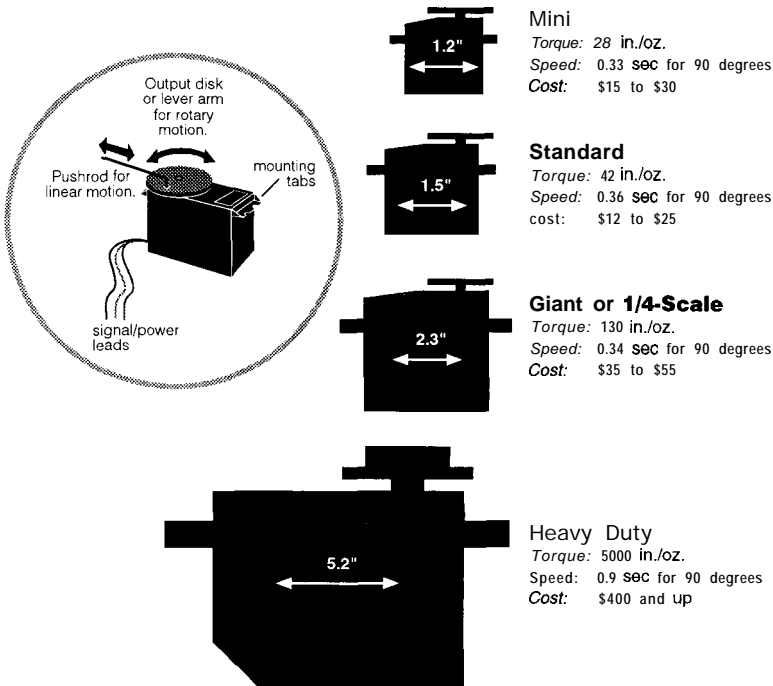
The accompanying drawing is a gallery of typical servo sizes and capabilities. Although they vary widely in torque, speed, and size, most servos are designed for 90° travel. You can overcome this limitation in most cases, but there's a right way and a wrong way to do it.

One of the common mistakes made by those unfamiliar with servos is to assume that because you can twist the servo's output shaft through 180° that you can also command the servo to move through 180°. It isn't so. Sure, their analog electronics can generally be tricked into moving beyond 90° with pulses wider or narrower than the nominal 1-2-ms range. But, this is generally unwise since many servos develop a bad case of the shakes when fed out-of-spec pulses.

What should you do if you need more travel than your servo is designed to provide? Most radio-control experts agree that the best course of action is to modify the feedback-pot circuit. This may sound daunting, given the small size of the servo and its close quarters, but the pot is close to the top of the housing and is designed to be accessed for occasional maintenance.

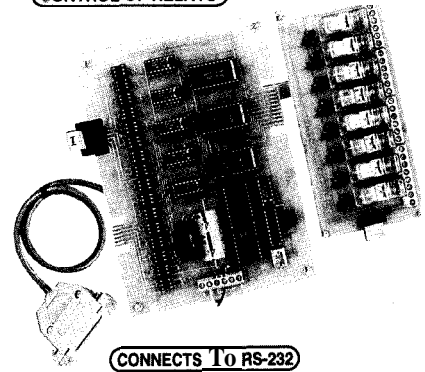
Here's how it's done—remove the long screws that hold the servo housing together and slide the pieces of housing apart slowly. Try to remove the portion of the housing that covers the output-shaft end without disturbing the mechanism inside. When you have the cover off, you should immediately recognize the feedback pot by the three wires trailing from it to a tiny, tightly packed circuit board. Make note of the arrangement of the pot wires, then cut the two wires connected to the ends of the pot's conductive track. Leave the wiper connection intact.

Add a 1.5-2.2 kΩ resistor in series with each end of the pot and reassemble the servo. Now, with nominal servo pulses you'll get increased travel. Make sure that you haven't extended the servo's range too far. If it hits the mechanical stop pins, it may shear them off, damaging the gear train.



## RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

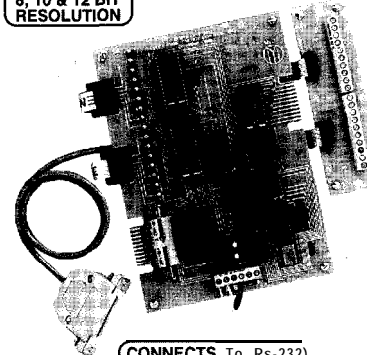


CONNECTS TO RS-232

AR-16 RELAY INTERFACE (16 channel),..... \$ 89.95  
Two 8 channel (TTL level) outputs are provided for connection to relay cards or other devices (expandable to 128 relays using EX-16 expansion cards). A variety of relays cards and relays are stocked. Call for more info.  
AR-2 RELAY INTERFACE (2 relays, 10 amp)....\$ 44.95  
RD-8 REED RELAY CARD (6 relays, 10 VA) ..... \$49.95  
RH-6 RELAY CARD (10 amp SPDT, 277 VAC)...\$ 69.95

## ANALOG TO DIGITAL

8, 10 & 12 BIT RESOLUTION



CONNECTS TO RS-232

ADC-16 A/D CONVERTER\* (16 channel/8bit)...\$ 99.95  
ADC-8G A/D CONVERTER\* (6 channel/10bit)...\$124.90  
Input voltage, amperage, pressure, energy usage, joysticks and a wide variety of other types of analog signals. RS-422/RS-485 available (lengths to 4,000'). Call for info on other A/D configurations and 12 bit converters (terminal block and cable sold separately).  
ADCIE TEMPERATURE INTERFACE\* (6 ch)...\$ 99.95  
Includes term. block & 8 temp. sensors (40° to 146° F).  
STA-6 DIGITAL INTERFACE\* (6 channel).....% 99.95  
Input on/off status of relays, switches, HVAC equipment, security devices, smoke detectors, and other devices.  
STA-8D TOUCH TONE INTERFACE\* \$ 134.90  
Allows callers to select control functions from any phone.  
PS-4 PORT SELECTOR (4 channels RS-422)...\$ 79.95  
Converts an RS-232 port into 4 selectable RS-422 ports.  
CO-485 (RS-232 to RS-422/RS-485 converter).....\$ 44.95

\*EXPANDABLE—expand your interface to control and monitor up to 512 relays, up to 576 digital inputs, up to 128 analog inputs or up to 128 temperature inputs using the PS-4, EX-16, ST-32 & AD-16 expansion cards.

FULL TECHNICAL SUPPORT...provided over the telephone by our staff Technical reference & disk including test software & programming examples in Basic, C and assembly are provided with each order

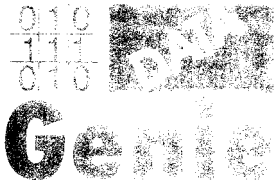
HIGH RELIABILITY...engineered for continuous 24 hour industrial applications with 10 years of proven performance in the energy management field.

CONNECTS TO RS-232, RS-422 or RS-485...use with IBM and compatibles, Mac and most computers. All standard baud rates and protocols (50 to 19,200 baud). Use our 600 number to order FREE INFORMATION PACKET. Technical information (614) 464-4470.

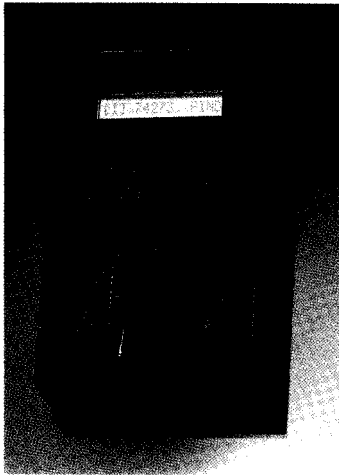
24 HOUR ORDER LINE (800) 842-7714  
Visa-Mastercard-American Express-COD

International & Domestic FAX (614) 464-9656  
Use for information, technical support & orders.

ELECTRONIC ENERGY CONTROL, INC.  
380 South Fifth Street, Suite 604  
Columbus, Ohio 43215-5438



Data Genie offers a full line of test & measurement equipment that's innovative, reliable and very affordable. The "Express Series" of stand-alone, non-PC based testers are the ultimate in portability when running from either battery or AC power. Data Genie products will be setting the standards for quality on the bench or in the field for years to come.

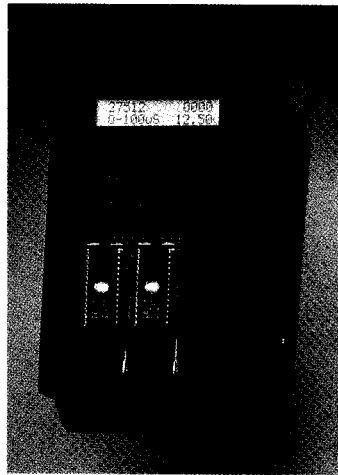


**HT-28 Express**

**LOGIC TESTER**

The HT-28 is a very convenient way of testing Logic IC's and DRAM's. Tests most TTL 74, CMOS 40/45 and DRAM's 4164-414000, 44164-441000. It can also identify unknown IC numbers on TTL 74 and CMOS 40/45 series with the 'Auto-Search' feature.

\$189.95

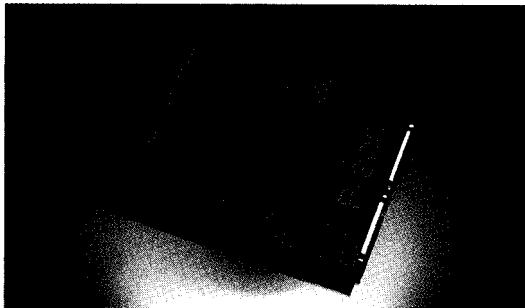


**HT-14 Express**

**EPROM PROGRAMMER**

The HT-14 is one-to-one EPROM writer with a super fast programming speed that supports devices from 27328 to 27080, with eight selectable programming algorithms and six programming power (VPP) selections.

\$289.95



**P-300**

**PC EXPANSION CARD**

The Data Genie P-300 is a useful device that allows you to quickly install add-on cards or to test prototype circuits for your PC externally. Without having to turn off your computer to install an add-on card, the P-300 maintains complete protection for your motherboard via the built-in current limit fuses.

\$349.95

**MING**  
Microsystems  
Division of MING & P, INC.<sup>TM</sup>  
17921 Rowland Street  
City of Industry, CA 91748  
TEL: (818) 912-7756  
FAX: (818) 912-9598

Call for a dealer near you.  
**1-800-473-6606**

Data Genie products are backed by a full year limited factory warranty

packing functions worthy of two PICs (or a fancier controller] into one, my circuit divides responsibilities that could be handled by a single PIC into two. I designed the SSC as a kit for the educational market, not for mass production. I felt that the cost of an additional PIC 16C55 (about \$5) multiplied by expected sales of controller kits (a hundred or so) didn't justify the programming effort required to merge the two functions into one chip. Using straightforward programming techniques (described in the next section), I had a prototype working within a few hours.

As long as I was throwing money around, I elected to use a MAX232 chip in an unusual way. One receiver section converts incoming  $\pm 10$ -volt RS-232 serial signals into inverted, logic-level signals that go to the serial-receive PIC. The other MAX232 receiver section feeds its logic-level output into one of the transmitter sections. The signal gets converted right back into its original  $\pm 10$ -volt form. This reconstituted serial signal goes to an AT-style RS-232 connector (DE9 male) that I call the *daisy-chain connector*.

This setup lets each SSC generate a fresh copy of the RS-232 serial signal to be passed on to the next SSC in the chain. It's not as elegant or robust as real multidrop schemes, but it is transparent and easy to use.

### SSC FIRMWARE

As I mentioned, I traded the elegance of a single-chip design for the programming ease of using two PICs. This allowed me to program and debug the chips separately, and then connect them together via the 12-wire interface shown in the schematic. I started with the serial-receive chip.

Since the PIC doesn't have a serial port, firmware takes care of serial reception. With an 8-MHz clock, the routine expects a data rate of 19,200 bps. To change the data rate, the user merely substitutes a different clock resonator: 4 MHz for 9600 bps or 2 MHz for 4800 bps. The parallel interface with the servo-controller PIC is asynchronous within this range of clock speeds.

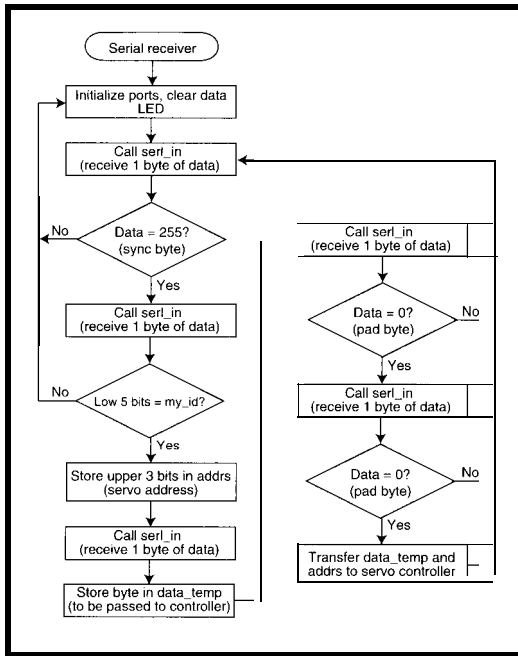


Figure 3—The serial receiver PIC handles all communications with the outside world.

Figure 3 is a flowchart of the overall program. As the figure shows, the body of the program has a ladder-like structure. Failure to receive a required element of the 5-byte data package causes the program to fall back and wait for a sync byte.

Unless the PIC receives a sync/start byte (FFh) and its own ID number (0 through 1Fh—assigned when the program is committed to EPROM) in immediate succession, it keeps watching the serial input for a sync byte. Once sync and ID are received, the next byte contains servo timing data, destined for the servo number specified by the upper three bits of the ID number. Servo data can range from 1 to FEh, representing pulse widths from just under 1 ms to just over 2 ms—the nominal legal range for most servos. (Since 0 and FFh have special significance as sync bytes, they aren't legal data values.) After accepting the servo data, the receiver looks for two pad bytes (both 0s) to complete the transaction.

Once the receiver has accepted the data, it puts a high on the data-valid (DV) bit, and then writes the data to port RC. It writes the address to the upper three bits of port RB. Once the data and address are in place, the receiver clears DV and leaves it cleared until new data arrives.

The purpose of the DV bit is to lock out the servo-controller PIC during the two-step address/data write. Otherwise, the servo controller might pick up data for one servo and pair it with another servo's address.

The servo controller (see Figure 4) generates the eight output pulses sequentially, with brief delays between pulses, and a 4-ms delay after all pulses have been sent, which is known as the end of the "frame" in servo parlance. I chose this scheme because it mimics the operation of radio-control transmitters. I haven't had the occasion to send SSC-generated signals over the air, but it would be nice not to have to reprogram it to do so.

During the delays, the controller PIC checks the DV line. If it's low, the servo controller reads in the data and address, then rechecks DV. If it is still low, the controller writes the data to the corresponding servo's pulse-timing register within the buffer. If DV is

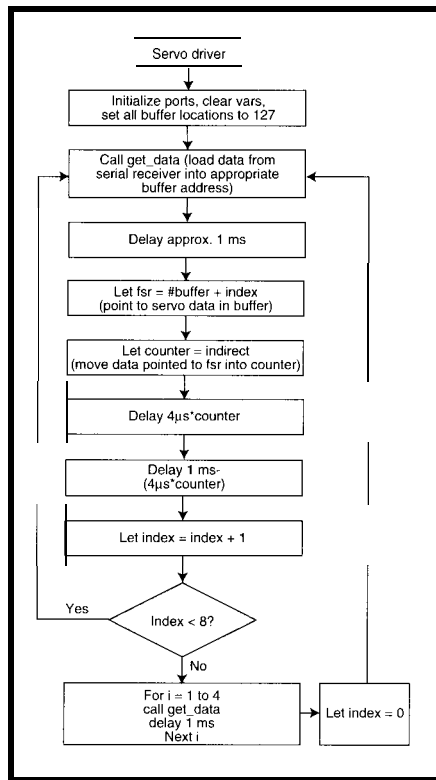


Figure 4—The servo driver PIC can concentrate on the motor control side of things while the other PIC handles the serial.

# 188SBC

Use Turbo or MS 'C

Intel 80C188XL

Two 1 meg Flash/ ROM sockets

Four battery backed, 1 meg RAM

16 channel, 12 or 16 bit A/D

8 channel, 12 bit D/A

2 RS-232/485 serial, 1 parallel

24 bits of opto rack compatible I/O

20 bits of digital I/O

Real-time clock

Interrupt and DMA controller

8 bit, PC/104 expansion ISA bus

Power on the quiet, 4 layer board is provided by a switcher with watchdog and power fail interrupt circuitry.

The 188SBC is available with Extended Interface Emulation of I/O - a Xilinx Field Programmable Gate Array and a breadboard area. Define and design nearly any extra Interface you need - we'll help! 188SBC prices start at \$299.

Call right now for a brochure!

# 552SBC

The 80C552 is an 8051 with:

8 ch. 10 bit A/D 2 PWM outputs

Cap/cmp registers 16 I/O lines

RS-232 port Watchdog

We've made the 552SBC by adding:

2-RS-232/485 multi-drop ports

24 more I/O Real-time Clock

EEPROM 3-RAM & 1-ROM

Battery Backup Power Regulation

Power Fail Int. Expansion Bus

Start with the Development Board all the peripherals, power supply, manual and a debug monitor for only \$349. Download your code and debug it right on this SBC. Then use OEM boards from \$149.

## True Low-cost In-circuit Emulation

The DryICE Plus is a low-cost alternative to conventional ICE products. Load, single step, Interrogate, disasm, execute to breakpoint. Only \$448 with a pod. For the 8051 family, including Philips and Siemens derivatives. Call for brochure!

**8031SBC as low as \$49**

Call for your custom product needs. Quick Response.



**HTE** HiTech Equipment Corp.  
9400 Activity Road  
San Diego, CA 92126  
(Fax: (619) 530-1458)

Since 1983

(619) 566-1892

70662.1241 @compuserve.com



high, indicating that the data or address may have changed while the controller was reading it, the PIC discards the data.

The controller gets a minimum of two opportunities to read each received servo address/data pair, since the fastest that new data can arrive is every 2.6 ms (5 bytes x 10 bits x 1/19,200 bps = 2.6 ms), and the receiver samples every 2 ms (maximum). Even if the controller discards a servo command, it either has already received it, or is about to receive it again, so no harm is done.

## USING THE SSC

Any computer with a serial port and a programming language to access it can send commands to the SSC. Listing 1 is a QBASIC program that sends properly formatted data to the controller. The only real point of interest in this demo program is that it filters out position values of 0 and 255 to preserve the significance of these numbers as synchronization bytes.

This kind of program—one that permits you to instantly command

large changes in position—is suitable only for lightweight and/or well-damped servo loads. Heavy, springy, lever-arm loads, like the Tinkertoy arm (see Photos 1 and 2), don't take kindly to abrupt acceleration and deceleration. I have found that adding some intermediate positions allows the arm to start and stop much more smoothly. I have written a simple motion recording and playback program for the Macintosh that requires me to enter these points manually. Adding support for automatic "smoothing" of motion paths is definitely on my to-do list.

Even without automatic smoothing, I was able to point-and-click my way through a three-axis servo sequence that had the Tinker arm picking up an object, moving it to a new position a foot away, setting it down, moving away from the new position, and then moving back to pick up the object and return it to its original spot. By looping this sequence, I had the arm moving the object back and forth for an hour without a fumble. ☐

**Scott Edwards operates Scott Edwards Electronics, specializing in high-tech documentation including manuals and application notes. He recently completed The PIC Source Book, a book on PIC assembly language programming. You may reach him at (602) 459-4802, (602) 459-0623 (fax), or 72037.2612@compuserve.com.**

## SOURCE

Light-duty servos:

ACE R/C  
116 W. 19th St.  
P.O. Box 511  
Higginville, MO 64037  
(800) 322-7121  
(816) 584-6303

Heavy-duty servos:

Condor R/C Specialties  
1733 Monrovia Ave., Unit G  
Costa Mesa, CA 92627  
(714) 642-8020  
Fax: (714) 642-8021

A partial kit for the Serial Servo controller with all ICs, ceramic resonators, and PC board is available for \$50 postpaid from:

Scott Edwards Electronics  
964 Cactus Wren Lane  
Sierra Vista, AZ 85635

Check, money order, or school purchase order only, please. Specify serial data rate (4800, 9600, or 19,200 bps) and ID number (O-31) for the controller. Printed circuit boards are available for those who wish to program their own PICs. Price is \$10 postpaid.

## SOFTWARE

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

## I R S

401 Very Useful  
402 Moderately Useful  
403 Not Useful

Listing 1—A sample QBASIC program generates commands for the serial servo controller.

```
DEFINT A-Z
Sync.byte = 255
Pad = 0

'In the line below, be sure that the baud rate (19,200 in this
'example) is correct. It is proportional to the frequency of CR1
'as follows: 8 MHz-19200, 4 MHz-9600, 2 MHz-4800

OPEN "com1:19200,N,8,1,CDO,CSO,DSO,OP0" FOR OUTPUT AS #1
CLS
PRINT "SERIAL SERVO CONTROLLER": PRINT : PRINT
PRINT "At the prompt, type in the board number (0 to 32), a comma,"
PRINT "the servo number (0 to 7), a comma, and a position value
      (1 to 254)."
```

```
PRINT "Press <CNTRL> <Break> to end."
Again:
LOCATE 8, 1
PRINT "
      LOCATE 8, 1
      INPUT "Servo,position>", Board.ID, Servo, Position
      'Perform some basic error trapping
      IF Servo > 7 THEN Servo = 7
      IF Servo < 0 THEN Servo = 0
      IF Position > 254 THEN Position = 254
      IF Position < 1 THEN Position = 1
      Servo = (Servo * 32) + Board.ID
      PRINT #1, CHR$(Sync.byte); CHR$(Servo); CHR$(Position);
      CHR$(Pad); CHR$(Pad);
      GOTO Again
CLOSE
```

# Analyst 2 Data Line Monitor

## FEATURE ARTICLE

Bill Payne



I cannot recount how many times I have had trouble trying to troubleshoot a

PC's serial data link. Whether the external device is a modem, a printer, a serial-to-parallel converter, or another computer, the problem is that I have needed to see the actual data flowing between the two devices. I have tried to use a breakout box with LED indicators, but it only works for slowly changing control-signal leads. If the problem is caused by an incorrect transmission rate, fast glitches occurring on the control-signal leads, or a mismatched character set, simple tools will not do the job. Such problems force me to pull out my Analyst 2 Data Line Monitor (Analyst 2) from the desk drawer.

In the field of data communications, there are protocol analyzers and there are data line monitors. Protocol analyzers are very sophisticated devices which enable a user to monitor the transmission on a serial communications link and to break down that transmission into the specific parts of the software protocol. A data line monitor passively bridges an RS-232 communications link, capturing and then displaying the serial data stream in a humanly intelligible form.

Analyst 2 can be used to monitor all asynchronous, synchronous, and user-defined, bit-oriented communications systems at speeds up to 38,400 bps. During the capture of data, it can switch the display of the serial data stream from a character-oriented format to a hexadecimal format by pressing a single key. I can set up Analyst 2 to look for specific charac-

ters before it starts or stops capturing the serial data stream. I can set it up to monitor specific control-signal leads for a transition to control capturing the stream or to stop it when a transmission error occurs.

Analyst 2 has the ability to perform signal distortion analysis, which checks the communications line for the relationship between the actual and theoretical signal durations. Signal distortion can occur in any communications system if the reference clock for the transmitter or receiver becomes skewed in any way due to age, improper frequency selection, or any of a dozen other factors. Analyst 2 can measure all types of signal distortion on a communications line.

Analyst 2 has the ability to measure the time delay between various hardware-handshake signal leads on the DB-25 interface. In this mode, Analyst 2 will actively stimulate the request-to-send (RTS) signal lead and wait for a corresponding response on the selected signal lead. clear-to-send (CTS) delay times for turning on and off can be measured relative to the assertion/deassertion of the RTS signal lead. Also, the time between the deassertion of the RTS signal lead until the detection of the carrier detect (CD) signal can be measured. This time duration is of paramount importance in half-duplex communications systems where it is commonly referred to as *turn around time*.

Analyst 2 also has the ability to simulate raw data traffic in all communications modes. This function can be used to validate the integrity of a communications line under test by inserting a known message into it. It can be used to validate the performance of printers, modems, and even software communications programs. Analyst 2 accomplishes this by outputting the Quick Brown Fox (QBF) message. The QBF message contains all the letters of the alphabet, numbers 0-9, a carriage return, and a line feed. The message can be transmitted in the ASCII, EBCDIC, or Baudot character sets. Analyst 2 pauses approximately 250 ms at the

You make your serial connections, run the appropriate software on both ends, and nothing happens. What do you do next? If you have an Analyst 2 data line monitor, you'll have your connection buzzing in no time.

end of each transmission before repeating the pattern.

The most powerful feature of Analyst 2 is the ability to review the captured serial data stream at the user's convenience. This is accomplished through the use of battery-backed memory for storing the captured data. Using the Review mode, I can change the display parameters at any time without affecting the capture memory contents. For example, I may have originally specified that the serial data being captured was transmitted least-significant bit first. Through Review mode, I can change the displayed bit order to most-significant bit first without affecting the original serial data in the nonvolatile capture memory. While reviewing the captured data, I can switch between the selected and hexadecimal character set with a single keystroke.

## HARDWARE DESCRIPTION

The complete schematic for the Analyst 2 is shown in Figure 1.

From the outside, you see the Analyst 2's user interface, which is made up of a 2-line, 32-character backlit LCD, 8 tricolor LEDs, and an 8-button keypad. The 8 LEDs represent the various EIA RS-232 interface leads in real-time for data terminal equipment (DTE). The following signal leads are monitored and displayed:

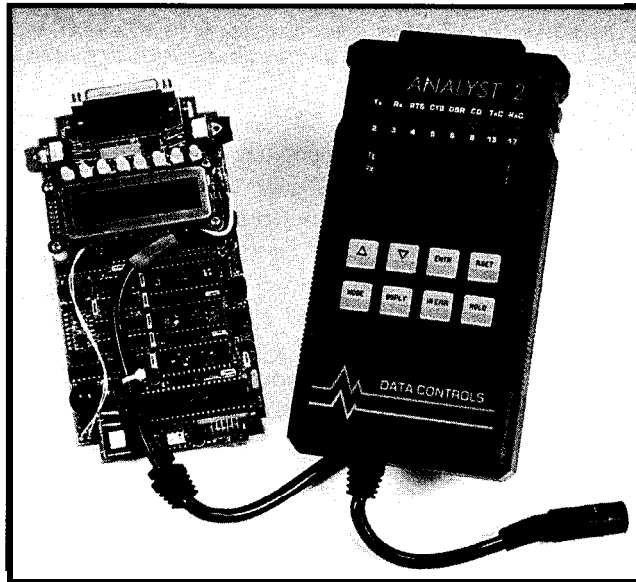
- Pin 2: Transmit Data (TxD)
- Pin 3: Receive Data (RxD)
- Pin 4: Request To Send (RTS)
- Pin 5: Clear To Send (CTS)
- Pin 6: Data Set Ready (DSR)
- Pin 8: Carrier Detect (CD)
- Pin 15: Transmit Clock (TxC)
- Pin 17: Receive Clock (RxC)

During the power-on self-test (POST), the TxD, RxD, TxC, and RxC LEDs glow red indicating a mark (idle) condition. The RTS, CTS, DSR, and CD LEDs glow green indicating a space (off) condition. If an error occurs

in the POST, the LEDs will flash, the audible alarm will sound, and the LCD will display an error code.

The 8-button keypad is arranged as two rows of four keys each. The first row is composed of the Up Arrow, Down Arrow, Enter, and Reset keys. The second row is composed of the Mode, Display, Insert Error, and Hold keys. These keys function as follows:

\*Down Arrow-in the setup phase of the operating mode, this key scrolls the menu selections in reverse



order. In the Review mode, it reverses the order of the captured data to be written to the LCD.

- \*Up Arrow-in the setup phase of an operating mode, this key scrolls the menu selections forward. In the Review mode, it displays the captured data to be written to the LCD in the order it was captured.
- \*Enter-this key is used exclusively for entering the information displayed on the LCD into the operating mode setup configuration.
- \*Reset-during the setup phase of an operating mode, this key brings back the Main Menu. During an actively running test, Reset restarts the test.
- \*Mode-this key is active during the setup and operation of a test and causes an immediate return to the Main Menu when pressed.
- \*Display-only available during Monitor mode, this key toggles the

display from current to hexadecimal character set.

•In Err-this key is not functional.

\*Hold-this key stops the capture of serial data and transfers the user to Review mode for editing of the captured data.

Internally, Analyst 2 is composed of a Zilog Z86C81 (Z8) microcomputer, Zilog 28030 serial communications controller (ZSCC), up to 64 KB of SRAM, 32 KB of program memory, an LM555 dual timer, a Texas Instruments TL7705A reset generator, and various RS-232 drivers and receivers.

Because of its versatility, I chose to use the Zilog Z8 as the processor in Analyst 2. The Z8 has a multiplexed address data bus which can be directly attached to any of the Zilog ZBUS peripherals. It supports up to 64 KB of external program memory, up to 64 KB of external data memory, and has 128 internal registers.

The internal register space is divided into 124 general-purpose registers, 16 CPU and peripheral control registers, and 4 I/O port registers. Each register is 8 bits wide and can be used as accumulators, address pointers, indexes, data, or stack registers.

A register pointer logically divides the register file into 9 groups of 16 working registers which enables the program operating in one register bank to context switch to another register bank in the event of an interrupt condition. It also provides a simple multitasking operating system to be developed which uses a separate register bank for each operating task. When a task switch is done, only the register pointer is updated, and no pushing or popping of the stack contents is needed.

Port 1 on the Z8 microcomputer is configured as the multiplexed address/data bus. This port is wired directly to the Z80C30 SCC and to a 74ALS373 for latching the lower 8 bits of the address bus. Port 0 is configured as

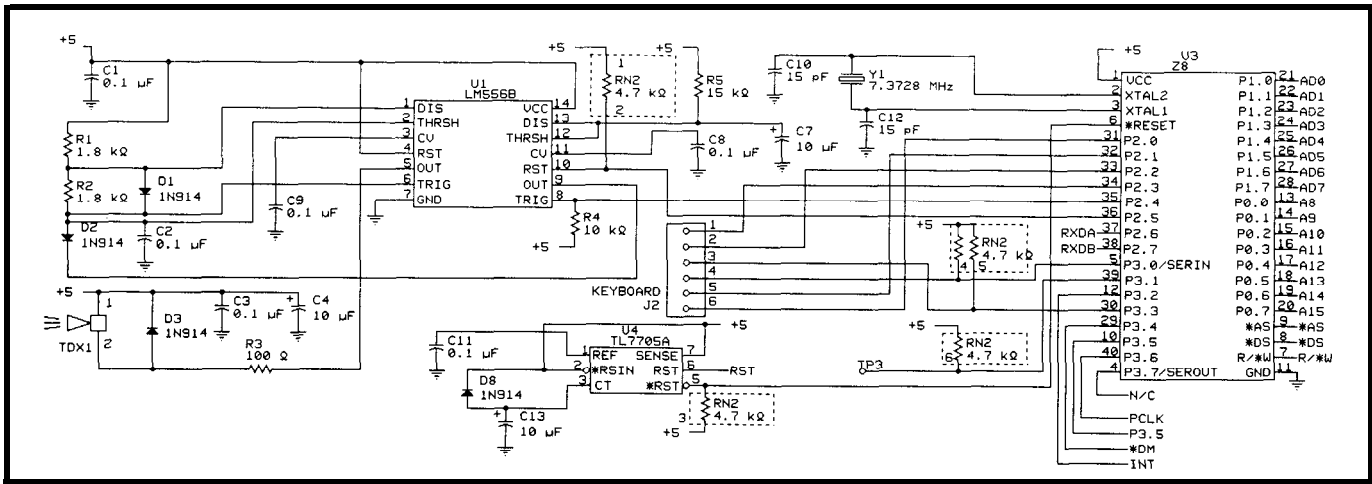


Figure 1 a—At the core of the Analyst 2 is a Zilog Z8 processor. The TL7705A generates a clean reset for the unit while an LM556 dual timer handles the beeper. The keyboard goes right into the processor to be handled by the firmware. Finally, note that the processor generates the UART's 3.6864-MHz clock on its PCLK output.

address lines A8–A15. All pins on this port are individually pulled up to VCC through a 10 kΩ resistor, a necessary step since these pins are undefined during reset. Port 2 is bit definable and thus is used for multiple purposes. Pins 0-3 are used as the active scan lines for the keyboard. Pins 4-5 are used to control the 556 timer for the piezo alarm. Pins 6-7 are used to monitor the serial interface during the distortion tests.

The Z80C30 ZSCC is a dual-channel, multiprotocol data communications peripheral. The device contains on-chip baud rate generators and digital phase-locked loops for each channel. The ZSCC is designed to handle all asynchronous, byte synchronous, and bit synchronous communications protocols. In addition, the device is capable of generating and checking Cyclic Redundancy Check (CRC) codes in all synchronous modes

and provides complete error detection in all asynchronous modes.

The ZSCC is connected to the multiplexed address/data bus from the Z8 microcomputer which enables it to be accessed as a ZBUS peripheral at a much higher rate than the nonmultiplexed version of this part (Z85C30). The Z8 provides the ZSCC with a 3.6864.MHz clock which is derived from the Z8 7.3728.MHz crystal clock. This clock frequency allows the ZSCC

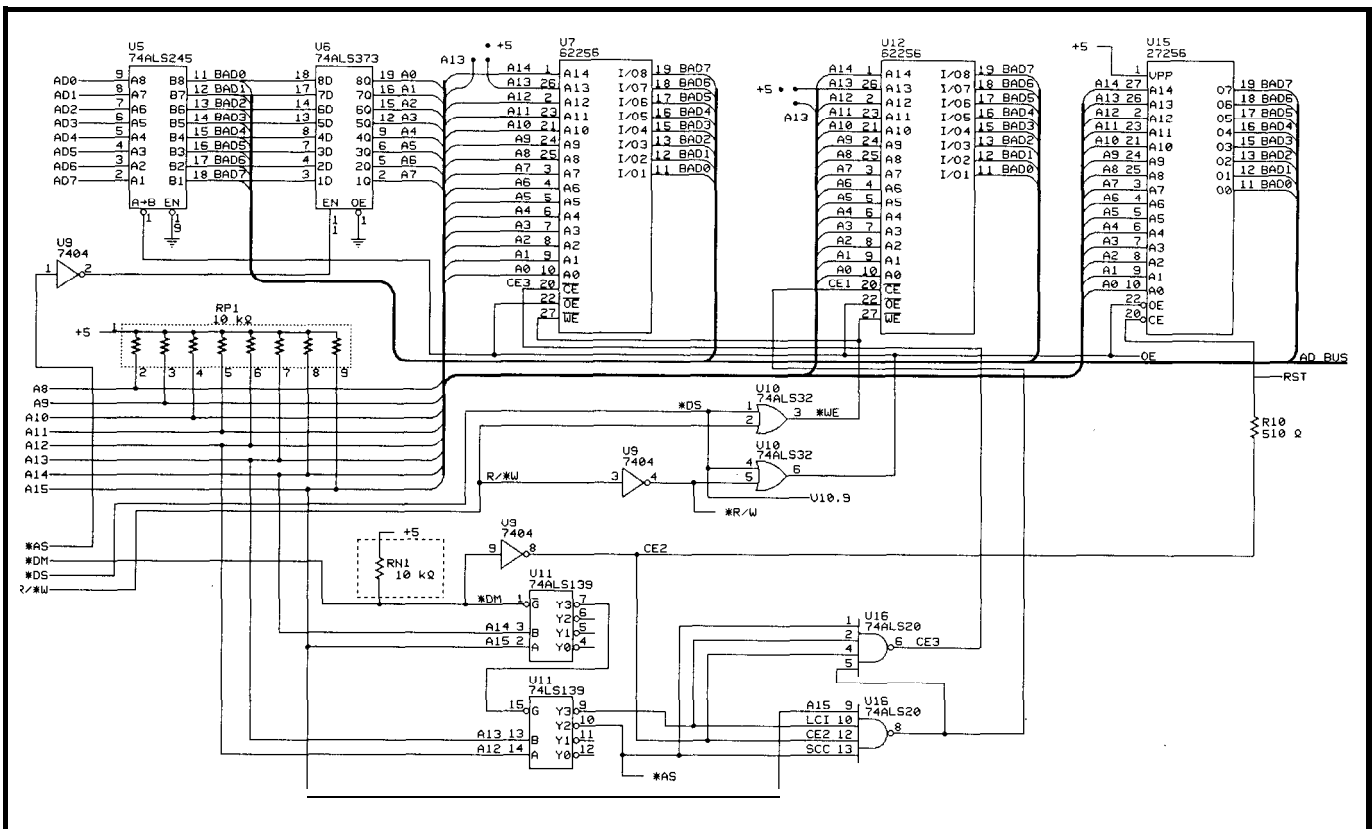


Figure 1b—The Analyst 2 contains 32 KB of EPROM and 64 KB of RAM. All memory and peripheral decoding is done using discrete logic rather than programmable devices.

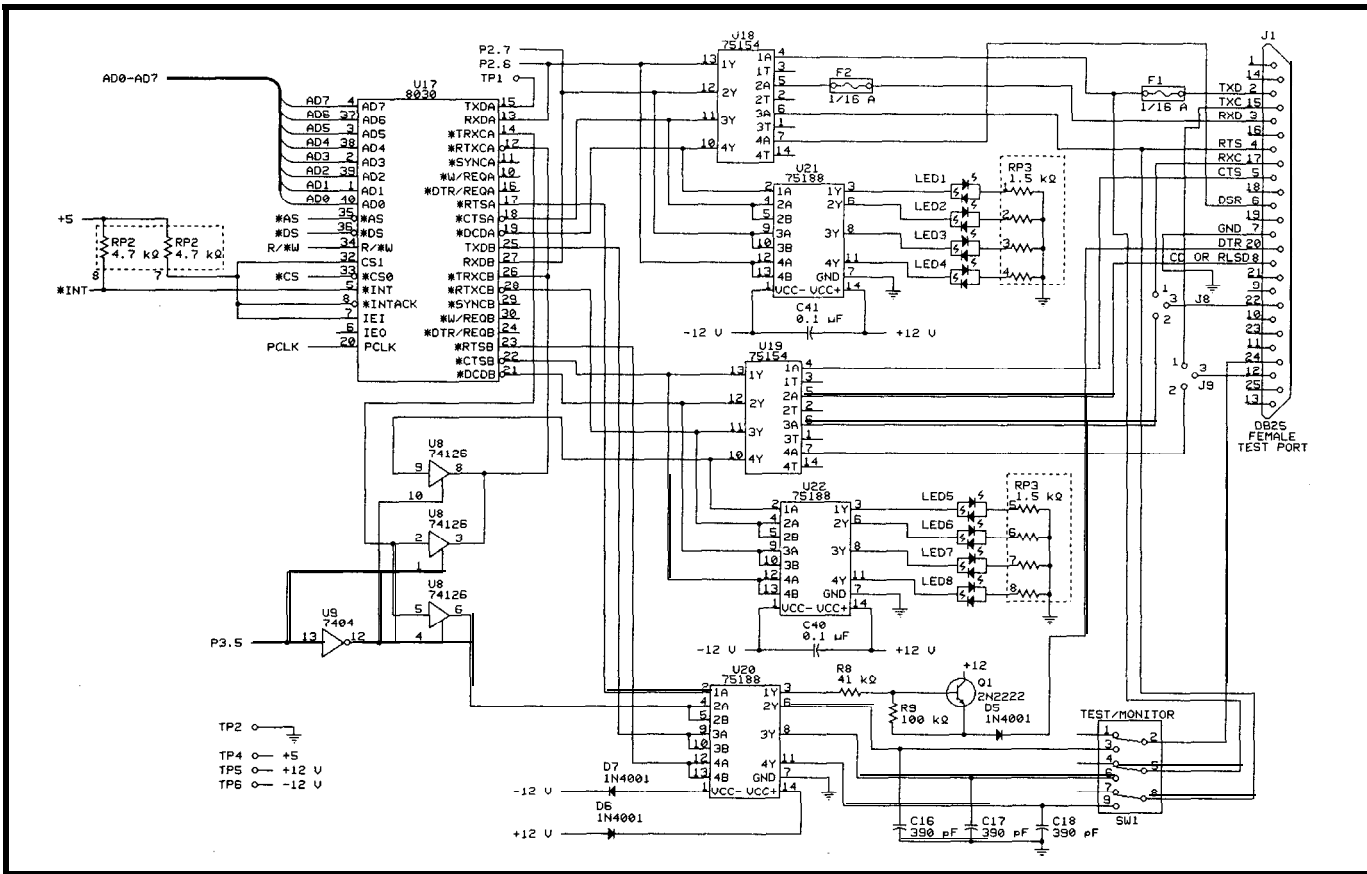


Figure 1 c-The Z80C30 serial communications controller is designed to handle all asynchronous, byte synchronous, and bit synchronous communications protocols

## TURBO-128 THE NEXT GENERATION EMBEDDED CONTROLLER

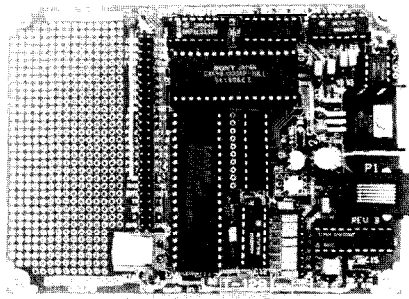
**\*\* NO DEVELOPMENT TOOLS REQUIRED \*\***

READY TO PROGRAM IN BASIC OR ASSEMBLY

Photronics Research Introduces the T-128. A True Single Board BASIC Development System. The T-128 is based on Dallas Semiconductor's new 8051-compatible DS80C320. With its 2X clock speed (25MHz) and 3X cycle efficiency, an instruction can execute in 160ns; an 8051 equivalent speed of 62.5MHz!!! Equally impressive is the T-128's high-speed NVRAM Interface. Any of the 128K RAM may be programmed directly from a PC file through the console, eliminating EPROMs, and associated tools. Program Development has never been faster or more convenient, even with the finest EPROM emulator. The T-128 features PORT 0 bias and EA-select for DS87C520 upgrade.

### PROCESSOR

- Dallas Semiconductor's DS80C320
- 300% more efficient than the 6051
- Three 16-bit Timer/Counters
- 13 Interrupts (6 Ext., 7 Int)
- A second 16-bit Data Pointer
- 364 Bytes of Internal RAM
- Programmable Watchdog
- Brownout Protection
- Power-Fail Reset/Interrupt
- Power-On Reset
- Fully supported by Franklin C



Only 12" x 10" x 1/2" w/ 60-pin DIN connector for power connected by a single 4-conductor telephone wire [very convenient]

Comes Ready to Run  
with power adapter/cable assembly.  
Includes utility diskette with  
**DETAILED TECHNICAL MANUAL.**  
**\$199 in RTV.**

### BASIC-520

- Modified BASIC52 Interpreter [BASIC-520]
- Now Fast Enough for New Applications
- Stack BASIC Programs and Autorun
- CALL ASM Routines for Maximum Speed

### I/O

- Three S-bit Parallel Ports
- Two Full-Duplex RS232 Serial Ports
- Decoded Device I/O Strokes
- SO-Pin Bus Connector

### MEMORY

- Entire 126K Memory Map populated with fast NVRAM (64K DATA + 64K CODE)
- All memory programmed on-board
- Partitionable as CODE/DATA/OVERLAID
- Code Space is Write-Protectable
- State-of-the-Art Data Protection

### UPGRADE [available soon]

- DS87C520 processor (33MHz)
- Instruction cycle: 121 ns
- 8.25 MIPS
- 8051 equivalent: 82.5 MHz
- Internal 16K ROM/I K SRAM
- Runs BASIC 800% faster!

• 109 Camille St. • Amite, LA 70422 • (504) 748-9911 • Tech Support (504) 748-7090 • FAX (504) 748-4242

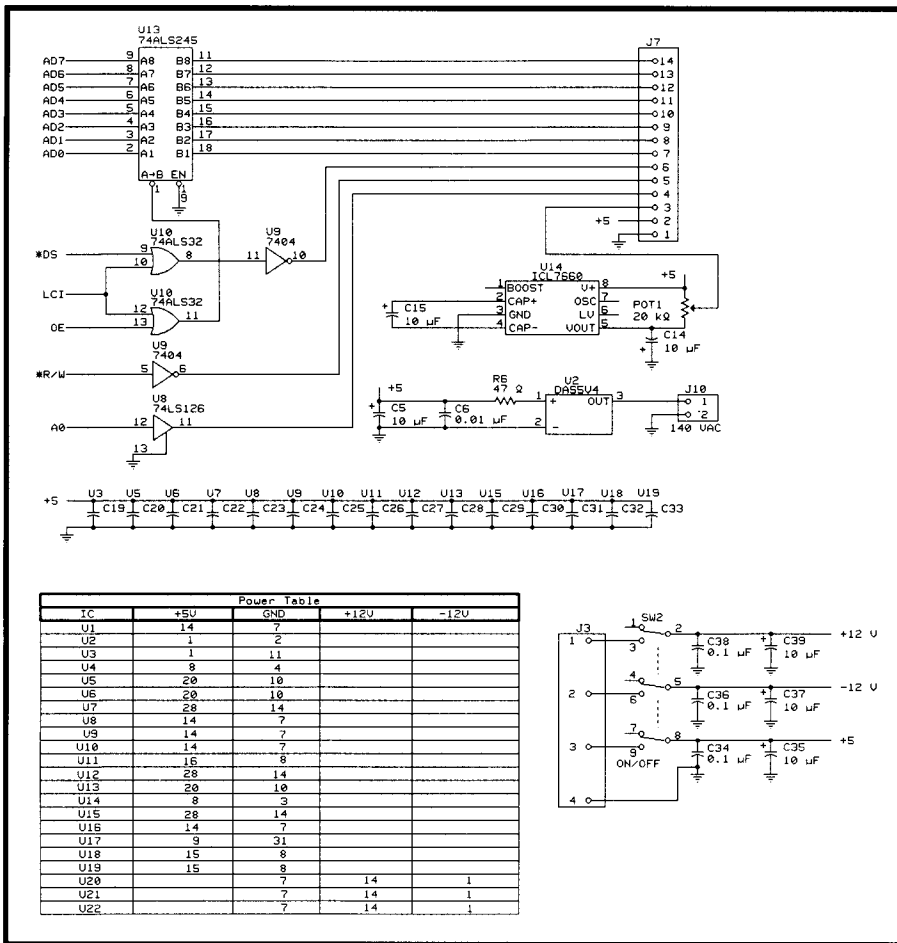


Figure 1d—The LCD interface includes an ICL7660 power inverter to create a negative bias for the display plus a DAS5V4 DC-to-AC converter to drive the EL backlight.

to meet all timing requirements for data speeds ranging from 50 through 38,400 bps.

During a monitoring operation, both receive channels of the Z80C30 are used. Channel A is attached to Pin 2 and Channel B is attached to Pin 3 on the DB-25 interface connector. This configuration enables the unit to monitor both the transmit and receive sides of a serial communications session. When working with a protocol which requires the use of a phase-locked loop for clocking information recovery, Channel A clocking is connected through a 74HC125 to the clock-output pin from Channel B. In this configuration, the baud rate generator for Channel B is used to provide the 32x clock for the phase-locked loop in Channel A.

The NE556 timer is a dual 555 timer in a single DIP package. One 555 timer is used as the frequency generator to

drive the piezo transducer. The other 555 timer is used as a period controller for the frequency generator. With this design, the Z8 microcomputer only needs to toggle a port pin to reset or fire the piezo alarm. The period of the alarm is fixed by the second 555 timer to approximately 0.5 s. This alarm is used as an audible feedback for keypad entries and for signaling.

The TL7705A is a voltage-controlled reset device which will hold the Z8 microcomputer in a reset state whenever the power to the device is out of tolerance. Once the power

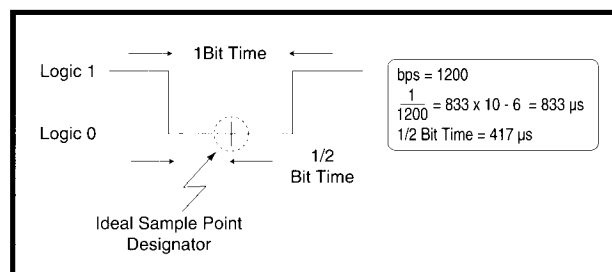


Figure 2—At 1200 bps, a single bit time lasts 833 μs while half a bit time is 417 μs.

supply has stabilized, the TL7705A continues to assert the Reset line to the processor for approximately 250 ms. This allows the Z8 enough time for the oscillator to stabilize before fetching the first instruction from EPROM.

## SOFTWARE DESCRIPTION

Analyst 2's various modes of operation are Data Monitor, Review, Distortion, Time Delay, and Simulate.

### •Monitor

After all setup configurations have been entered into the Analyst 2 for the Monitor mode, the user will be prompted with one final menu selection:

Execute—This selection starts data capture as specified by the user. Serial data displays on the LCD as it starts flowing into the capture memory. At higher baud settings, capture memory fills faster than display speed. When capture memory is full, the alarm sounds three times signifying that it is ready to begin capture again.

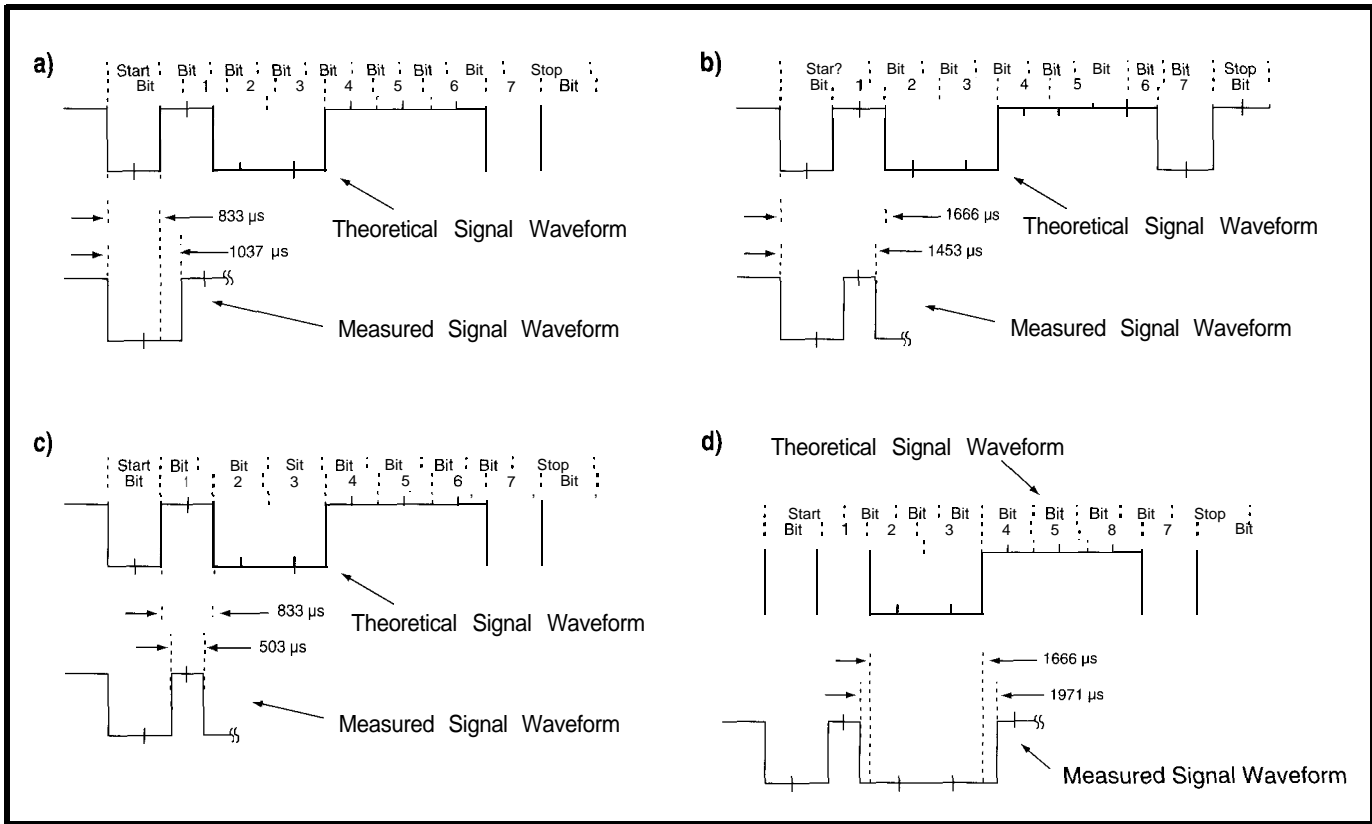
Dsply—During the capture process, the Dsply (display) key toggles between the selected and hexadecimal character set without affecting the data capturing in progress.

Hold—This key immediately stops incrementing the display, and all capturing functions cease. It terminates a capture prior to full capture memory. Rset (reset) invokes the Review mode, allowing the user to scroll forward and backward through the captured serial data stream.

### •Review

The Review mode displays the captured serial data in the manner specified by the menu selections. If an

error condition, such as a parity error or framing error, was detected in the Monitor mode, it displays on the LCD when in the Review mode. The Up and Down Arrow keys are used to control the direction of the displayed data. If either key is held for four character cycles, Analyst 2 goes into an automatic state



**Figure 3** *J-Gross distortion* measures the time between when a transition occurs and when it should occur. Two examples include positive distortion (a) and negative distortion (b). *Isochronous distortion* represents the difference between the measured and theoretical width of a bit time. Two examples include negative distortion (c) and positive distortion (d).

for reviewing the captured serial data. At any time, the Dsply button can change the display to a hexadecimal format. Analyst 2 beeps once when it reaches the end of the nonvolatile capture memory.

The display immediately stops if the Hold key is pressed during the review process. Pressing the Rset key resets the display to the beginning of the capture memory and restarts the operation.

**\*Distortion**

Distortion is defined as a deformity of the data communications signal compared to its theoretical timing parameters. Gross, isochronous, and bias distortion may be present on a data communications line, each relating a specific set of measurements on the signal waveform to the theoretical timing parameters.

Figure 2 defines the terminology. It is assumed that the measurements are taken on a data communications line operating at 1200 bps.

The 1200-bps speed implies a bit time of 833 μs. This time applies to all

bits in the data stream including the start, parity, and stop bits in an asynchronous communications environment. Most communications systems determine the state of the bit by sampling at the middle of the bit time. For my example of 1200 bps, a midsample occurs approximately 417 μs after the falling edge of the bit-time period.

Gross distortion measures the time between when a transition occurs and when it should occur (see Figure 3a). The reference point is an arbitrarily chosen transition from logic 1 to logic 0. The transition time is measured from the reference point to a transition from logic 0 to logic 1. The measured time is compared to the theoretical time and a percentage difference is calculated and displayed on the LCD:

$$\frac{MBT - TBT}{TBT} \times 100 = \% \text{GrossDistortion}$$

where **MBT** represents the Measured Bit Time and **TBT**, the Theoretical Bit Time.

For example, if the transition edge that ends the Start Bit and starts the Bit 1 time period should occur at the theoretical time of 833 μs after the falling edge of the Start Bit period, and the measured time period is actually 1037 μs after the falling edge of the Start Bit, then the gross distortion is 24%:

$$\frac{1037 \mu s - 833 \mu s}{833 \mu s} \times 100 = 24.4\%$$

When the measured transition time of the specified bit begins after the theoretical transition time, but before the half-bit time of the next consecutive bit transition, the specified value is considered positive. The largest positive value measured is the maximum gross distortion result.

As another example in Figure 3b, the transition edge that ends the Bit 1 period and starts the Bit 2 period should occur at the theoretical time of 1666 μs after the falling edge of the Start Bit. The measured time period for the communications line being tested is 1453 μs after the falling edge of the Start Bit, and the gross distortion is:

$$\frac{1453\mu\text{s} - 1666\mu\text{s}}{1666\mu\text{s}} \times 100 = -12.8\%$$

When the measured transition time of the specified bit begins before the theoretical transition time begins, but after the last half-bit time of the preceding bit transition, the specified distortion is considered negative. The largest negative value measured is the minimum distortion value.

The average gross distortion is found by adding the positive distortion values with the absolute value of the negative distortion values, and then dividing by the total number of measurements.

Isochronous distortion represents the difference between the measured and theoretical pulse width of a bit-time. Logic 1s and 0s are treated the same, and the levels between the transitions are ignored. Measurement samples are arbitrarily started at the first logic 0 transition in the communications data stream. The transition time is measured from this reference point until another transition is

detected. The measured time is then compared to the theoretical time, and the percentage difference is calculated and displayed on the LCD using the formula:

$$\frac{MPW - TPW}{TPW} \times 100 = \% \text{Distance}$$

where *MPW* represents Measured Pulse Width and *TPW*, Theoretical Pulse Width. Ten bit times of live data are measured for this test.

In Figure 3c, the first measured pulse width after the logic 0 bit time is 503  $\mu\text{s}$ . The theoretical pulse width is calculated as 833  $\mu\text{s}$  (again assuming 1200 bps). The isochronous distortion is:

$$\frac{503\mu\text{s} - 833\mu\text{s}}{833\mu\text{s}} \times 100 = -39.6\%$$

The negative value calculated shows that the pulse width measured for the selected bit rate is 39.6% less than the theoretical pulse width.

Figure 3d offers another illustration. The next measured pulse width after the logic 1 bit time is 1971  $\mu\text{s}$ . The theoretical pulse width is calcu-

lated as two times the 833- $\mu\text{s}$  bit time or 1666  $\mu\text{s}$ , and the isochronous distortion is:

$$\frac{1971\mu\text{s} - 1666\mu\text{s}}{1666\mu\text{s}} \times 100 = 18.3\%$$

The positive value calculated shows that the pulse width measured for the selected bit rate is 18.3% larger than the theoretical pulse width.

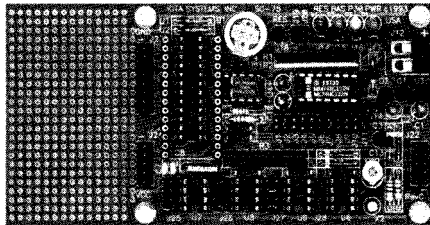
Bias distortion represents the difference in the time duration of logic 1s (marks) and logic 0s (spaces) in the data stream, and is calculated by:

$$\frac{MPW - SPW}{MPW + SPW} \times 100 = \% \text{Bias}$$

where *MPW* represents Mark Pulse Width and *SPW*, the Space Pulse Width. This test is used in asynchronous systems to validate the performance of a transmitting device. Although the alternating mark-space pattern is the preferred pattern to use on a test system, it also works with normal data.

In Figure 4a, the measured pulse width for the logic 0 bit time is 1375

## Speed Your Development Process By Using Our Controllers!



We offer an array of controller boards and software tools for the 8051 and 87C751 families of microcontrollers. Complete packages are available to help you develop your projects. We also have a selection of add-on peripherals such as LCD and keypad interfaces.

### Features:

- Breadboard area
- Flexible I/O arrangement
- Powerful controller BASIC for the 87C752
- Simulators

**Ph: (702) 831-6302 • Fax: (702) 831-4629**

**Iota Systems, Inc.**

POB 8987 • Incline Village, NV 89452-8987

The  
only  
8051/52  
BASIC  
compiler  
that is  
100 %  
BASIC 52  
Compatible  
and  
has full  
floating  
point,  
integer,  
byte & bit  
variables.

- Memory mapped variables
- In-line assembly language option
- Compile time switch to select 8051/803 1 or 8052/8032 CPUs
- Compatible with any RAM or ROM memory mapping
- Runs up to 50 times faster than the MCS BASIC-52 interpreter.
- Includes Binary Technology's SXA5 1 cross-assembler & hex file manip.util.
- Extensive documentation
- Tutorial included
- Runs on IBM-PC/XT or compatible
- Compatible with all 8051 variants
- **BXC51 \$ 295.**

**508-369-9556**

**FAX 508-369-9549**



**Binary Technology, Inc.**

P.O. Box 541 • Carlisle, MA 01741





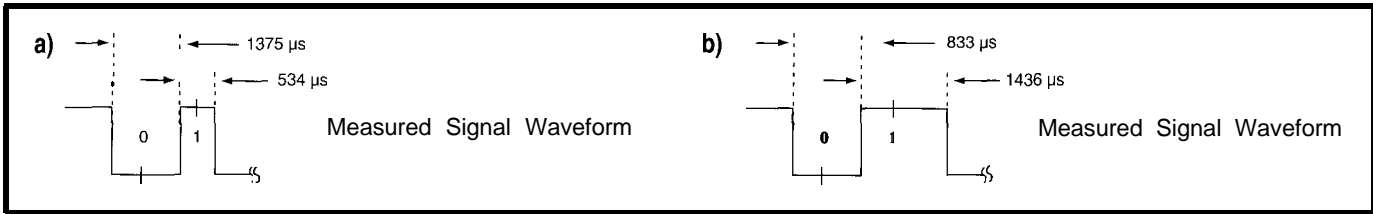


Figure 4—Bias distortion represents the difference in time duration of logic 1s and logic 0s in the data stream. Examples include negative distortion (a) and positive distortion (b).

μs. The measured pulse width for the logic 1 bit time is 534 μs. Since bias distortion is concerned only with the ratio of a mark to a space, no theoretical bit times are necessary (Analyst 2 still requires the user to enter the speed of the communications line, but only to properly setup the interface). In this example, the bias distortion is:

$$\frac{534 \mu\text{s} - 1375 \mu\text{s}}{534 \mu\text{s} + 1375 \mu\text{s}} \times 100 = -44.1\%$$

The negative bias distortion shows that the ratio of the pulse widths measured for the selected bit rate are 44.1% larger for logic 0 than the pulse width for the logic 1.

In Figure 4b, the measured pulse width for logic 0 bit time is 833 μs, and the measured width for logic 1 bit time is 1436 μs. Its bias distortion is:

$$\frac{1436 \mu\text{s} - 833 \mu\text{s}}{1436 \mu\text{s} + 833 \mu\text{s}} \times 100 = 26.6\%$$

The positive value calculated shows that the ratio of the pulse widths measured for the selected bit rate are 26.6% smaller for the logic 0 pulse width than for the logic 1 pulse width.

#### \*Time Delay

The Time Delay test is used to measure the internal delay on various control signal leads within the RS-232 interface. Analyst 2 actively stimulates a control signal and waits for a response. This sequence is repeated for 10 samples before the test completes. This function is useful for testing a modem's internal delays between signals such as RTS and CTS. Analyst 2 can measure time intervals including

(see Figure 5a). This process is repeated for ten sample periods.

If the selection is RTS off to CTS off, the unit deasserts the RTS signal and waits for the CTS signal to go to the inactive state [see Figure 5b).

If the selection is RTS off to CD on, the unit deasserts the RTS signal and waits for the CD signal to go to the active state (see Figure 5c).

#### \*Simulate

The Simulate mode is used to generate known data traffic on a communications line, which is useful in validating the integrity of printers, modems, or even terminal emulation software. Analyst 2 outputs the Quick Brown Fox message for the test.

### DATA COMMUNICATIONS FORMATS

The framing selections supported by Analyst 2 are asynchronous, synchronous with one synchronization character, synchronous with two synchronization characters, and synchronous data link control (SDLC/HDLC).

Asynchronous communication uses what's known as *character-framed data* in which each transmitted character has a start bit, 7 or 8 data bits, and 1 or more stop bits (see Figure

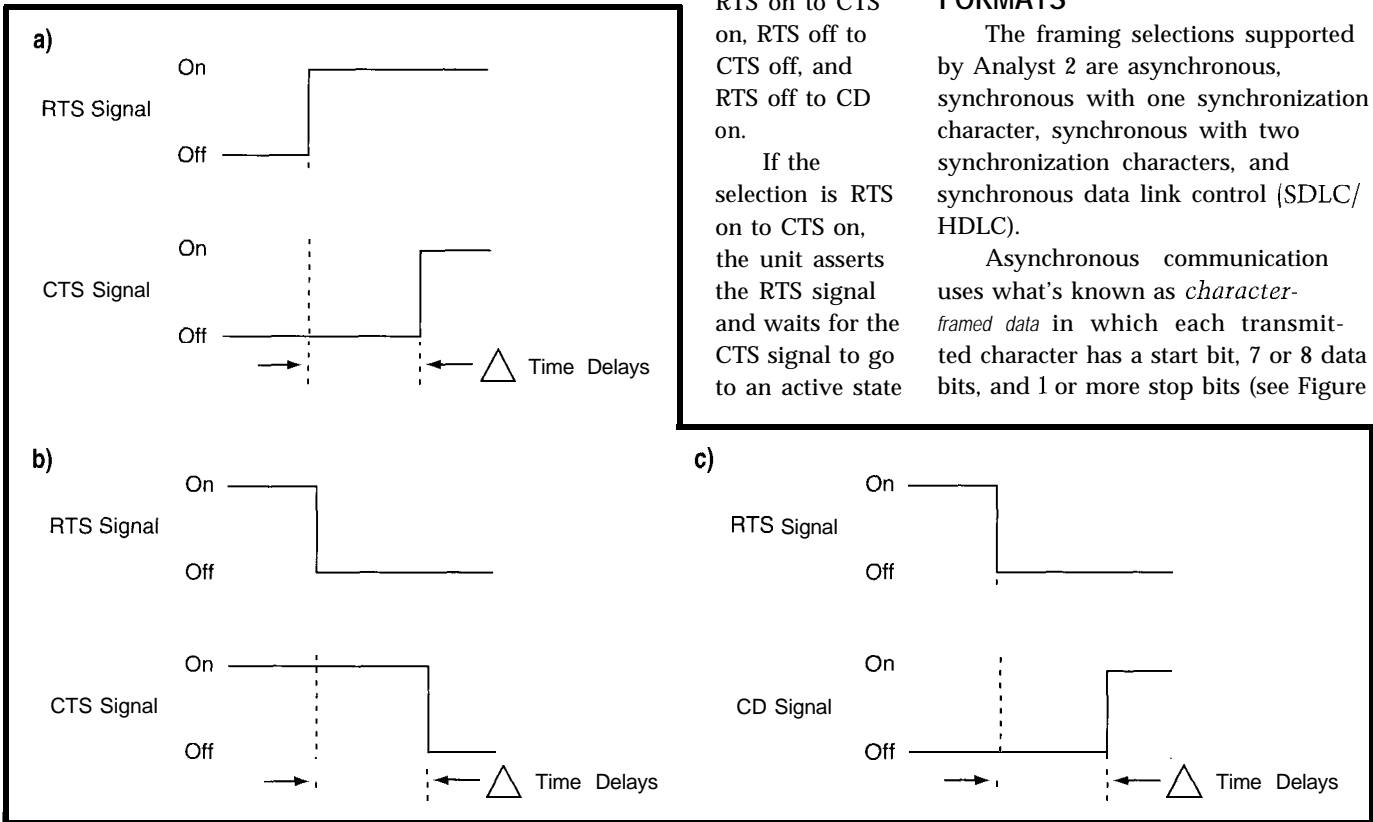


Figure 5—The Analyst 2's time delay test is used to measure the internal delay on various control signal leads within the RS-232 interface. Options include a) RTS on to CTS on, b) RTS off to CTS off, and c) RTS off to CD on (c).

# PIC16C5x Real-time Emulator

Introducing RICE16-5x and RICE5x-Junior, real-time in-circuit emulators for the PIC16C5x family microcontrollers: affordable, feature-filled development systems from

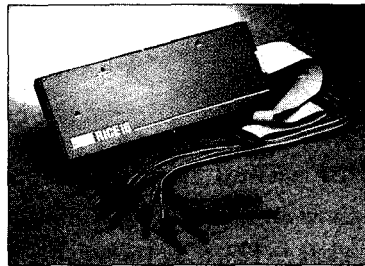
**\$599\***

\* Suggested Retail for U.S. only

## RICE16 Features:

Real-time Emulation to 20MHz

- PC-Hosted via Parallel Port
- Support all oscillator types
- 8K Program Memory
- 8K by 24-bit real-time Trace Buffer
- Source Level Debugging
- Unlimited Breakpoints
- External Trigger Break with either "AND/OR" with Breakpoints
- Trigger Outputs on any Address Range
- 12 External Logic Probes
- User-Selectable Internal Clock from 40 frequencies or External Clock
- Single Step, Multiple Step, To Cursor, Step over Call, Return to Caller, etc.
- On-line Assembler for patching instruction



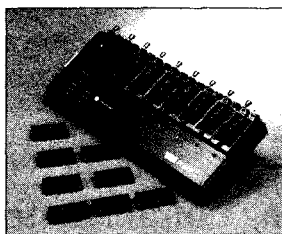
- Support 16C71, 16C84 and 16C64 with Optional Probe Card
- Easy-to-use Windowed Software
- Comes Complete with TASM16 Macro Assembler, Emulation Software, Power Adapter, Parallel Adapter Cable and User's Guide
- 30-day Money Back Guarantee
- Made in the U.S.A.

RICE5x-Junior supports PIC16C5x family emulation up to 20 MHz. It offers the same real-time features of RICE16 without the real-time trace capture.

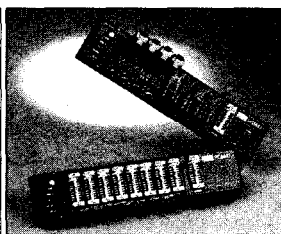
## PIC Gang Programmers

Advanced Transdata Corp. also offers PRODUCTION QUALITY gang programmers for the different PIC microcontrollers.

- Stand-alone COPY mode from a master device
- PC-hosted mode for single unit programming
- High throughput
- Checksum verification on master device
- Code protection
- Verify at 4.5V and 5.5V
- Each program cycle includes blank check, program and verify eight devices
- Prices start at **\$599**



PGM16G: for 16C5x family  
Throughput: 16C54: 4 seconds  
(8 devices) 16C57: 10 seconds



PGM47: for 16C71/84  
16C71: 10 seconds  
16C84: 15 seconds



PGM17G: for 17C42  
17C42: 13 seconds

Call **(214) 980-2960** today for our new catalog.

For RICE16.ZIP and other product demos, call our BBS at (214) 980-0067.



Advanced Transdata Corporation Tel **(214) 980-2960**  
14330 Midway Road, Suite 120, Dallas, Texas 75244 Fax (214) 980-2937

6a). In this framing format, the data communications line is normally in an active (marking) state. No clocking signal is provided on the interface. Instead, the receiver uses a 16 times clock which starts whenever an edge transition occurs (start bit).

Synchronous communications with one synchronization character is a framing format that uses message-framed data and is transmitted as a series of characters with no start or stop bits (see Figure 6b). Instead, this format uses a uniquely defined character to mark the start of a message. Synchronization is maintained through the use of clocking signals on the interface and detection of the unique synchronization patterns in the data stream.

Synchronous communications with two synchronization characters is a framing format developed by IBM. It is referred to as *bisync communications* and is a subgroup of the message-framed data format. It uses a unique sequence of two contiguous characters to define the start of a message packet, and synchronization is maintained through the use of clocking signals on the interface and detection of the two synchronization characters (see Figure 6c).

Synchronous Data Link Control (SDLC) is another framing format developed by IBM. It is widely used, is the basis for the IBM/SNA environment, and was a model for HDLC, the international version. This format differs from the other protocols in that it is bit-oriented rather than character-oriented. Its special bit pattern, referred to as a flag, uses a binary 01111111 pattern for proper synchronization (see Figure 6d). The transmitter circuitry modifies any data which has this specific pattern by a technique referred to as zero insertion. With this technique, a flag character can only be sent on demand and will not occur in the transmitted data stream.

## SOFTWARE SETUPS

Analyst 2 supports both asynchronous and synchronous clocking modes. The mode affects how the unit determines where the middle of a bit occurs and the duration of a bit. The

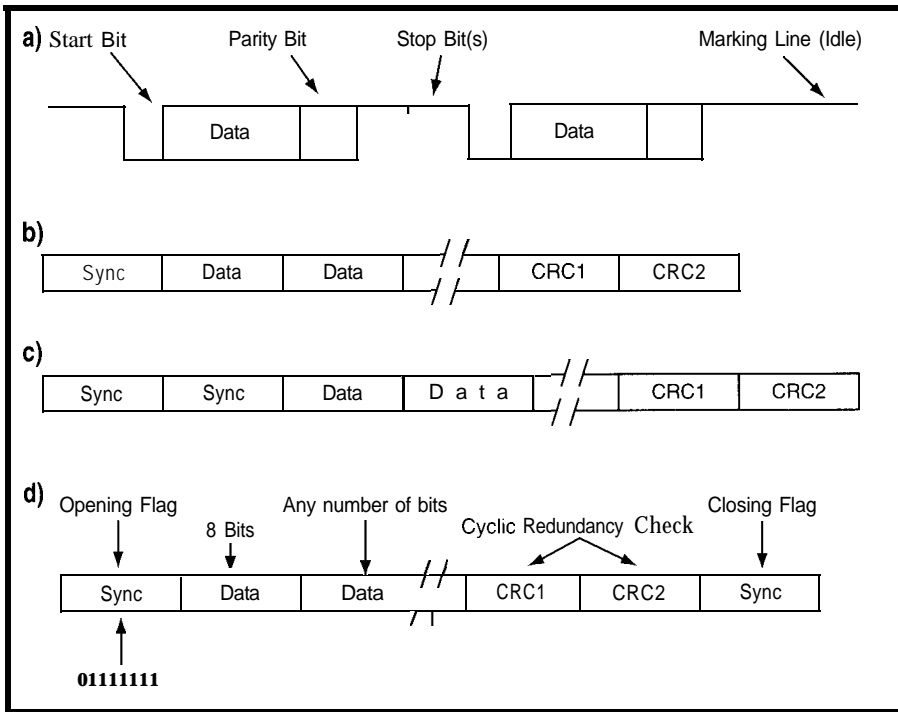


Figure 6—Various serial communications methods include a) asynchronous, b) synchronous using one sync character, c) bisynchronous, and d) SDLC/HDLC/X.25 bit-oriented.

unit can either use an internally generated clock or accept clocking from an external source. Additionally, it can recover the clock from an encoded data stream, such as nonreturn to zero Invert (NRZI), for reception and transmission if desired.

The available selections for clocking Analyst 2 are Internal, NRZI, and From DCE.

\*Internal—this selection implies that no clocking signals will be present on the DB-25 interface. Instead, all clocking will be generated internally by the device and is used exclusively by asynchronous communications systems such as those on the COM1 and COM2 ports of a PC. After selecting Internal, the baud rate for the communications line has to be set to any of the standard values from 50 to 38,400 bps.

● NRZI—this selection implies that clocking is embedded within the data stream itself and is recovered using a digital phase-locked loop. This clocking information is then used to process the received bitstream into proper bit-timing periods. Most communications systems use non-return-to-zero (NRZ) encoding for bit transmission which preserves a one as

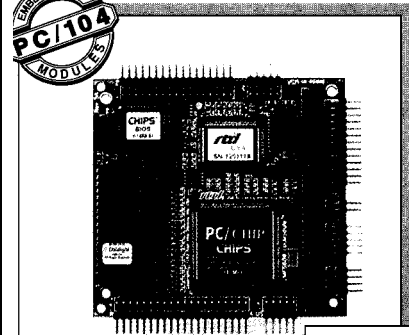
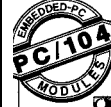
a logic 1 and zero as a logic 0. In NRZI encoding, logic 1 is represented by no change in the signal polarity, and logic 0 is used to alter the polarity of the signal (see Figure 7). For Analyst 2 to process NRZI encoding properly, the communications speed for the line being tested must be entered. This value is used to set up the baud rate generators for a xl clock for all transmitted data and a x32 clock for the input to the digital phase-locked loop for the receiver.

\*From DCE—this selection implies that the communications line to which Analyst 2 is attached is providing clocking signals on the DB-25 interface leads (see Figure 8). These clocking signals are used both to receive and transmit data. In most cases, the transmit clocking is on interface pin 15, and the receive clocking is on interface pin 17. If this is not the case, Analyst 2 can be configured to derive clocking from either of the pins for both transmit and receive operation.

The setup for the data format must be entered after the clocking information has been entered. The menus are for Bits per Character, Parity (asynchronous only), Stop Bits

## Replace Four Conventional PC/104 Modules with One CMF8680 cpuModule™

Embedded PC/XT Controller with Intelligent Power Management



**\$595**

PC/104 Compliant  
Actual Size: 3.6 x 3.8 x 0.6"

- PC/XT compatibility with 286 emulation
- 14 MHz, 16-bit C&T FE680 CPU
- +5V only; 1.6W at 14.3 MHz, 1W at 7.2 MHz
- Intelligent sleep modes, 0.1W in Suspend
- ROM-DOS and RTD enhanced BIOS
- Compatible with MS-DOS & real-time operating systems
- 1 M bootable solid state disk & free software
- 4K-bit configuration EEPROM (2K for user)
- 2M on-board DRAM
- IDE & floppy interfaces
- CGA CRT/LCD controller
- Two RS-232 ports, one RS-485 port
- Parallel, XT keyboard & speaker ports
- Optional X-Y keypad scanning/PCMCIA interface
- Watchdog timer & real-time clock

Expand This Or Any PC/104 System with the

### CM106 Super VGA Controller utilityModule™

- Mono/color STN & TFT flat panel support
- Simultaneous CRT & LCD operation
- Resolution to 1024 x 768 pixels
- Displays up to 256 colors

**\$295**

### Speed Product Development with the SK-CM1 06-X Starter Kit

Your kit includes the CMF8680 cpuModule, CM 106 SVGA controller, CM 102 keypad scanning/PCMCIA utilityModule, CMF8680 cable kit & VGA monitor cable for just \$1095

Additional PC/104 compliant modules from RTD:

- CM104 1.8" hard drive carrier utilityModule
- 12- & 14-bit analog I/O modules
- 12-bit, 4-20 mA analog output modules
- opto-22 & digital I/O modules

For more information on our PC/104 and ISA bus products, call today.



### Real Time Devices USA

200 Innovation Blvd. • P.O. Box 906  
State College, PA 16804 USA  
(814) 234-8087 / Fax: (814) 234-5218

### RTD Europa • RTD Scandinavia

Real Time Devices is a founder of the PC/104 Consortium.

(asynchronous only), Bit Order, and Data Inversion.

•Bits per Character-this sets the number of bits used by the ZSCC to determine character boundaries. You can choose 8, 7, 6, or 5 bits.

\*Parity-provides a means of error detection in asynchronous communications. It can be set to none, odd, or even.

\*Stop Bits-this sets the number of stop bits used by the ZSCC to determine character boundaries in asynchronous systems. The number of Stop Bits can be set to 1, 1.5, or 2 bits.

\*Bit Order-this sets the order in which the captured bits are received. Most transmission systems use a 1-8 (least significant bit first) transmission scheme. Analyst 2 can be set to either 1-8 or 8-1.

\*Data Invert-this selection offers the option of decoding protected data by inverting each bit as it is captured. The setting is a toggle in which "yes" inverts the data and "no" captures it as it is. Few civilian communication systems invert data which is being transmitted.

If a synchronous data format has been selected, Analyst 2 prompts the user for the Sync Character, Drop Sync Character, and characters to be suppressed [if any].

•Sync Character-a specific bit pattern used by the ZSCC to establish character boundaries for the incoming data stream. The available selections are Sy\_Sy, Dle\_Sy, Flag, or User.

If the data format selected was BiSync, the default will be the Sy\_Sy bit pattern, which loads the sync pattern detector of the ZSCC with a 16-bit value. If the code selection is ASCII, the value becomes a hexadecimal 16 16, and if EBCDIC, the value is the hexadecimal 3232. Some Bisync systems use a Data Link Escape (DLE) character before the sync character for system-bit synchronization.

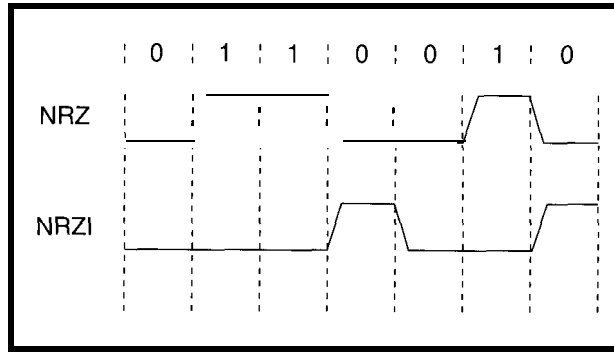


Figure 7-h NRZ encoding, a 1 is represented by a high level and a 0 is represented by a low level. In NRZI encoding, a 1 is represented by no change in level and a 0 is represented by a change in level.

To capture data properly in this type of system, the user needs to select the Dle\_Sy menu item. If the code selection is ASCII, the 16-bit sync pattern detector of the ZSCC is loaded with a hexadecimal 1016, and if EBCDIC, the value is a hexadecimal 1032.

With SDLC/HDLIC format, the default flag marks the beginning or end of the transmission frame. The sync pattern detector will be loaded with a hexadecimal 7E.

With the 1\_Sync data format, Analyst 2 requires the entry of a two-digit hexadecimal number to be used as the sync character. This enables Analyst 2 to be used in proprietary communication systems which use nonstandard synchronization bit patterns.

\*Drop Sync-set the bit pattern for the ZSCC so it can look for the next sync character in the data stream. If the selection is 1, the ZSCC will begin searching for a new sync character as soon as the line goes to a marking state. This state is commonly referred to as the *idle line* condition on a communications line.

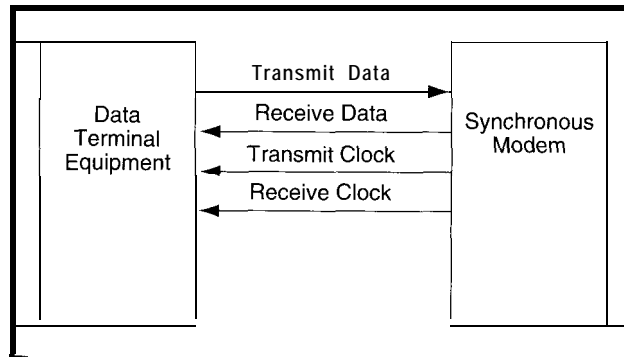


Figure 8-In a synchronous setup, the modem provides the clocking information to the DTE (the clocks are both xl).

\*User-this selection enables the user to enter a two-character hexadecimal value. When this value is detected in the incoming data stream, the ZSCC enters hunt mode and begins looking for the next sync character.

In some synchronous environments, a specific character is used to signify a marking line instead of a signal level. In this case, Analyst 2 needs to be configured to suppress that character

from being transferred into the capture memory. The available selections are mark, space, none, sync character, and user defined.

The display update speed may seem relatively slow compared to the data speed. However, data capture can take place at 38,400 bps, a rate exceeding LCD readability. As the LCD updates, the capture RAM continues to fill according to the selected mode of operation. The available modes are Continuous, Buffer, Error Stop (asynchronous only), Signal, or User.

\*Continuous-captures data until the capture RAM is filled. When this occurs, an alarm sounds once and Analyst 2 begins ignoring the data on the line. The LCD starts incrementing at a faster rate since it now has the priority. When the LCD gets to the end of the capture RAM, the alarm beeps three times and the unit begins capturing data from the communications line after a 0.5-s pause. The unit exits from the monitor mode if any key is pressed during the pause.

\*Buffer-captures data until the capture RAM is filled. When this occurs, the unit beeps three times and begins ignoring data on the line. When the LCD gets to the end of the capture RAM, the unit waits for the Rset key to be pressed before beginning another capture.

\*Error Stop (Er Stop)-prompts the user for more information about the type of error: Parity (P), Framing (F),

Parity and Framing (P,F), Break (B), Parity and Break detect (P,B), Framing and Break (F,B), and Parity, Framing and Break detect (P,F,B). The data line is continually monitored for an error condition occurrence after a selection has been made. Data capture stops on detection of the selected error.

\*Signal-enables Analyst 2 to be configured to monitor a specific signal lead on the RS-232 interface for a transition. The unit is therefore able to function as a glitch catcher. The available selections are RTS (pin 4), CTS (pin 5), DSR (pin 6), and CD (pin 8). After selecting a signal to watch, the user must enable the trap function. Care should be exercised in enabling the trap function since this setting is stored in the nonvolatile configuration RAM and remains in effect at all times. The polarity of the signal to be watched must be entered after enabling the trap function. Either edge can be selected as the trigger.

\*User-enables Analyst 2 to be programmed to either start or stop capture at the occurrence of a specific

user-entered pattern. The unit prompts the user for up to 8 hexadecimal characters. "Don't cares" can be entered as "\*".

## CONCLUSION

Analyst 2 is a very powerful device for debugging a serial communications problem, but it also becomes a very handy tool around the shop for all sorts of things. For examples, RS-485 twisted-pair systems can be debugged by constructing a simple RS-232-to-RS-485 converter. Serial links between processors using a three-wire interface (such as that supported by the 805 1 and other microcontrollers) can be debugged with a simple TTL-to-RS-232 converter.

Good luck with the kit and happy bug hunting! 🐛

*Bill Payne holds a B.S. in Computing Sciences from the University of Oklahoma, College of Electrical Engineering. He has 12 years of experience in the design of computer-based equipment. He holds two*

*semiconductor patents and has four others pending. He is also a Novell Certified Netware Engineer (CNE). He can be reached at (214) 487-7074.*

## SOURCE

A complete kit including all components and a four-layer circuit board is available from Payne Research. They can be reached at (214) 487-7074.

## SOFTWARE

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

## I R S

404 Very Useful  
405 Moderately Useful  
406 Not Useful

# Put The Computer Applications Journal to work for you



**The Computer Applications Journal** readers are engineers, programmers, consultants, and serious computer technologists. Bring them into your store by selling **The Computer Applications Journal**.

**The Computer Applications Journal's** Direct Dealer Sales Program lets you increase your store traffic—increase your sales—increase your profits with no risk.

Let our advertising efforts turn into your profits.

For information on how your business can become part of the growing **Computer Applications Journal** success story, write or call:

**The Computer Applications Journal**  
attn: Dealer Sales

4 Park Street • Vernon, CT 06066

(203) 875-2199 • Fax (203) 872-2204

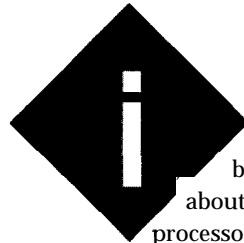
# A RISC Designer's New Right ARM

## Designing with the ARM Processor

Before you discount something as powerful as a 32-bit processor in your next embedded design, take the time to check out the ARM series of RISC processors. They make more sense than you might think.

## FEATURE ARTICLE

Art Sobel



I must admit I've been enthusiastic about the ARM processor for a long time. I hope I will be able to pass on some of my enthusiasm to you.

I first became aware of the ARM through Dick Pountain's article about the Acorn RISC computer (then made only in England) in *BYTE* (1987). Pountain has since written more about the ARM in *BYTE* and Tom Cantrell has mentioned the ARM processor in *CAJ*; however, there haven't been any articles in any magazine in the U.S. devoted to the ARM and addressed to the working engineer, experimenter, or homebrew enthusiast, even though there have been other articles devoted to other 32-bit processors, such as the i80960 and the AMD 29200. I hope to fill this gap with this article.

The subject matter, however, is so broad that I will be presenting the material in three parts. The first article provides a background to the ARM, an overview of existing systems which use the ARM, and some aspects of ARM architecture. The second article, scheduled for December, focuses on existing ARM development systems and hardware design considerations, and the third, coming in the new year, discusses software development with ARM.

## JUST A BIT OF HISTORY

I studied the architectures of the available processors for several years in hopes of making some sort of homebrew computer. Although I had been interested in the National 16C32 and the 68000, the simplicity of the ARM and its performance impressed me. As a processor designer myself, I appreciated the elegance, efficiency, and symmetry of the design. As well, it offered an auxiliary chip set-one that could be used to design and make a high (for that time) performance computer.

The ARM design promised a 3-4 MIPS computer with 2-3 times the performance of the then-current '286 computers. This chip set consisted of an ARM2, an MEMC 1 (including a DRAM controller, inverted page table MMU, and sound and video DMA channels), a VIDC (with 4-8-bit-per-pixel video and sound output at up to VGA resolution), and an IOC (with serial keyboard interface, interrupt controller, and peripheral decodes). I resolved to get this computer.

Acorn, unfortunately, would not sell any of its computers to the U.S. market. So, I had a small computer store in Manchester ship me an Acorn Archimedes A310 [see Figure 1]. After

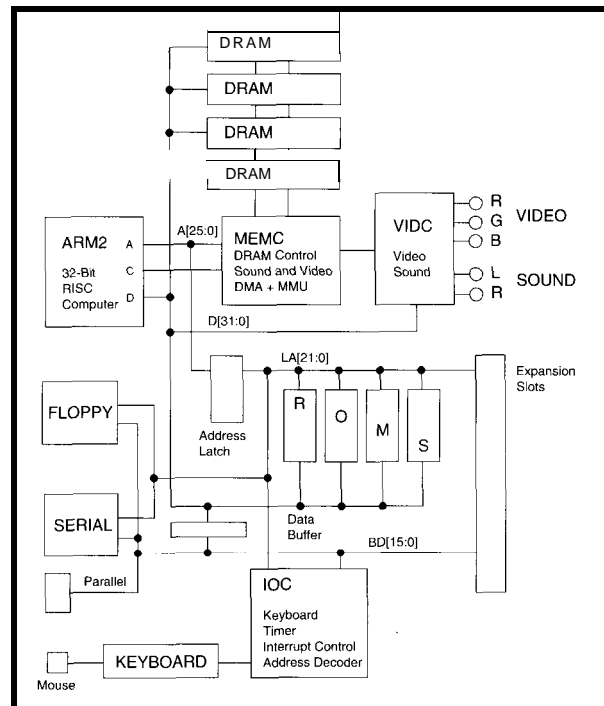


Figure 1-The Acorn Archimedes A310 system was one of the first desktop machines based on the ARM processor.

about 4 months (during which time the first A3 10 shipped to me was stolen at Heathrow airport), I received a computer. In addition to the ARM and support chips, the Acorn Archimedes A310 had 1 MB of RAM, a floppy drive, keyboard, video output, and serial and parallel ports.

With the addition of a multisync monitor, I was in business. I verified the advertised performance and wrote a BASIC version of the Mandelbrot set. To my delight, the A310 was 10 times faster than the same BASIC program running on my 8-MHz AT. This performance improvement was due not only to sheer processor power, but also to a superior BASIC interpreter and simple but efficient video display architecture. After all, the best VGA chip is no VGA. The complicated access modes and unpacked pixels of a VGA display chip get in the way of CPU access to display RAM.

At this time, the ARM processor was tightly connected to Acorn and, although Acorn thought it would be nice to have wider use for the chip, it had no way of actually promoting the architecture.

A division of VLSI technology in Tempe, Arizona attempted to develop an ARM test board. Unfortunately, they couldn't complete the project because of a lack of adequate development software. The development software only operated on the Acorn computer or on an ARM-based coprocessor board for the PC. This board was only manufactured by Acorn and had an operating system that was fundamentally incompatible with PC files. This was going to be harder than I thought!

Despite ARM's software problems, Apple was persuaded to try an ARM project to replace the obsolete Apple II. The result was very impressive. Not only did the ARM-based computer prototype emulate the 6502 and 65C816 processors, it even ran Macintosh software faster than the 68000.

The project was nipped in the bud—Apple did not want ARM to threaten existing computer groups, especially since the ARM processor was owned by a direct competitor

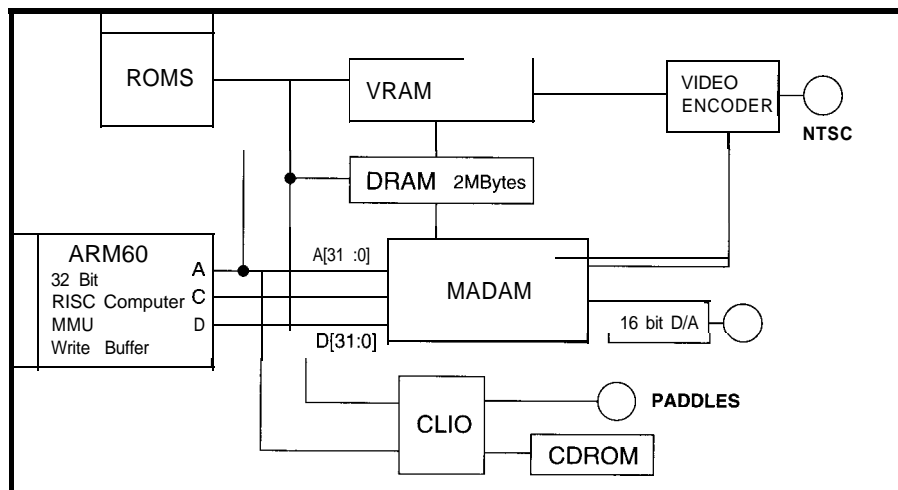


Figure P—The 3DO Interactive Multiplayer includes the ARM60 running at 12.5 MHz. Much of the drawing and I/O control is provided by two complex coprocessors. CLIO handles the drawing and warping commands and MADAM reflects I/O, memory control, and several DMA channels to the drawing coprocessor, sound, video, and CD-ROM.

(Acorn has a large percentage of the U.K. education market).

To promote the ARM processor and make it possible for Apple to buy the CPU, the advanced development group at Acorn that had developed ARM was regrouped to form a new company called ARM Ltd. (Advanced RISC Machines), owned by Acorn, Apple Computer, and VLSI Technology. The VLSI ARM application group, originally based in Arizona, was restarted to make the ARM processor in San Jose, California. This move made it possible for me to join their

hardware applications team. (After 20 years in one spot, I am firmly planted in Silicon Valley—even a job using the ARM had to come to me!)

### WHY A 32-BIT PROCESSOR?

To many, a 32-bit processor seems like a bit of overkill. However, if your application uses over 64 KB of code or handles more than 64 KB of data, a fast 32-bit processor may be just what you need (the ARM6 is smaller and less complicated than an 8086 in any case). Bitmapped graphics displays, image generation or analysis, large databases,

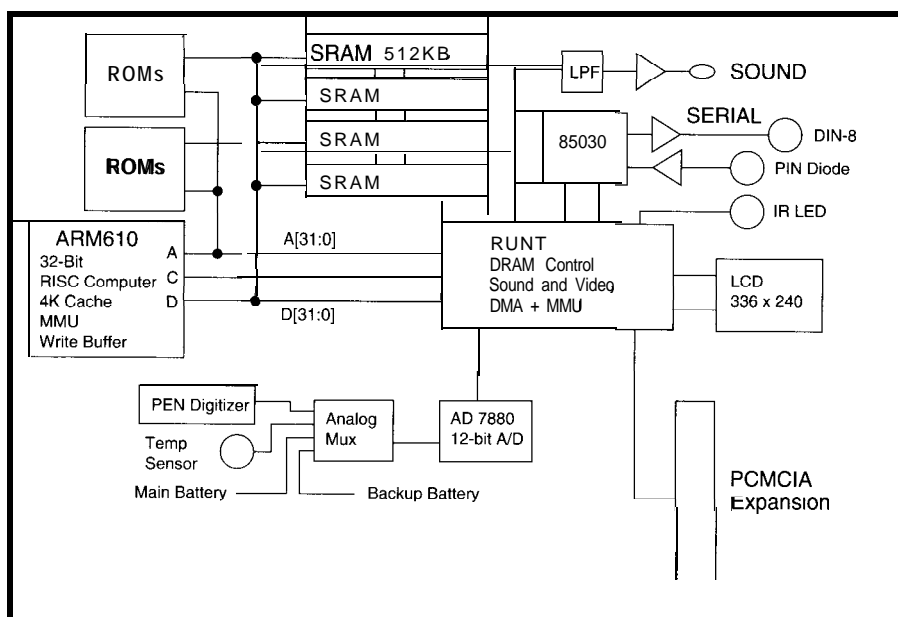


Figure 3—The Apple Newton uses the ARM610 running at a maximum speed of 20 MHz. The whole system is designed for low power dissipation using CMOS-SRAM and slowing the clock speed of the ARM when idle. Although ARM610's performance is roughly equivalent to the Motorola 68LC40, it dissipates much less power. Following versions of the Newton will be even more streamlined. What is represented in this entire block diagram will be absorbed into at most three chips.

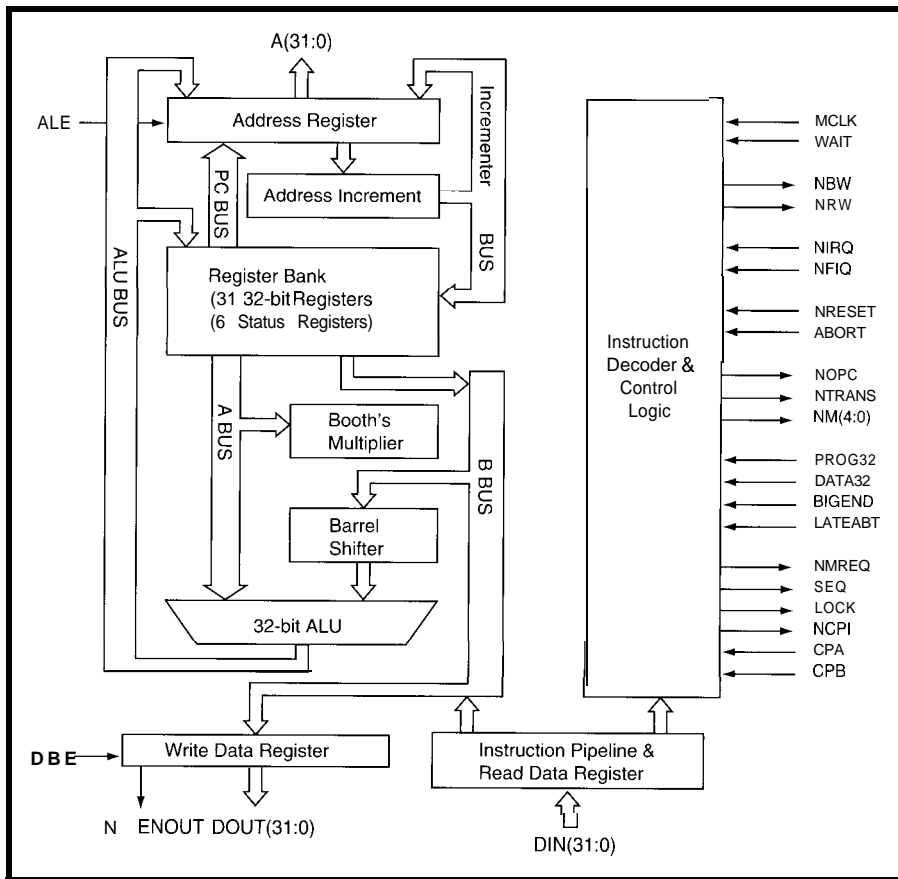


Figure 4—The ARM's unique architectural differences include conditional execution of all instructions, multiple overlapping register banks, a barrel shifter, multicycle multiply and multiply accumulate, and more.

sound generation, communication protocol stacks, or any fast control system requires such specifications.

## WHY A RISC PROCESSOR?

Although the vast majority of 32-bit processors on home computers today are CISC chips from Intel ('486 and Pentium) and Motorola (68020, 68030, and 68040), nearly all workstations use RISC processors. In the PC field, the PowerPC from IBM and Motorola is smaller and just as powerful as the Pentium. It may soon complete in the PC arena what has already occurred with workstations. In 32-bit embedded applications, use of RISC-based chips is increasing because they are smaller and less expensive for a given computing power than their CISC rivals.

## WHY THE ARM?

Most RISC chips are designed for speed and computing power with

multiple execution units, on-chip floating-point units, and very large caches. They have quickly evolved into very complicated devices.

In contrast, the designers of the ARM focused on getting good performance with minimal silicon and power dissipation. ARM's RISC provides more computing power per silicon area and is therefore a cost-effective way to higher performance in many embedded applications.

To illustrate my point, let's look closer. The 32-bit ARM6 boasts 8000 gates while the 8-bit Z80 has 4000. Because ARM6 is the smallest and simplest 32-bit RISC computer, it has

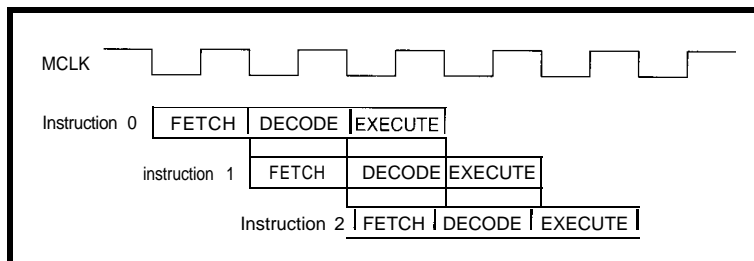


Figure 5—During operation, instructions are fetched into a read data register, decoded, and executed in a three-deep pipeline.

been designed into several mass-market computing machines such as laser printer controllers, medical patient monitors and recorders, cellular telephones, GPS handsets and car navigation, industrial PLC controllers, advanced disk drive controllers, network controllers, fuzzy logic applications, and automotive controllers, to name a few. As well, the ARM60 has now been designed into the 3D0 Interactive Multiplayer (see Figure 2) and the ARM610 is built into the Newton MessagePad (see Figure 3).

## ALL THAT'S NEEDED IS A LITTLE CODE

Along with making a greater commitment to the ARM as an embedded processor, the company also ported the development software to more universally available host computers. ARM now has an assembler, C compiler, linker, librarian, software emulator, remote debugger, and simple monitor program to go on target boards. This software operates on the PC as well as Sun, HP, IBM, and even my NeXT workstation. The PID and PIE development boards use the same host interface and development and debug software so that a program written for the PIE can also be run on the PID.

ARM6, unlike most other RISC chips, is easy to program in assembler and is a good target for compilers. My own experience indicates that writing assembly code for the ARM is easier than writing for the 80x86. (After all, the ARM has no separate I/O space, no segments, and no selectors; it's Just Plain Flat!) Translating code from 80x86 to ARM generally leads to a code growth of up to 20% in byte count while generating half the number of instructions.

## ARM6 CORE

The photograph of the ARM60 (Photo 1) and the ARM6 core block diagram (Figure 4) show the basic structure of the ARM6 core. There is a single register bank with one write and two read



ports. In one cycle, executing instructions can access two registers and write to a third register. Instructions are fetched into a read data register, decoded, and executed in a three-deep pipeline (see Figure 5).

Because of the pipelining, data processing instructions that don't access external memory (i.e., register based) are executed at a rate of 1 per clock cycle, even though each individual instruction takes 3 clock cycles to pass through the processor stages. (This type of pipelined structure is common to most RISC processors.)

Instruction address generation is performed by a dedicated address incrementer. When nonsequential addresses are generated (loads, stores, and branches), the address is taken from the ALU output.

The ARM's unique architectural differences include conditional execution of all instructions, multiple overlapping register banks, a barrel shifter in line with a B input bus to the ALU, multicycle multiply and multiply accumulate, powerful addressing modes for loads and stores, multiple register load and stores, and fully interlocked operation.

## RISC INSTRUCTION SET

The ARM is a variant of basic RISC architecture which can be loosely characterized as having:

- all instructions the same size (this makes direct pipelining possible)
- all data processing (calculations-add, and, etc.) done on registers in the chip
- all memory transfers restricted to load instructions. All memory reads transfer data from memory to a register; all memory writes transfer data from a register to memory.

\*instructions that usually take one cycle to execute. The ARM uses one of 16 registers (RO-R15) in the basic instruction format. Most instructions use three operands (source Rn, source Operand2, and destination Rd). The Operand2 field contains 12 bits which encode several immediate or register fields, some using the barrel shifter.

Listing 1—In an example of a computed jump where the next address is located in a memory-based table, the PC is used as an operand when the address of the jump table is calculated and as a destination when the data is loaded from memory and placed into the PC.

```

; RO enters with table Index (limited to size of jump table)

ADR   R1,=JumpTable ; Compute the address of the Jump Table
                        ; Done by the assembler by adding an
                        ; offset to the PC and writing it to R1
LDR   PC, [R1, RO , LSL #2]

; PC = content of memory at address [R1 + RO * 4]
; The ALU calculates the address of the memory to be loaded
; using preindexed addressing
; LSL = logical left shift in barrel shifter

JumpTable
DCD   JUMP_0
DCD   JUMP_1
DCD   JUMP_2
      etc.

```

The program counter (R15) can be used for a source or destination address. When the program counter is incremented by 4, the pipeline can run unblocked. But, when the next PC value is calculated or loaded from the ALU, the pipeline is restarted from the new address, and the old fetched and

decoded instructions are canceled. This feature enables construction of computed jumps, including elaborate jump tables, and PC-relative addressing of variables (needed for position independent code modules).

Listing 1 offers an example of a computed jump in which the next

*Position and/or Velocity*

# **Motion Control**



**MODELDC2**

*Highly integrated, state-of-the-art Digital Servo Controller provides*  
**MAXIMUM PERFORMANCE at MINIMUM COST**

<ul style="list-style-type: none"> <li>• <b>\$895*</b> complete with powerful software (*Qty 1, OEM discounts available)</li> <li>• 2 Axes DC Servo control,               <ul style="list-style-type: none"> <li>a) ±10 VDC, 12-bit DAC output, and/or</li> <li>b) Ia @ 12/24v direct motor drive (PWM)</li> </ul> </li> <li>• 2 Axes Stepper control (indexer)</li> <li>• Master/slave, electronic gearing, circular/elliptical contouring</li> <li>• Position, Velocity, Torque control</li> <li>• Use in PC/XT/AT/ISA-bus, or Stand-alone</li> <li>• Is not dependent on PC to do its job</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Interactive Servo-Tuning</b></li> <li>• 32K non-volatile RAM, 32K ROM</li> <li>• <b>Opto-isolated</b> inputs: limit switch (4ea), coarse home (2 ea), index pulse (2ea)</li> <li>• 16 Digital I/O, and 4 A/D inputs</li> <li>• 1.0 Mhz incremental encoder inputs, with single-ended and differential receivers</li> <li>• On-board RS232 communications interface</li> <li>• Control up-to 32 axes with multiple boards</li> <li>• Custom firmware available for OEMs</li> </ul>
--	---



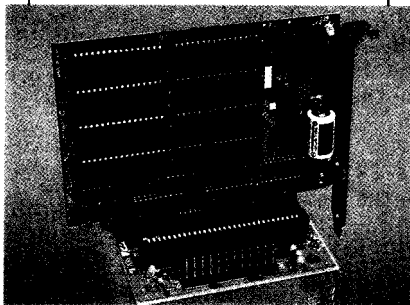
**Precision MicroControl**

C O R P O R A T I O N

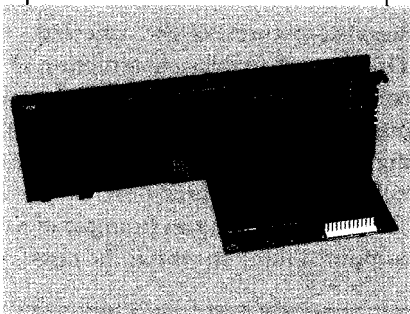
6122 Engineer Road • San Diego, CA 92111

TEL. (619) 565-1500

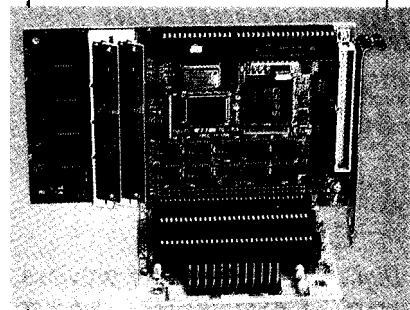
FAX (619) 565-1511



**SOLID STATE DISK — \$135\***  
 1/2 Card 2 Disk Emulator  
 EPROM, FLASH and/or SRAM  
 Program/Erase FLASH On-Board  
 1M Total, Either Drive Bootable



**25MHZ 386DX CPU — \$695\***  
 Compact AT/Bus or Stand Alone  
 On-Board SVGA, IDE, FDC, 2 Ser/Bi-Par  
 FLASH&RAM Drives to 2.5M  
 Cache to 128K, DRAM to 48M



**TURBO XT**  
**w/FLASH DISK — \$266\***  
 To 2 FLASH Drives, 1M Total  
 DRAM to 2M  
 Pgm/Erase FLASH On-Board  
 CMOS Surface Mount, 4.2" x 6.7"  
 2 Ser/1 Par, Watchdog Timer

All Tempustech VMAX products are  
 PC Bus Compatible. Made in the  
 J.S.A., 30 Day Money Back Guarantee  
 \*QTY 1, Qty breaks start at 5 pieces.

**TEMPUSTECH, INC.**  
**TEL: (800) 634-0701**  
**FAX: (813) 643-4981**

ax for 295 Airport Road  
 ast response! Naples, FL 33942

USER	FIQ	SUPERVISOR	ABORT	UNDEFINED	IRQ
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_sup	R13_abt	R13_und	R13_irq
R14	R14_fiq	R14_sup	R14_abt	R14_und	R14_irq
R15	R15	R15	R15	R15	R15

CP SR	SPSR_fiq CPSR	SPSR_sup CPSR	SPSR_abt CPSR	SPSR_und CPSR	SPSR_irq CPSR
-------	------------------	------------------	------------------	------------------	------------------

Figure 6—Most of the ARM's registers are shared by all operating modes. However, each mode has a unique R14 and R13 to avoid the overhead of storing and switching stack pointers.

address is located in a memory-based table. In this listing, the PC is used as an operand when the address of the jump table is calculated and as a destination when the data is loaded from memory and placed into the PC.

### OPERATING MODES

The ARM6 has six operating modes: User, Supervisor, Abort, Undefined, Interrupt (IRQ), and Fast Interrupt (FIQ). The processor is normally operated in the user mode, which has the lowest priority and restricted access in many applications. This is especially useful in PCs and workstations since it would prevent user code from causing the whole system to crash.

Other modes can be entered by an exception, software interrupt, illegal instruction (what Apple uses for a software interrupt), external interrupt, MMU or memory controller abort, or reset. An unavailable coprocessor also causes an illegal instruction mode switch.

### MULTIPLE OVERLAPPING REGISTER BANKS

Figure 6 shows the programmer's view of the ARM6 register set. Each operating mode has a unique R14 and R13 to avoid the overhead of storing and switching stack pointers. R14 is the dedicated link register which stores the program counter of the calling or interrupted routine, and R13

Data Processing	Cond	0 0	I	OpCode	S	Rn	Rd	Operand 2				
Multiply	Cond	0 0 0 0 0 0		A	S	Rd	Rn	Rs	1 0 0 1	Rm		
Swap	Cond	0 0 0 1 0		B	0 0	Rn	Rd	0 0 0 0 1 0 0 1	Rm			
Load/Store	Cond	0 1	I	P	U	B	W	L	Rn	Rd	Offset	
Undefined	Cond	0 1 1		OpCode Area to be decoded by Undefine Routine					0	OpCode_und		
Multiple Reg Load/Store	Cond	1 0 0	P	U	S	W	L	Rn	Register List			
Branch	Cond	1 0 1	L	Branch Offset								
Co-Proc Data Tram	Cond	1 1 0	P	U	N	W	L	Rn	CRd	CP#		
Co-Proc Data Op	Cond	1 1 1 0		Cp Code		CRn	CRd	CP#	CP	0	CRm	
Co-Proc Reg Trans	Cond	1 1 1 0		Cp Code		L	CRn	Rd	CP#	CP	1	CRm
Software Interrupt	Cond	1 1 1 1		OpCode Area to be decoded by Supervisor Routine								

Figure 7—The ARM instruction set consists of all 32-bit instructions and is arranged in order of increasing opcode

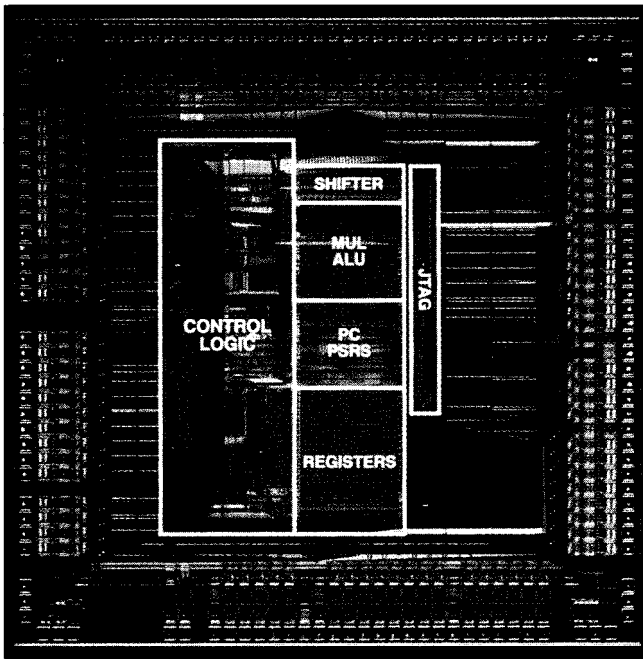


Photo 1—The 1-0.6-p ARM60 die contains a CMOS core, 32-bit address bus, PSR registers, pad ring, and JTAG.

is used as a stack pointer. In addition, the FIQ mode has unique R8\_fiq–R12\_fiq registers to speed up the fastest interrupt responses. Each nonuser mode also has a Saved Processor Status Register (SPSR), which stores the previous processor status and enables returns to the proper calling or interrupted mode. (In the more perfect and symmetrical ARM2 golden age, PSR bits were tucked into the upper 6 bits and lower 2 bits of R15 leaving only 24 bits in the address field.)

## BARREL SHIFTER

Barrel shifter facilitates very powerful instructions. First of all, it

provides shifts of magnitudes between 0 and 31 in each direction in the B-addressed register. Second, it shifts immediate fields, so one instruction can be used for immediate values larger than a byte or word if the number of bits set or added fits in a byte shifted by an even number of bits. Third, the barrel shifter enables indexes to be scaled by a binary multiplier in address calculations. Shift amounts

can also be in a fourth register. The shifts' types are discussed in the working of Operand 2 in the data processing instructions. (Refer to the ARM Data Manual for full details.)

The ARM instruction set depicted in Figure 7 consists of all 32-bit instructions. They are arranged in order of increasing opcode (with some exceptions).

## CONDITIONAL EXECUTION

The ARM6 has a 4-bit conditional field in all instructions. The conditions refer to the values stored in the Current Processor Status Register (CPSR). To make this feature more useful, most of the data processing

0000	0	EQ	Z	set	ALU=0
0001	1	NE	Z	clr	ALU!=0
0010	2	CS	C	set	Carry = 0
0011	3	CC	C	clr	Carry = 1
0100	4	MI	N	set	Bit 31=1 (negative)
0101	5	PL	N	clr	Bit 31=0 (positive)
0110	6	VS	V	set	(over flow)
0111	7	VC	V	clr	(no overflow)
1000	8	HI	C	set and Z clr	(unsigned higher)
1001	9	LS	C	clr and Zset	(unsigned lower or equal)
1010	A	GE	(N set and Vset) or (N clr and Vclr)		greater or equal
1011	B	LT	(N set and Vclr) or (N clr and V set)		less than
1100	C	GT	Z clr and ((N set and Vset) or (N clr and Vclr))		greater than
1101	D	LE	Z set or (N set and Vclr) or (N clr and V set)		less than or equal
1110	E	AL	Always		
1111	F	NV	Never-do not use		

Figure 8—The first 4 bits of every instruction are the condition field. They offer most normal compare functions used in execution control in user programs

# AMX™

## The Real-Time Multitasking Kernel

680x0, 683xx  
80x86/88 real mode  
80386 protected mode  
i960® family  
R3000, LR330x0  
Z80, HD64180

### Features

NEW  
DOS Compatible  
File System  
TCP/IP

- Full-featured, compact ROMable kernel with fast interrupt response
- Preemptive, priority based task scheduler with optional time slicing
- Mailbox, semaphore, resource, event, list, buffer and memory managers
- Configuration Builder utility eases system construction
- InSight™ Debug Tool is available to view system internals and gather task execution statistics
- Supports inexpensive PC-hosted development tools
- Comprehensive, crystal clear documentation
- No-hidden-charges site license
- Source code included
- Reliability field-proven since 1980

Count on KADAK.  
Setting real-time standards since 1978.

For a free Demo Disk  
and your copy of our excellent AMX  
product description, contact us today.

Phone: (604) 734-2796  
Fax: (604) 734-8114

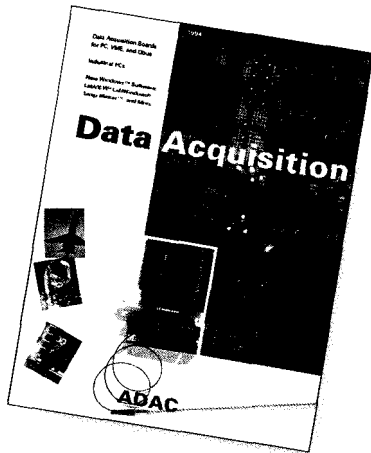


**KADAK Products Ltd.**  
206 - 1847 West Broadway  
Vancouver, BC, Canada V6J 1Y5

AMX is a trademark of KADAK Products Ltd.  
All trademarked names are the property of their  
respective owners.

# NEW Data Acquisition Catalog

Covers expanded low cost line.



**FREE!**

1994 120 page catalog for **PC, VME**, and **Qbus** data acquisition. **Plus** informative application notes regarding anti-alias filtering, signal conditioning, and more.

NEW Software:  
**LabVIEW®**, **LabWindows®**,  
**Snap-Master™**, and more

NEW Low Cost I/O Boards

NEW Industrial PCs

NEW Isolated Analog and Digital Industrial I/O

New from the inventors of plug-in data acquisition.

Call, fax, or mail for your free copy today.

**ADAC**

American Data Acquisition Corporation  
70 Tower Office Park, Woburn, MA 01801  
Phone: (800) 648-6589 Fax: (617) 938-6553

#123

## Important ARM History and Dates-from Acorn's Point of View

In 1982, Acorn Computer invented ARM. The company produced the 6502-based BBC Micro, a product similar to the Apple II in function and performance. Acorn realized the limitations to the 6502 and searched for a replacement in 1983.

For technical, aesthetic, and ethnic reasons, Acorn decided to develop their own 32-bit RISC processor using the outstanding talent provided by Cambridge University. The first versions of the ARM used the coprocessor on the BBC Micro, just as the CP/M card was used on Apple II. ARM operated through a small dual-port memory and communication device. The operating system, user interface, and I/O devices continued to be connected through the BBC main computer. It soon became apparent that the ARM was vastly outperforming the 6502.

After Italy's Olivetti purchased Acorn in 1985, a new ARM-based computer was needed to upgrade or replace the BBC Micro. The companion chips-MEMC (memory controller), IOC (I/O controller), and VIDC (Video controller)-were designed and a completed ARM-based Archimedes was introduced in 1987. The software for the new computer was a direct outgrowth of the BBC. The operating system (called Arthur) preserved system calls, file system, and a BASIC interpreter which was similar to the IBM PC and closely resembled preceding CP/M machines (much of the first Archimedes ROM was written in BASIC!). Since 1987 was post-Macintosh, the original Archimedes also included a window-based desktop.

The evolution of the ARM proceeded in both hardware and software. The original ARM1 processor, built in 3- $\mu$ m CMOS and operating at 6 MHz, was replaced by ARM2, a 2- $\mu$ m, 12-MHz system, which was used in the A310 Archimedes. In 1989, ARM3 came out with 4 KB of built-in cache (capable of operating faster than available DRAMs) and 1.5- $\mu$ m silicon. Independent, fast memory clocks decoupled the processor and cache from the slow DRAM and I/O. After several die shrinks, ARM3 now operates at 33 MHz with the original MEMC, IOC, and VIDC chips.

RISCOS, replacing the original Arthur operating system, included in built-in ROM a complete drawing package (similar to PostScript) and a more fully integrated window environment called WIMP (Windows, Icons, Menus, and Pointers). Acorn was even able to offer the X-Windows display system through RISCIX, a port of Berkeley UNIX.

Advanced RISC Machines Ltd. (see article for details) launched ARM610, a 32-bit core integrated with MMU, cache, write buffer, and coprocessor interface. ARM610 provided a lower price and smaller package by eliminating the coprocessor interface. ARM60, without built-in cache and MMU, was incorporated into the 3D0 interactive multiplayer.

Currently, ARM licensees are seeking applications for the ARM6 and ARM7 core processors in all sorts of embedded applications. The small size and computing efficiency of the ARM core enables integration of entire computing systems on one chip. The 86C650 Laser Printer Controller is one such example.

## ARM Time Line

- 1983 work begins on ARM instruction set and architecture
- 1985 ARM1 prototypes are working
- 1987 Archimedes is launched with the ARM2 processor and chip set
- 1989 ARM3 processor with cache and ARM2 with static ARM core
- 1990 ARM Ltd. founded from Acorn's advanced design division
- 1991 ARM600 samples delivered
- 1992 ARM Ltd. signs Plessey and Sharp as ARM licensees
- 1993 Apple launches Newton using ARM610

1993 3D0 launches Interactive Multiplayer with ARM60  
 1994 VLSI produces 86C650 Microcontroller

### Generations of ARMs

- ARM1 3-u CMOS, 32-bit data, and 26-bit address range
- ARM2 2-p CMOS, multiply and multiply accumulate instructions
- ARM3 1.5-0.6-u ARM2 with cache and cache control logic
- ARM2 1.5-1-p CMOS static ARM core and swap instruction
- ARM6 1-0.6-u embeddable static CMOS core with 32-bit addresses and PSR registers
- ARM60 ARM6 with pad ring and JTAG
- ARM600 ARM6 with 4-KB cache, MMU, write buffer, coprocessor port and JTAG
- ARM610 ARM600 without coprocessor port
- ARM700• ARM7 with 8-KB cache, MMU, write buffer, and JTAG with coprocessor port
- ARM710• ARM700 without coprocessor port
- Prototypes only

instructions have an S bit controlling the setting of the CPSR bits. Thus, a string of instructions may be affected conditionally by a previous instruction. If the condition is not satisfied, the instruction is skipped, costing an execution cycle.

The instruction skip takes less time than performing a branch, which invalidates the two instructions just fetched and causes an additional three-cycle delay before the first new instruction reaches the execution phase. Even though the condition codes use about 10% of the code space, the loss in code density is made up for by the prevention of many forward branches and improved processor speed. See Figure 8 for the condition code table and Listing 2 for examples of the use of condition codes in the performance of absolute value and 4-bit nybble to ASCII conversion.

### DATA PROCESSING INSTRUCTIONS

Most instructions in a program are of the form shown in Figure 9. The Cond field is the condition code already discussed. The S bit enables the setting of the conditions in the particular instruction. The instruction operates on Rn and Operand 2 and writes it into Rd. Rd and Rn or Operand2 can refer to the same register. The instruction **ADD R6, R6, R7** adds R7 to R6 and stores it in R6.

Operand 2 is very flexible. It can be an immediate byte constant encoded into the instruction or an 8-bit constant shifted an even number of positions by the barrel shifter. Large numbers with a small number of set bits can be generated within one instruction and combined with the data processing instruction to improve code density. It can also designate a register as in the ADD example above or be used as a register shifted by a constant. The shift amount can also come from the lowest 5 bits of a third source register.

Normal shifts include:

- \*logical shift left-the register designated in Operand2 is shifted left 0-31 bits and the right bits are filled with zeros
- \*logical shift right-the register designated in Operand2 is shifted right 0-31 bits and the left bits are filled with zeros
- \*arithmetic shift right-the register designated in Operand2 is shifted right 0-31 bits and the left bits are filled with bit 31
- rotate right-the whole register is rotated right by the immediate value

### MULTIPLY AND MULTIPLY ACCUMULATE

The ARM6 has a built-in 2 x 16-bit Booth's algorithm multiplier and a

PLA which perform 16-bit unsigned multiplication with a 32-bit result. This is not strictly RISC, as the instruction can take up to 17 cycles, but results in higher code density than doing it stepwise. Many multiplication instructions using small fixed constants can be performed and executed more quickly with data processing instructions and the barrel shifter. For instance, the instruction **ADD R5, R5, LS L #2** multiplies R5 by 5.

### POWERFUL LOAD/STORE ADDRESSING MODES

The ARM6 uses the RISC model of accessing memory only through loads and stores. The address of the register to be loaded or stored is calculated from the contents of a base register and an optional index. The index can be plus or minus a 12-bit constant, the contents of another register, or the contents of a register that has been shifted. The index can be added to the base register before or after the load or store or it can be added to the base register. A side benefit of such powerful addressing modes is more compact code.

To perform a byte load in single-transfer mode, the addressed byte is loaded into the least-significant byte of the register. The upper 24 bits of the register are zeroed. Data in an ARM system is fixed to the respective byte lane so that bytes must be shifted to the appropriate location with the barrel shifter. If byte 1 in a data word of 0x11223344 is loaded with the LDRB instruction, the loaded register will be 0x00000033 for the little-endian address mode.

To perform a byte store, the least-significant byte is replicated on the output data four times. The memory controller reacts to the NBW high to generate a write strobe in the correct byte lane. Thus, if a register was 0x55667788, the write data would be 0x88888888.

The ARM6 has another multiple-cycle instruction that provides storing and loading multiple registers in heaps or stacks as the programmer desires. The stacks can be ascending or descending, empty or full, thereby enabling the passing of parameters,

saving a state, or preserving registers which are overflowing available register space. Up to 16 registers can be saved or restored in one instruction. Only whole words are loaded and stored. Compare Listing 3's example of the use of these instructions for procedure entry and return with a typical 'x86 listing which has separate instructions for each stack push or pull.

## FULLY INTERLOCKED OPERATION

The ARM6 does not use the branch delay or load delay slots available in other RISC chips for several reasons:

- it makes assembler programming very difficult (although sometimes one has to get to that level for maximum efficiency)
- the chip is busy during these. There are potential problems in aborting during the branch delayed slot since the PC points to the branch address, but the instructions point to the old address. Therefore, an extra copy of the PC has to be kept around for recovery purposes.
- the ARM uses the ALU to calculate the return address and to place it in R14 in the branch and link instruction. (BL is used for subroutine calls.) At least half the time, the compiler places a NOP in the delay slot anyway, wasting the program space.

## ARM ADDRESS SPACE AND SPECIAL VECTORS.

The ARM6 directly addresses  $2^{32}$  bytes or approximately 4 GB as one linear space. I hope that is enough for your application! I had thought that the 26-bit address ARM with 65 MB was enough for PCs and most embedded controllers, but it wasn't enough for Apple.

There are a few special addresses called *exception vectors*, located in the beginning of the address space. A

Opcode[Cond][S] Rd, Rn, Operand 2	
<b>Possible opcodes</b>	
<b>AND</b> Rd = Rn AND Operand2	<b>SUB</b> Rd = Rn - Operand2
<b>EOR</b> Rd = Rn EOR Operand2	<b>RSB</b> Rd = Operand2 - Rn
<b>ADD</b> Rd = Rn + Operand2	<b>SBC</b> Rd = Rn - Operand2 + C - 1
<b>ADC</b> Rd = Rn + Operand2 + C	<b>RSC</b> Rd = Operand2 - Rn + C - 1
<b>TST</b> Cond = Rn AND Operand2	<b>TEQ</b> Cond = Rn EOR Operand2
<b>CMP</b> Cond = Rn - Operand2	<b>CMN</b> Cond = Operand2 - Rn
<b>ORR</b> Rd = Rn OR Operand2	<b>MOV</b> Rd = Operand2
<b>BIC</b> Rd = Rn AND NOT Operand2	<b>MVN</b> Rd = NOT Operand2

Figure 9—Most instructions in a program do data processing and have a similar form

branch to the appropriate software exception handler is usually placed at these locations, although any instruction may be placed there. In the PID board, the instruction loads the PC directly from a branch table located slightly higher in memory, thereby avoiding the restrictions of a 32-MB branch range.

The reset vector, located at address 0x00, is the most important. The ARM starts at this address after NRESET is deasserted. It is important that the ROM is located here after a hardware reset. Sometimes after the chip environment is set up, RAM is switched into the low-memory space so the other exception vectors can be modified by software.

The undefined vector at address 0x04 (as well as any coprocessor that is not available) switches execution to this location. Apple uses this vector as their software interrupt.

The software interrupt vector is at address 0x08 and is used for operating system services just as the INT instruction is in the PC. Acorn has hundreds of SW I instructions already defined.

The abort prefetch and abort data vectors are at addresses 0x0C and 0x10. An abort is generated by an MMU or memory controller if the address accessed is not available or illegal. The ARM monitor software comes with a code module that can unwind and restart aborted code.

The out-of-range vector is located at address 0x14. The ARM2 and 26-bit address modes use this vector for an address greater than 65 MB.

The IRQ or interrupt vector is at address 0x18. When the external IRQ pin is held low and IRQ is enabled, the ARM jumps to this vector. The IRQ code checks for the source of the interrupt, jumps to the appropriate interrupt handler, and resets the particular cause before exiting.

The FIQ or fast interrupt vector is at address 0x1C. Since this is the last entry in the vector table, the FIQ code can start without a jump for the fastest possible response. The FIQ acts in place of other system's DMA resources.

## WHY IT HASN'T BEEN EASY TO DO AN ARM PROJECT

The ARM60 is quite different from typical CPUs or microcontrollers. For example, the chip is lacking IORD and IOWR pins because the chip is meant to be *controlled* by a memory controller which in turn has responsibility for generating the other (normal?) control signals seen by RAM,

Listing 2—Calculating absolute values and converting a 4-bit nybble to ASCII are simplified by use of the ARM's condition codes.

```
;Absolute_Value Code
MOVVS    RO, RO          ; set the condition codes with the S bit
RSBMM   RO, RO, #0      ; if result is M (bit 31 set)
                                ; then RO = 0 RO (positive value)

;Nybble_to_ASCII 0-9 -> 0x30-0x39, A-F ->0x41-0x46
; enter with RO restricted to 0x0 to 0xF

ADD      RO, RO, #0x30    ; add 30h to nybble to make 0-9 ascii
CMP      RO, #9          ; is it over range
ADDGE   RO, RO, #7       ; add 7 more to create A-F
```

## LOAD and STORE addressing modes

Below are examples of the powerful addressing modes generated by the ARM instruction set:

**Normal mode**-A register is stored at an address determined by another register. For example:

LDR R0, [R1]—Load register 0 from the address pointed to by register 1.

**Pre-increment modes**-In these modes, the address of the data is calculated from adding or subtracting the base register to or from an offset or index register. The following examples illustrate several forms.

LDR R0, [R1, #Offset]—Load register 0 with a pointer made from register 1 added to or subtracted from an immediate offset of 12 unsigned bits ( $\pm 3FFh$ ) in the instruction word.

LDR R0, [R1, R2]—Load register 0 with a pointer made from register 1 combined with register 2.

LDR R0, CR1, R2, LSR #2]—Load register 0 with a pointer made from register 1 combined with register 2 and shifted right 2 bits.

LDR R0, [R1, #Offset]!—The addition of the exclamation point indicates that the base register is loaded with the calculated data address. Thus, this form loads register 0 with a pointer made from register 1 combined with Offset and then loads base register R1 with the calculated address.

**Post-increment modes**-In these modes, the offset is added to or subtracted from the base register after the data is loaded or stored. For example:

LDR R0, [R1], #offset—Load register 0 with a pointer of register 1 then adds Offset to R1.

LDR R0, [R1], R2—Load register 0 with a pointer to register 1 and then combines with R2 to R1.

LDR R0, [R1], R2, LSR #3—Load register 0 with a pointer to register 1 and then adds R2 shifted right 3 places.

ROM, and I/O devices. The memory controller can be made from two or three GALs in the case of a static RAM system. The designer then adds a set of peripheral chips in the final board that are addressed in memory space as if they were static RAM

### MAKING IT EASY (OR EASIER—IT'S NEVER THAT EASY)

As hardware applications manager, one of my first tasks was to build development cards so the customers would not have as steep a learning curve as I did. Through this endeavor, I was able to design, build, and program the ARM.

PID, the first Platform Independent Development board, used the ARM600, which includes the ARM6 cell with 4 KB cache, an MMU, and a write buffer. In addition, the ARM600 has a coprocessor port for use with a floating-point coprocessor or another customer-invented coprocessor.

PID features 1 to 16 MB of DRAM, 128 KB to 4 MB of EPROM, QuickLogic PGA-based memory controller, and an interrupt controller. The output of the memory controller uses the standard Intel IORD and IOWR pulses, and a 16C551 serial/parallel port chip is onboard. Communication to the host computer is

accomplished through the serial port. Once the application is developed, it can be transported to the EPROM and live independently of the host. Expan-

sion to the customer's application is provided by an AT-like slot, which has proved popular with our ARM developers.

## Video Frame Grabber

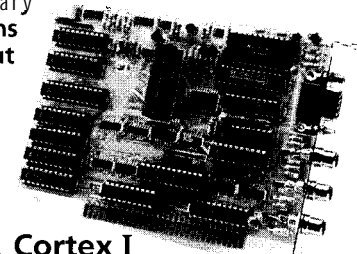
- 5495 Including Software with "C" Library
- Half Slot Card for Compact Applications
- Real Time Imaging with Display Output
- 8 Bit (256 Gray Levels)

### FEATURES:

- Single 512 X 484 Image or Four 256 X 242 Images
- External Trigger
- Input Look Up Table
- Low Power Option Available
- Elegant Software Interface
- <10 nsec Pixel Jitter Means Accurate Digitization
- EISA (PC) Bus and STD Bus Products Available
- RS-170 and CCIR Video Formats Available
- Binary and TIFF File Formats

The Cortex I is used in machine vision, industrial control, medical, security and scientific applications around the world.

ImageNation strives to delight customers with quality products and personal service at a competitive price.



Cortex I

Call today for volume pricing or to discuss your application.

P.O. Box 276  
Beaverton, OR 97075  
(503) 641-7408

**ImageNation Corporation**  
Providing Imaging Solutions  
FAX (503) 643-2458 • (800) 366-9131

Listing J-Program example using *the load multiple and store multiple instructions of the ARM. In this listing, the program is getting a byte from the 8250 equivalent serial port in the PID board.*

```

GetByte
  STMDB   sp!,{r2,lr}   ; save regs r2, and lr(r14) on the stack
                                ; (r13) with multiple register store
                                ; instruction
  MOV     r2,#IOBase    ; place IOBase (480 0000)
                                ; in r2 using rotated immediate byte

GetByteLoop
  LDRB   r0,[r2,#IRQ0]; put status of RX in r0 w/load byte opcode
  TST   r0,#SCCRX      ; see if a character has been received
                                ; TST = opl AND op2 with no write back
                                ; conditional branch
  BEQ   GetByteLoop    ; get the character
  LDRB  r0,[r2,#RBR]   ; Multiple Register Load restores r2 and
  LDMA  sp!,{r2,pc}   ; loads value that was in link register
                                ; (r14) into the PC to continue program
                                ; execution at the instruction after the
                                ; call

```

Meanwhile in England, ARM developed the PIE (or Platform Independent Evaluation) board, which has 128 KB of SRAM, one 128-KB EPROM, and two GALs to run and decode memory. The serial port is provided by a Signetics 269 1. Expansion is somewhat more difficult with this board as it has only the HP-compatible logic analyzer ports for add-on circuitry.

I am currently working on a faster replacement for this board (to be called NPIE) which takes advantage of the latest 40-MHz ARM60 parts that have just come out of the ovens.

## ARM PERFORMANCE MEASUREMENTS

The ARM6 has impressive performance. Table 1 shows a few of the standard benchmarks we ran comparing an 80486DX2-66 PC with the PID board running at 24 MHz.

Yes, the '486DX2-66 can run faster at some benchmarks, but it's 10 times the power and cost. However, without cache enabled, the '486 processor shows half the performance of the ARM60 because of its small and irregular register set requiring more loading and unloading of resources.

<u>Dhrystone</u>	
486DX2-66 with int. cache, 256 KB ext. cache	40,000
486DX2-66 with caches off	4,000
24-MHz PID with internal 4 KB cache	27,000
24-MHz PID with internal cache off	8,000
<u>Bit-BLIT</u>	
486DX2-66 with int. cache, 256 KB ext. cache	15.3 ms
24-MHz PID with internal 4 KB cache	11.9 ms
24-MHz PID with internal cache off	36.5 ms
<u>Vector Drawing</u>	
486DX2-66 with int. cache, 256 KB ext. cache	0.337 ms
24-MHz PID with internal 4 KB cache	0.402 ms
24-MHz PID with internal cache off	1.283 ms
<u>Temperature</u>	
486DX2-66	Too hot to touch for long
ARM610	Barely warm
<u>Price of one</u>	
486DX2-66	Retail >\$500
AM610	~\$50

Table 1—The ARM610 running on the PID board is capable of giving the '486DX2-66 a run for its money.

## CONCLUSIONS

ARM processors give excellent performance at low power and low cost. You can have cost effective 32-bit RISC performance for embedded applications and home projects. The ARM's small set of instructions is very powerful because of the unique processor architecture. Programmers also tell me the ARM is fun to program. (I have always heard this reaction from converted Intel x86 and 680x0 programmers.)

My next article will cover building an

ARM60-based board that will be capable of 20,000 Dhrystones or more with inexpensive cache SRAMs. I will also touch on other ARM development boards and products from third-party vendors. □

**Art Sobel is the hardware applications manager for embedded products at VLSI Technology. He has spent 24 years in Silicon Valley designing disk drive electronics, disk drive controllers, laser interferometers, laser printer controllers, many controller chips, and speech synthesizers. He can be reached at [sobel\\_a@vlsi.com](mailto:sobel_a@vlsi.com).**

## REFERENCES

van Someren, Alex and Carol Atack, **The ARM RISC Chip, A Programmer's Reference Manual**. Addison-Wesley (1993), ISBN 0-201 40695-0.

ARM60 Data Manual  
ARM610 Data Manual

## SOURCES

VLSI Technology  
18375 South River Pkwy.  
Tempe, AZ 85284  
(602) 752-6630  
Fax: (602) 752-6001

Other suppliers of ARM processors and information:  
GEC Plessey Semiconductors  
1500 Green Hills Rd.  
Scotts Valley, CA 95066  
(408) 438-2900

Cheney Manor  
Swindon  
Wiltshire  
United Kingdom SN2 2QW  
(0793) 51800

Sharp  
5700 NW Pacific Rim Blvd.  
Camas, WA 98607  
(800) 642-0261

407 Very Useful  
408 Moderately Useful  
409 Not Useful



# Feeling Out a Braille Digital Clock

## FEATURE ARTICLE

Wayne Thompson



Most people think of Braille as raised bumps on paper.

Certainly, paper Braille is still in widespread use today as a means of communication for persons who are blind or visually impaired. But, the last decade has delivered to the blind community a wealth of high-tech equipment including the "refreshable Braille display."

This remarkable innovation produces temporary lines of Braille by raising and lowering small metal or plastic pins in the proper dot patterns to produce one line of text in Braille. After reading the line, the user commands the device to display the next line of text. The pins instantly raise or lower to present the next line in a "refreshed" Braille display.

Refreshable Braille displays can be connected to personal computers or other electronic devices to provide immediate display of text, analogous to the visual screen for sighted persons. The advantages are obvious. No wasted paper. No noisy Braille embossers. No storage of bulky Braille books. No waiting. Just the same instant access to information as for sighted folks. (Well, almost the same. There still is no such thing as a full-page refreshable Braille display.)

Over the years, single-line refreshable Braille displays have appeared in a number of commercial products from companies specializing in devices for the visually impaired. One application that has been overlooked is a Braille digital clock. No doubt, this is due to the fact that talking clocks and traditional time-pieces with flip-up covers and tactile

markings have been readily available for under \$50. A Braille digital clock with a refreshable Braille display is neat, but not likely to be a big seller when the h-character Braille display alone costs over \$300. (Braille displays typically cost about \$50 per character and are available as single-piece units consisting of 1-80 characters.)

Nevertheless, when something is needed on the job, higher cost can often be justified. Such was the case at the Kentucky Department for the Blind, a state government rehabilitation agency that assists persons who are blind or visually impaired to find and maintain employment,

### "STAND BY, FOR NEWS!"...

Many blind and visually impaired people throughout the country are employed by commercial radio stations. One essential duty of disc jockeys, news announcers, and commercial spot producers is accurate detection of time.

Counting down to a network news break at 2:00:00 P.M. or producing a 30-second spot demands 1-second precision. Sighted people in radio just watch the second hand or digits of an ordinary clock, and visually impaired persons have traditionally used an old style table clock with motors and gears which slowly turn wheels. The clocks are modified by placing tactile dots on the surface of each of three wheels showing hours, minutes, and seconds. Space on the wheel circumference limits tactile resolution to the nearest five seconds. Worst of all, these mechanical clocks are no longer available and parts are becoming too scarce to maintain them.

An inexpensive talking clock is what most visually impaired people use today. But for a broadcaster, talking on the air, monitoring a soon-to-be broadcast network feed, and listening to a talking clock would be a bit much-not to mention the risk of accidentally broadcasting a background robotic voice speaking the time of day. Clearly, a silent Braille digital clock is the ticket.

Since we could not discover such a thing commercially, we designed and built one.

The Braille digital clock offers persons who are blind or visually impaired an alternative to talking clocks and obsolete motor-driven wheel clocks with tactile markings.

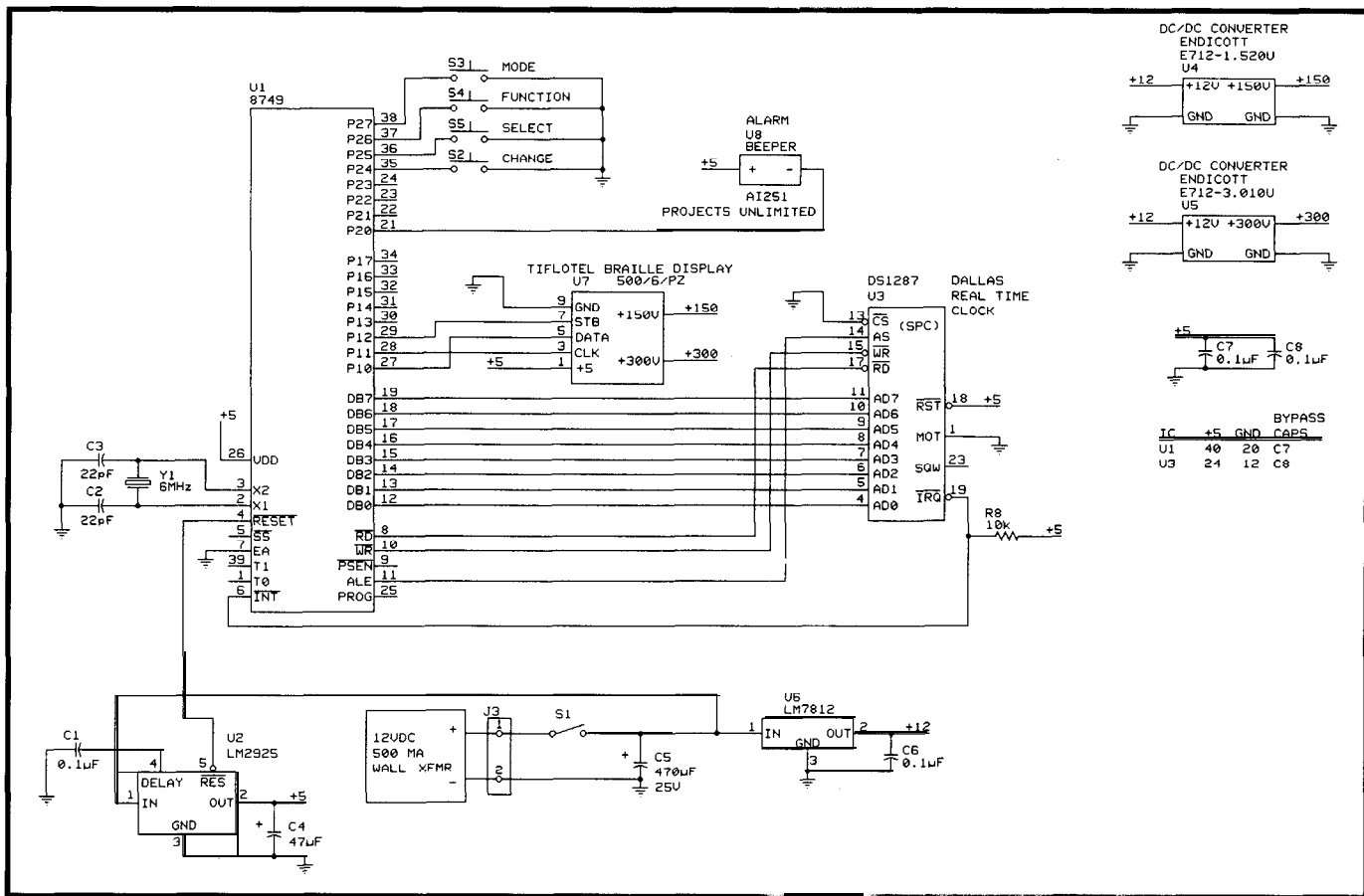


Figure 1—The 8749 single-chip microcomputer reads time and date from the DS1287 real-time clock and updates the six-character Tiflotel Braille display each second.

## YOU CAN HELP

Currently, three broadcasters in Kentucky who are blind are using this device daily on the job. The motor-driven clocks they had used for years broke beyond repair. Undoubtedly, the next few years will find many others throughout the country in the same predicament. Due to the high cost of refreshable Braille displays, the potential market for a Braille digital clock/calendar is too small for most manufacturers to consider building one. Consequently, the prospect of a commercially available unit is dim. That's where you come in!

Such a small but vital need can only be met by hand building the units as needed. Every state has a rehabilitation agency with funds available for the purchase or fabrication of equipment necessary to get or maintain employment for persons with visual impairments. Many state agencies do not have the in-house technical staff to build a project such as this Braille digital clock but could contract the job

out to a competent technician or engineer.

A quick call to your state's rehab agency for the blind might land you some moonlighting work or at least make yourself known as a technical "gun for hire." And, you'll be providing a much appreciated service to your fellow earthlings with visual impairments who are, like the rest of us, becoming increasingly dependent on technology.

## LET'S GET TECHNICAL

As can be seen from Figure 1, the Braille digital clock/calendar is quite simple from a hardware standpoint. Most of the work is done in firmware.

The time and date are kept by the Dallas Semiconductor DS 1287 real-time clock chip. This chip contains on-board, nonvolatile memory for over lo-years operation in the absence of power. The 8749 single-chip microcomputer is interrupted once each second by the real-time clock. During each interrupt, the firmware stored

within the 8749's on-board EPROM reads the time and updates the Tiflotel six-character Braille display unit. The program also checks for a match to the preset alarm time and activates the alarm beeper if enabled. The four user buttons are constantly monitored for user commands which perform various functions such as setting time, date, A.M./P.M., alarm on/off, and weekday.

The six Braille characters (called cells) usually display hours, minutes, and seconds with two cells for each, starting from the left (HHMMSS). Of course, the unit's seconds digits change once each second as time marches on. When the user invokes various other modes and functions with the push buttons, the display instantly shows the corresponding items such as date (MMDDYY), A.M./P.M., weekday name, and certain keywords such as View or Set. The pins instantly raise and lower silently and are not affected by a finger resting over a cell when it changes (as some Braille displays are).

The unit can be powered from a 12-VDC, 500-mA wall transformer. The LM2925 provides regulated 5 V and a reset pulse for the 8749. The 12-V regulator is needed to supply a constant voltage to the input of the high-voltage DC/DC converters. These two devices from Endicott provide the 150-V and 300-V levels (at low current) needed by the piezoelectric crystal elements within the Tiflotel Braille display. When not in use, switch S1 can power down the microprocessor and Braille display; the DS 1287 continues to keep accurate time.

The standard software could be modified for special applications. Depending on the application, the six-character Braille display could be replaced by any length model from 1 to 80 cells with no hardware changes since Braille display data bits are sent serially and clocked into each cell through a shift register arrangement. Only a software change would be needed to accommodate a different length Braille display.

## THE BRAILLE DISPLAY

The six-character refreshable Braille display module used in this project is a type 500/6/PZ piezoelectric and was manufactured by Tiflotel in Italy. It comes with a built-in hybrid

circuit for activation of the piezoceramic elements on which Braille dots are mounted. In addition to the piezoceramic elements requiring 300 V at 10 mA and 150 V at 20 mA, the user must also supply 5 V to the Tiflotel unit to power the integral 74HC4094 8-bit SIPO shift-latch register controlling each cell.

This arrangement provides for a very simple 3-wire user interface: Data, Clock, and Strobe. Data is sent to the Braille display serially on the Data line (pin 5). As shown in Listing 1, once the Data line is set high or low as required, the normally low Clock line (pin 3) is pulsed high to shift the data bit into the Braille display's internal shift registers. Subsequent Clock pulses shift data from dots 1-8 corresponding to bits 0-7. Data continues to shift through all cells from cell 6 to cell 1, the leftmost cell.

Once all 48 data bits have been shifted into the Braille display (6 cells times 8 bits each), the normally low Strobe line (pin 7) is pulsed high to latch all bits and cause all Braille dots to instantly raise or lower corresponding to the new data just stored. Timing constraints shown in Figure 2 are simply those specified for the integral 74HC4094 latching shift registers. These limits are easily met by the

relatively slow access of the 8749's firmware.

Since the S-bit Braille code does not match the ASCII code, the `br1_it` subroutine that controls the Data, Clock, and Strobe lines also performs a Braille code table look-up for all ASCII characters by calling a `scbrl` subroutine. Braille displays larger (or smaller) than six cells could be easily accommodated by minor changes to the `br1_it` subroutine.

## OPERATING THE CLOCK

If the firmware's only task were to read the time from the clock chip and update the Braille display, the code would be relatively simple. But, the seemingly universal obsession of humanity to control things forces upon every invention that innocent-sounding addition called "user input."

Even a clock must be set. Those four user push buttons look so simple on the schematic. But adding software to support them doubles the source-code length and reaffirms my most often proven corollary to Murphy's Law, "The last 10% of a project will take 90% of your time."

As described below, the four push buttons (Mode, Function, Select, and Change) enable users to set or view various features of the Braille clock/

## An Economical Braille Display Continues to Elude

Currently, sighted people can purchase a high-resolution, color video monitor for a few hundred dollars. People who are blind must pay thousands of dollars for a one-line, twenty-character Braille display. A full-page Braille display, equivalent to a video screen's 80 characters by 25 lines, doesn't even exist.

For many years, the search for a new method or technology which would lead to the mass production of an inexpensive, refreshable Braille display has been pursued. So far, the two most popular approaches include displays with their pins activated by either solenoid or piezoelectric crystal action. Either approach results in a cost of about \$50 per Braille cell. At that rate, an 80 x 25 Braille display would cost \$100,000 (about the cost of some past prototypes).

The close spacing of adjacent dots and cells coupled with the need to raise and lower each dot individually poses a considerable engineering challenge. The sheer number of pins to actuate, especially for a full-page design, multiplies complexities.

Lots of existing technologies seem to have potential.

Magnetics, pneumatics, piezoelectric, purely mechanical (gears, cams, etc.), solenoids, micromotors, embossing/erasing on paper or plastic, rotating wheels (like an odometer), and so on. Dozens of ideas have been proposed and many tried. But still, no winners. Funds exist to pursue promising ideas. The National Federation of the Blind has a Research and Development Committee which evaluates and explores the potential of every idea brought to their attention.

Maybe one of the new exotic technologies is the answer. There are phase-transition polymer gels which exhibit light-induced volume changes. There are new metals which expand in response to applied voltage. Electro-rheological fluids change from liquid to solid with applied voltage.

Who knows where the ultimate answer lies? Researchers often discover technologies that yearn for applications, waiting only for someone to say, "Hey, that could be used to do this!" Or, it could turn out to be something so simple that everyone will remark, "Why didn't I think of that?"

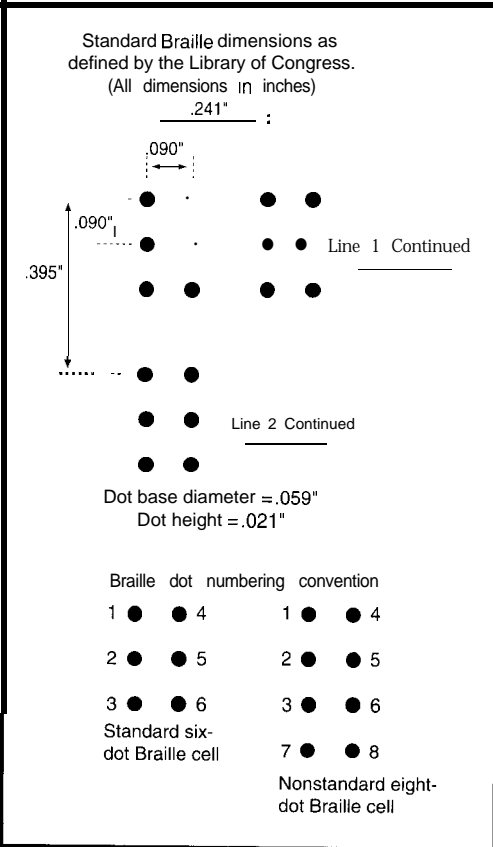
## What is Braille?

Braille is a system of writing and reading first defined by Louis Braille. The system uses tactile raised dots to represent characters. The six dots of a Braille "cell" (one symbol) are arranged and numbered as shown in the figure below. This allows  $2^6$  or 64 possible dot combinations. Elaborations, such as prefixes and multicell symbols, have extended literary Braille

usage into other fields such as music, mathematics, and chemistry.

The Library of Congress has defined the standard dimensions for Braille. The Braille produced by many printers and displays does not have these exact dimensions, but some degree of variance is acceptable.

Additional dots 7 and 8 are sometimes added to extend the adopted Braille code. There is no standard for 8-dot Braille. It has become popular in refreshable Braille displays because computers often need to display more than 64 symbols. Dots 7 and 8 can be used to indicate things like control characters, underlining, case, upper ASCII character (bit 7 high), cursor location, italic character, and so on.



another, press and release the Change button while holding down the Function button. The Braille display alternately reads Time, Alarm, and Date.

The time function means that the current time is displayed as six digits. The leftmost two digits are hours, the middle two digits are minutes, and the rightmost two digits are seconds. Of course, one or more cells will automatically change each second.

The alarm function means the current alarm setting is displayed as six digits. The format is the same as the time function.

Date function means that the current date is displayed as six digits. The leftmost two digits are month, the middle two digits are day of month, and the rightmost two digits are year.

These three functions may be viewed or set depending on the currently selected Mode.

## THE SELECT SWITCH

The Select switch selects various items for setting or viewing. It works differently in Set mode versus View mode.

While in View mode, each press of the Select button displays the following items one at a time: current function (i.e., time, date, or alarm), A.M./P.M., alarm on/off, and weekday. You cannot change the value of these items while in View mode.

While in Set mode, the Select button works a little differently. Each depression of the Select switch moves to the next Braille cell to the right for setting. The cell currently selected is indicated by raising dots 7 and 8. If you want to change the value of the cell currently selected, each depression of the Change button will increment that cell's value by one. You continue pressing Change until the digit becomes what you want. Then you press Select to move to the next digit on the right. In this way, all six cells may be set, one at a time, to the desired time, date, or alarm.

There is one exception to the above procedure. If the rightmost digit is selected while in Set Time mode, then pressing the Change button forces that digit to be zero rather than

calendar. Unless noted otherwise, all references to push-button depressions are intended to mean press and release that button by itself. Immediately after powering on, the Braille display shows the current time and begins incrementing each second.

## THE MODE SWITCH

The Mode switch selects one of two modes: View or Set. To see which mode is currently active, press and hold the Mode switch. The Braille display will read either View or Set. The default mode after power up is View. In View, you can only view items; there's no danger of accidentally changing any item. Set mode allows you to set various items using the other push buttons (described later). To change from one mode to the other,

you press and release the Change button while holding down the Mode button. The Braille display alternately reads View and Set.

Requiring such a deliberate action to get from View to Set mode ensures that clock values will never be changed accidentally by inadvertent switch depressions. The Mode switch is also used to silence a sounding alarm (described later).

## THE FUNCTION SWITCH

The Function switch selects one of three different functions: Time, Alarm, or Date. To see which function is currently active, press and hold the Function switch. The Braille display reads either Time, Alarm, or Date. The default Function after power up is Time. To change from one Function to

incrementing it. This facilitates setting the clock to the exact second. Simply set the other five digits first, then Select the unit seconds digit (rightmost cell). Press Change at the exact instant your reference timepiece reaches zero in the unit seconds column (which, of course, happens every ten seconds).

There are a few more items to Set in addition to the six digits. While setting the time, if the rightmost digit is already selected and the Select button is pressed again, the display shows either A.M. or P.M. Press the Change button to toggle between A.M. and P.M.

Press Select again and the display shows either Alm On or Almoﬀ indicating the alarm status. Press Change to toggle between Alm On and Almoﬀ.

Press Select again and the display shows one of the seven weekdays. Press Change repeatedly to step through all seven weekdays.

Pressing Select one more time takes you back to the starting place, which is the leftmost Braille digit (indicated by dots 7 and 8 raised on cell 1). You may go through any or all of the Select items again until they are all set the way you want them.

When setting the alarm, the weekday choices are skipped. If the alarm is on, the clock will emit a high-

pitched tone each day when clock time matches the previously set alarm time. Be sure to correctly set alarm A.M./P.M. setting also. (Alarm on/oﬀ can also be set from within the set time function). To silence a sounding alarm, simply press the Mode switch.

When setting the date, the weekday, A.M./P.M., and alarm on/oﬀ choices are skipped.

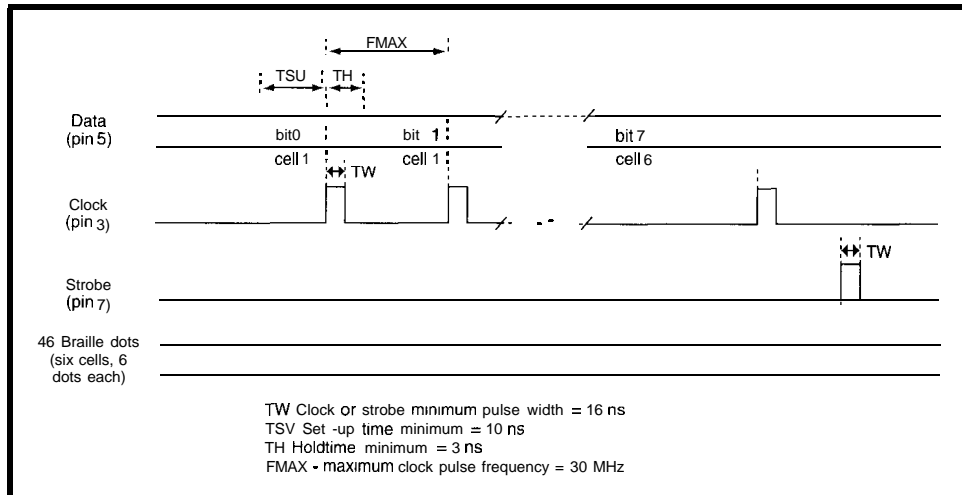
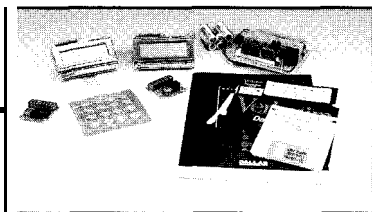


Figure 2—Timing specs for the Tiflotel Braille display's integral 74HC4094 SIPO registers are fast enough that a user perceives an instantaneous change in the six Braille characters.

## Batteries Included



Mid-Tech's ec.25 offers the ultimate in flexibility and extensibility for your specialized embedded computing needs. Specifically designed for battery operation, this small (4" by 4" by 1") computer is capable of extremely long run times due to its built-in power management capability.

The included PC hosted IPL utility allows serially downloading executable programs. A number of ready-to-run applications are included on diskette along with source code for all peripheral drivers and a number of useful utilities written in both assembler and C.

The ec.25 accommodates an unparalleled complement of I/O options for a variety of data collection, monitoring, and control tasks.

- 8031 Processor (8031 compatible)
- 64K Lithium-backed RAM
- Bootstrap Loader with PC-based IPL Utility
- RS-232 and CMOS RS-485 Ports
- 12C LCD/Keypad Port
- RTC/Timer with 256 bytes RAM
- 512 Bytes EPROM
- 8-channel 8-bit ADC
- 2-channel 8-bit DAC
- 16-bit Bidirectional DIO
- Dual Switch-Mode/Pass-Mode Voltage Regulators
- Full Featured Battery Manager/Fast Charger
- 120 Binary I/O Points over Twisted Pair
- NiCd Battery and Line Power Options Included

The full-up ec.25DK developers kit is \$495.00 (single quantity). Call for OEM pricing and availability of special configurations

Mid-Tech Computing Devices  
 P.O. Box 218, Stafford, CT 06075-0218  
 Voice/fax: (203) 684-2442

## SAVE YOUR ISSUES OF THE COMPUTER APPLICATIONS JOURNAL

A durable custom-made binder is now available to store and protect your copies of Circuit Cellar INK, The Computer Applications Journal. These binders feature a special spring mechanism to hold individual issues securely in place. Each binder can hold up to twelve issues.

To order your binder, call Circuit Cellar, Inc. at (203) 875-2751 or fax us at (203) 872-2204

**\$12.95 EACH** (Visa and MasterCard accepted, add \$4 domestic s&h, CT residents add 6% sales tax, add \$6 outside USA. U.S. funds only)

**CIRCUIT CELLAR, INC.**  
**ATTN: BINDER OFFER**  
**1 PARK STREET**  
**VERNON, CT 06066**

## DISABILITY AWARENESS

The recently passed Americans with Disabilities Act (ADA) has spawned a new public awareness of the many ways in which individuals with disabilities are discriminated against, often unintentionally. Projects like this Braille digital clock are custom designs which attempt to overcome

the information denial inherent in products designed without regard for sight-impaired users. It's astounding to realize that something equivalent to a common digital watch is not available to the blind population.

However, custom devices and modifications are expensive. Most individuals cannot afford the engineer-

ing costs involved in creating such specialized access devices. Typically, their only recourse is rehabilitation agencies who use public funds to provide technical services to individuals who qualify for assistance. Often their employment is at stake. They want the opportunity to be competitive with their sighted coworkers. Relatively minor accommodations, like a Braille digital clock for a radio DJ, can make that happen.

The readers of this magazine are the kind of people who could really help-engineers, technicians, tinkers, inventors, all-around technical problem solvers, and project builders. The application of technology for the disabled is an exploding field with new problems and solutions occurring daily. There are sources of funding existing for exploring these new ideas. Be a pioneer. Turn on that soldering iron, your thinking cap, and give us a call if you come up with any great ideas!

Wayne Thompson is an electrical engineer working for the past 13 years at the Kentucky Department for the Blind, a rehabilitation agency serving persons who are blind or visually impaired. He can be reached at Kentucky Department for the Blind, 427 Versailles Rd., Frankfort, KY 40601, (502) 573-4754, or (502) 573-3976 (fax).

Listing 1—The `bra1_it` and `ascbrl` routines directly control the strobe, data, and clock lines of the Braille display module.

```
;Send the six ASCII bytes in the BRLDAT buffer to the 6-cell
;braille display.

bra1_it  mov r1,#BRLDAT  ;init buffer pointer
         mov r3,#1      ;init for 6 bytes to send (start
                       ; with bit 0 high and shift left)
bra3     mov a,@r1      ;get ASCII data
         call ascbrl    ;convert to braille
         mov r2,a       ;save braille char
         mov r0,#CMODE  ;in SET mode?
         mov a,@r0
         jb0 bra5       ;no, jump
         mov r0,#CSELECT ;yes. This cell SELECTED for SET?
         mov a,@r0
         xrl a,r3
         jnz bra5       ;no, jump
         in a,p2        ;MODE or FUNCTION key down?
         cpl a
         jb7 bra5       ;yes, jump
         jb6 bra5       ;yes, jump
         mov a,r2       ;no, recover char
         orl a,#0c0h    ;raise bits 6 & 7 (dots 7 & 8)
         jmp bra4
bra5     mov a,r2       ;recover char
bra4     mov r2,#8
bra0     rrc a
         jc bra1
         anl p1,#0feh   ;set data bit low
         jmp bra2
bra1     orl p1,#01h    ;set data bit high
bra2     orl p1,#02h    ;raise clock line to send bi
         anl p1,#0fdh   ;lower clock line
         djnz r2,bra0   ;send remaining bits
         inc r1         ;point to next ASCII digit
         mov a,r3       ;adjust cell counter
         rl a
         mov r3,a
         jb6 bra6       ;jump if done
         jmp bra3
bra6     ;all bits are now in the braille display
         orl p1,#04h    ;raise strobe line to write a 1 cells
         anl p1,#0fbh   ;lower strobe line
         ret

*****
;Enter with ASCII code (32-95 only) in accum. Exit with
;equivalent braille code in accum.

ascbrl   mov r2,a       ;save ASCII code
         mov a,#32     ;subtract 32 from ASCII code
         cpl a         ;(form one's complement)
         add a,#1      ;(form two's complement)
         add a,r2      ;add to get difference)
         movp3 a,@a    ;get braille code from TABLE
         ret
```

## CONTACT

Tiflotel  
24032 Calolziocorte (BG)  
Italy

## SOFTWARE

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

## IRS

410 Very Useful  
411 Moderately Useful  
412 Not Useful

## DEPARTMENTS

56 Firmware Furnace

64 From the Bench

70 Silicon Update

76 Embedded Techniques

84 ConneCTime

# Journey to the Protected Land

## Booting into Protected Mode



Now that  
Ed's  
covered  
the basic

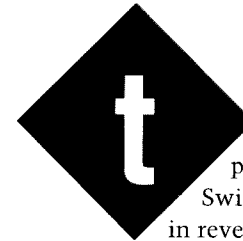
of protected-mode operation, it's time to revisit the disk boot process. With this month's code, you'll be able to insert a floppy into your embedded '386SX and end up in protected mode.

## FIRMWARE FURNACE

Ed Nisley

*So, naturalists observe, a flea  
Hath smaller fleas that on him prey  
And these have smaller still to  
bite 'em;  
And so proceed ad infinitum.*

Jonathan Swift, *On Poetry*, 1733



he PC boot process resembles Swift's chain of fleas in reverse. The BIOS

starts from ROM and invokes its bootstrap loader routine, which loads the first sector from disk or diskette, which loads the operating system's loader, which loads the OS itself, which loads your program, which loads your data. With any luck, you eventually get some useful work out of these trained fleas.

In the last three issues, I've investigated 32-bit protected mode using a monolithic program read into RAM by the diskette boot sector loader. Having both real- and protected-mode code in one file simplified the discussion, but the real-mode part of the program isn't of much interest. Wouldn't It Be Nice If we could simply write a PM program, copy it to diskette, and run it on the '386SX?

This month I'll make that possible by cracking the code into two parts: a loader and a "pure" 32-bit PM program. The loader flea starts in real mode, finds the PM flea, reads it into RAM, then branches to it in 32-bit protected mode. The PM flea will evolve into the Firmware Furnace Task Switcher, which will eventually spawn little PM fleas of its own.

Listing I-This code reads the diskette's boot sector, copies the data directly into the BootBlock structure, then computes the values needed for the DriveBlock structure. Both structures are elements of arrays of structures, but PMLoader uses only the first element, which corresponds to Drive 0. Although there isn't room to show the structures, the Teutonic naming conventions indicate what the variables do for a living.

```

int DiskInitialize(int Drive) {
union REGS regs;
struct SREGS sregs;
int NumTries;
BIOSPB *pBoot;
DRVPB *pDisk;
BYTE far *fpBuffer;

fpBuffer = farmalloc(DEFAULT_SECTORSIZE);
if (NULL == fpBuffer){
    DiskError("*** Cannot allocate diskette read buffer!".0);
    return 1;
}
for (NumTries=1; NumTries <= MAX-ATTEMPTS; ++NumTries){
    regs.h.ah = 0x02;           // read sectors
    regs.h.al = 0x01;         // one at a time
    regs.h.ch = 0;           // track 0
    regs.h.cl = 1;           // sector 1
    regs.h.dh = 0;           // head 0
    regs.h.dl = Drive;       // from the selected drive
    regs.x.bx = (WORD)FP_OFF(fpBuffer);
    sregs.es = (WORD)FP_SEG(fpBuffer);
    int86x(0x13,&regs,&regs,&sregs); // read the sector
    if (!regs.x.cflag){       // leave loop if it works!
        break;
    }
}
if (regs.x.cflag){
    DiskError("*** Cannot read diskette boot sector!",regs.h.ah);
    return regs.h.ah;
}
_fmmemcpy(MK_FP(_DS,&BootBlock[Drive]),fpBuffer,sizeof(BIOSPB));

pBoot = &(BootBlock[Drive]); // set up local pointer
pBoot->DriveID = Drive;       // set correct drive number

/*- compute "DOS" parameter block entries */

pDisk = &(DriveBlock[Drive]);
pDisk->DriveID = Drive;
pDisk->Unit = 0;
pDisk->SectorSize = pBoot->SectorSize;
pDisk->MaxSectorInCluster = pBoot->ClusterSize - 1;
pDisk->ClusterShift = LogBase2[pBoot->ClusterSize];
pDisk->NumReserved = pBoot->NumReserved;
pDisk->NumFATs = pBoot->NumFATs;
pDisk->MaxRootFiles = pBoot->MaxRootFiles;
pDisk->FirstRootDirSector = pBoot->NumReserved +
    pBoot->NumFATs * pBoot->FATSize +
    (WORD)pBoot->HiddenSectorsL;
pDisk->FirstDataSector = pDisk->FirstRootDirSector +
    (((pBoot->MaxRootFiles * sizeof(FATDIRENTRY)) +
    pBoot->SectorSize) - 1) / pBoot->SectorSize;
pDisk->MaxClusterNum = ((pBoot->NumSectors
    pDisk->FirstDataSector + 1) / pBoot->ClusterSize) - 1;
pDisk->FATSize = pBoot->FATSize;
pDisk->MediaID = pBoot->MediaID;
pDisk->Status = 0;
pDisk->pNextDPB = NULL;
pDisk->FirstFreeCluster = 0;
pDisk->FreeClusters = 0xFFFF;

return 0;
}

```

Most of this column is devoted to the real-mode code that processes a diskette's directory and File Allocation Table structure. I'll concentrate on reading data from the diskette because that's what we need for the loader, but extending the code to write data is, as usual, a simple matter of firmware. Even if you're not interested in fleas, the information should be useful.

## BASIC BOOTING

I last discussed the PC's boot sequence in *CA/31*, so a quick review will set the stage for the loader's task. There's not enough room for a listing, but the boot loader source is included in this month's BBS files.

The story begins after the BIOS completes its power-on self tests and initializes all the BIOS extensions. It then executes Int 19 which, unless an extension has captured the vector, transfers control to the BIOS bootstrap loader routine. If all goes well, the BIOS startup flea never regains control.

The default BIOS Int 19 code resets drive A to Track 0 (*gronk!*), reads the first 5 12-byte sector from the diskette into memory, then branches to the first byte at 0000:7C00. What happens next is up to the code in that boot sector. Two fleas down.. .

In addition to executable code, the boot sector contains a table of diskette parameters used by the DOS file system. I'll cover the table in more detail later, but for now, suffice it to say that the boot loader uses it to pull the diskette's root directory into storage. The boot sector code for this column reads just the first sector of the root directory; you can load the whole smash if your code needs it.

Depending on how much work you want to do, the boot sector code can scan the root directory for a specific filename or, as I've done, simply pick the first valid file. The directory entry tells you where the file begins on the disk in units of DOS clusters. The parameter table contains the information needed to convert that value into the physical track, head, and sector values required by the BIOS.

If the file is contiguous and the diskette has no defects, you can simply read successive sectors until you've



loaded the entire file. Those are reasonable restrictions as you'll see in the discussion of the diskette FAT structure, and both DOS and I insist on them. Producing a contiguous file is easy: just copy it from your hard disk onto a good formatted diskette with no other files. That also ensures it's the first file in the directory, although depending on which version of DOS formatted the diskette, the volume label may be the first directory entry.

The boot sector loader plunks the file at address 1000:0000, then branches to the first byte. The file may be an ordinary embedded program similar to the ones we've been using or, as in this column, a more complex loader for yet another program in yet another diskette file.

The only tricky part of the process is putting the boot sector onto the diskette in the first place. That sector is not included in the diskette directory, which means it's not accessible by the ordinary DOS file copy functions. `DEBUG` can read and write every diskette sector using the BIOS functions, so this script does the job:

```
DEBUG
L 100 0 01
N BOOT1440.SEC
L
W 100 0 01
Q
```

The motive for reading the sector with the first `L` command is that some versions of `DEBUG` expect the starting and ending sector numbers rather than the starting sector and number of sectors. You may actually read two sectors on some systems. The second `L` command overwrites the first 512 bytes with the contents of `BOOT1440.SEC`, which is the boot sector program for 1.44-MB 3½" diskettes. The `W` command then writes one or two sectors to the diskette.

The boot sector code for this column blinks bit 7 on parallel port 378 if it encounters a problem. If all goes well, it turns that bit on before branching to the file loaded from diskette. `BOOTSECT.ASM` includes parameters for the four common diskette formats and you get the four

Listing 2—The diskette's directory and File Allocation Table are large enough that they must be dynamically allocated on the far heap. The Drive Parameter Block structure in the `DriveBlock` array provides the information to calculate the storage needed for the directory and FAT, locate their sectors, and read them in.

```
int DiskReadDir(int Drive) {
    int Index;
    SECTORINFO Sector-Info;
    DRVPB *pDisk;

    pDisk = &(DriveBlock[Drive]);

    fpDirectory[Drive] = farrealloc(fpDirectory[Drive],
        pDisk->SectorSize * (pDisk->FirstDataSector
        pDisk->FirstRootDirSector));

    if (NULL == fpDirectory[Drive]){
        DiskError("Can't allocate directory buffer!".0);
        return 1;
    }
    fpFAT[Drive] = farrealloc(fpFAT[Drive],
        pDisk->FATSize * pDisk->SectorSize);

    if (NULL == fpFAT[Drive]){
        DiskError("Can't allocate FAT buffer!".0);
        farfree(fpDirectory[Drive]);
        return 1;
    }
    SectorInfo.Drive = Drive;
    SectorInfo.fpBuffer = fpFAT[Drive];
    SectorInfo.LogicalSector = pDisk->NumReserved; // first FAT sect
    do {
        if (DiskReadSector(&SectorInfo)){// if error, try again
            continue;

            ++SectorInfo.LogicalSector;
            SectorInfo.fpBuffer += pDisk->SectorSize;
        } while (SectorInfo.LogicalSector <= pDisk->FATSize);

    SectorInfo.Drive = Drive;
    SectorInfo.fpBuffer = (BYTE far *)fpDirectory[Drive];
    SectorInfo.LogicalSector = pDisk->FirstRootDirSector;
    do {
        if (DiskReadSector(&SectorInfo)){// if error, try again
            continue;
        }
        ++SectorInfo.LogicalSector;
        SectorInfo.fpBuffer += pDisk->SectorSize;
    } while (SectorInfo.LogicalSector < pDisk->FirstDataSector)

    SectorInfo.fpBuffer = NULL;

    return 0;
}
```

corresponding binary files as well, named `BOOT*.SEC`.

Three fleas down, two to go.. ..

## ONCE BOOTED, TWICE READ

By definition, the boot sector code must fit within the first 512-byte diskette sector. The program it reads from diskette isn't so tightly constrained; as long as it's within our boot sector loader's 64-KB capacity, any-

thing goes. I took advantage of this space by writing `PM Loader in C` and sending readable diagnostic messages to the serial port.

`PM Loader` starts by rereading the boot sector to get the diskette's parameter table, the directory to find the target file's name and starting point, and the File Allocation Table so it can handle a fragmented file. The first two are already in RAM, but it's

**Listing 3—Converting from the logical sector numbers in the BPB and DPB into the physical track, sector, and head values required by BIOS diskette I/O functions requires values from the BPB. I collected all the information required to identify a logical sector into a SECTORINFO structure to simplify passing the values from one function to another. Note that physical sector numbers start with one on each track, while logical sectors start with zero at the beginning of the diskette: head 0, track 0, sector 1 is logical sector 0.**

```
int DiskCvtSector(SECTORINFO *pSI) {
    BIOSPB *pBPB;

    pBPB = &(BootBlock[pSI->Drive]);

    pSI->Sector = 1 + (pSI->LogicalSector % pBPB->TrackSize);
    pSI->Head = (pSI->LogicalSector / pBPB->TrackSize) %
                pBPB->NumHeads;
    pSI->Track = (pSI->LogicalSector / pBPB->TrackSize) /
                pBPB->NumHeads;

    return 0;
}
```

better to reread the diskette. The extra credit project this month gets PM Loader into RAM without invoking the boot sector loader. Beware of clever shortcuts!

Although the diskette parameter table doesn't contain all the values needed for a DOS-compatible file I/O system, I didn't have to invent many new wheels. The DOS Drive Parameter Block described in Shulman's *Undocumented DOS* contains the additional values DOS uses in one (relatively) tidy structure. We don't need all of them for the loader code this month, but many of them will come in handy when we extend the code later on.

DOS copies the parameter table into an internal structure called the BPB (BIOS Parameter Block), then computes the DPB (Drive Parameter Block) values. The `DiskInitialize()` function in Listing 1 imitates this process, although some of the more esoteric values that we don't need right now probably don't match their DOS counterparts.

`BootBlock` (the BPB) and `DriveBlock` (the DPB) are arrays of structures so we can eventually handle more drives by just bumping the size of the arrays. The arrays in this code have only a single element, which means if you *really* need more than one drive, you'd better test the code!

The FATs (there are `NumFATs` identical copies, although DOS assumes all disks and diskettes have

two copies regardless of BPB) start just after any reserved sectors following the boot sector. The `NumReserved` field, which includes the boot sector, is thus the FAT's starting sector number, and `FATSize` is the number of sectors in one copy of the FAT.

Similarly, `FirstRootDirSector` and `FirstDataSector` mark the start of their respective diskette areas, while their numeric difference is the size of the root directory in sectors. Each directory entry is 32 bytes long, so a 512-byte sector holds 16 entries. `MaxRootFiles` in both the BPB and DPB gives the maximum number of root directory entries, and is always a whole number of sectors.

None of the references indicate where the "hidden sectors" are located, so I arbitrarily stuck them between the FATs and the root directory. Another reasonable spot is between the root directory and the data area. The fact that this isn't specified anywhere tells you how often it's used; diskettes formatted by plain old DOS have no hidden sectors. If any of you folks have good info on this, drop me a note on the BBS and I'll tweak the code to match reality.

Listing 2 reads the first FAT and the entire root directory into RAM. Because these are relatively large blocks of information (the root directory on a 1.44-MB diskette is 7 KB), I allocate space for them on the far heap. The `fpDirectory` and `fpFAT` arrays hold far pointers to the struc-

tures. As with the BPB and DPB arrays, there is one element per drive, and I've only verified the code with that one.

The "sectors" referred to in these calculations are formally known as *logical sectors*. The first logical sector on a diskette corresponds to the boot sector and is number zero. The BPB `NumSectors` value tells how many sectors are on the diskette, so the last logical sector is `NumSectors - 1`.

The BIOS diskette I/O functions, however, identify physical sectors by their track, sector, and head location: the diskette boot sector is at track 0, head 0, sector 1. Yes, physical sectors start at 1 and logical sectors start at 0. Nobody said this was going to be easy.

I put the drive number, logical sector, and physical values into a single `SECTORINFO` structure to simplify the calculations. Listing 3 shows how the conversion works. Entries in the drive's BPB are essential to the calculation, which means the only sector you can locate without the BPB is the boot sector that contains the BPB itself. Referring back to Listing 1, you'll see that I special-cased that by hardcoding the physical values into the BIOS call.

Although it takes quite a while to describe all this, only a few seconds of real time have elapsed since that *gronk*. These fleas are fairly fleet. . .

## FINDING THE FILE

A familiar part of file I/O, regardless of whether you use the C library or program directly with the DOS `Int 21` functions, is opening a file before you use it. I decided to mimic this operation in `PM Loader`, even though it uses only a single file; the extra code will come in handy one of these months. Listing 4 shows how `DiskOpenFile()` works.

Once again, I borrowed a structure from the innards of DOS: the File Control Block (FCB) documented in, of course, *Undocumented DOS*. I've avoided a lot of complexity by omitting the hocus pocus needed to deal with subdirectories; by definition, the file we want is in the root directory.

DOS allocates its internal FCBs as they're needed, but `PM Loader` uses a simpler scheme: the FCBs are ele-

ments of an array called `FileBlock`. In fact, `FileBlock` has but one element, thus only one file may be open at a time. If your application needs more, just make `FileBlock` larger and test the code to make sure it actually works for more than one file.

`DiskOpenFile` converts the file name from a C string into separate blank-padded name and extension fields which are then stored in the next available FCB (which, for `PMLoader`, is the only FCB). Finding the file in the directory is a simple matter of comparing the FCB fields with successive directory entries.

Up to this point, there is little chance for error. Assuming the diskette sectors are readable, the code must find the FAT and root directory at the positions specified by the BPB. However, it's entirely possible that `DiskOpenFile` won't find the file it's looking for, perhaps because you forgot to copy it onto the diskette. Don't snicker—it could happen.

A standard DOS program can prompt you to insert another diskette, request a different filename, or whatever. `PM Loader` takes a simpler approach by displaying a message on the serial port and blinking an error indication on the parallel port. If you're going to boot a 3%bit protected mode program, you've got to put it on the diskette first!

Assuming you've done that, `DiskOpenFile` will locate the appropriate directory entry, copy the directory values into the file's FCB, and return a "file handle" to the calling program. In this case, the handle is simply the address of the File Control Block within the `FileBlock` array. While this doesn't promote absolute information hiding, it's appropriate for a simple-minded embedded program like `PM Loader`.

The directory entry and the FCB contain two essential values: the file's starting cluster number and its size. In the style of Bill Cosby's Noah, "Riiiiight. What's a cluster?"

## FACING THE FATS

Most files on a hard disk or diskette are larger than a single sector. Rather than keeping track of the

*Listing 4—Before using a file you must first locate it in the diskette's directory and compute several values from the directory entry. This function mimics some of the steps DOS goes through to open a file for access by filling in a copy of the DOS File Control Block. `PMLoader`, like older versions of DOS, allocates a fixed number of FCBs to simplify the code, with `NumOpenFiles` locating the current FCB in the array. `PMLoader` doesn't write or create files, but extending this code should be straightforward, if tedious.*

```
void *DiskOpenFile(int Drive, char *pName, int Mode) {
    DOSFCB *pFCB;
    FATDIRENTRY far *fpDir;
    char *pChar;
    int Index;
    int Match;
    char FileSpec[MAX_FILESPEC+1];

    if (NumOpenFiles >= MAX_FILES) {
        DiskError("Can't open another file, too many open already",0);
        pFCB = NULL;
        goto Done;
    }
    pFCB = &(FileBlock[NumOpenFiles]); // get pointer to our block

    memset(pFCB->Name, ' ', sizeof(pFCB->Name)); // set up file name
    memset(pFCB->Ext, ' ', sizeof(pFCB->Ext));
    memcpy(FileSpec, pName, sizeof(FileSpec));
    pChar = strtok(FileSpec, ".");
    memcpy(pFCB->Name, pChar, min(strlen(pChar), sizeof(pFCB->Name)));
    pChar = strtok(NULL, ".");
    if (NULL != pChar) {
        memcpy(pFCB->Ext, pChar, min(strlen(pChar), sizeof(pFCB->Ext)));
    }

    Index = 0;
    do {
        fpDir = fpDirectory[Drive] + Index;
        Match = _fmemicmp(fpDir->Name, MK_FP(_DS, pFCB->Name),
            sizeof(pFCB->Name)+sizeof(pFCB->Ext));
        ++Index;
    } while ((Index < DriveBlock[Drive].MaxRootFiles) &&
        (0 != Match) &&
        ('\0' != fpDir->Name));

    if (0 != Match) {
        DiskError("*** Cannot find file",0);
        pFCB = NULL;
        goto Done;
    }

    pFCB->NumHandles = 1; // fill in the FCB
    pFCB->OpenMode = Mode;
    pFCB->Attributes = fpDir->Attributes;
    pFCB->pDRVPB = &(DriveBlock[Drive]);
    pFCB->FirstCluster = fpDir->StartCluster;
    pFCB->Time = fpDir->Time;
    pFCB->Date = fpDir->Date;
    pFCB->Size = fpDir->Size;
    pFCB->CurrentPosition = 0;
    pFCB->RelCluster = 0;
    pFCB->AbsCluster = fpDir->StartCluster;
    pFCB->DirSector = ((WORD)((DWORD)fpDir
        (DWORD)fpDirectory[Drive]) /
        DriveBlock[Drive].SectorSize) +
        DriveBlock[Drive].FirstRootDirSector

    ++NumOpenFiles; // done, so tick open file counter

Done:
    return pFCB;
}
```

**Listing 5**—DOS manages the space on a diskette in terms of clusters, which are contiguous groups of sectors. Cluster numbers start at 2 and converting them to logical sectors requires values from the diskette's BPB and DPB. This code omits the tracing statements that display end-of-file markers and other special cluster numbers.

```
int DiskCvtCluster(WORD Cluster,SECTORINFO *pSI){
    int Drive;
    Drive = pSI->Drive;
    pSI->LogicalSector = ((Cluster - 2) *
        BootBlock[Drive].ClusterSize) +
        DriveBlock[Drive].FirstDataSector;
    return pSI->LogicalSector;
}
```

**Listing 6**—The File Allocation Table is an array of cluster numbers represented as 12-bit integers. A file's directory entry gives you the starting cluster number, which you look up in the FAT to find the next cluster. This function extracts the 12-bit FAT entry corresponding to a given cluster.

```
WORD DiskGetFATEntry(WORD Cluster,int Drive) {
    WORD FATEntry;

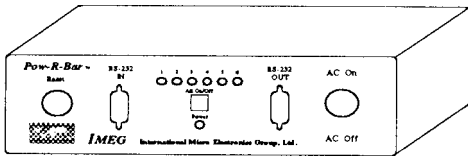
    FATEntry = *(WORD far*)(fpFAT[Drive] + ((Cluster * 3) / 2));

    if (0 == ((Cluster * 3) % 2)){ /* even = low 12 bits */
        FATEntry &= 0x0FFF;
    }
    else { /* odd = high 12 bits */
        FATEntry >>= 4;
    }
    return FATEntry;
}
```

individual sectors, DOS manages the disk using larger units called clusters. A cluster is simply a contiguous group (hence the name) of logical sectors that is always allocated as a unit. The smallest file thus occupies one cluster rather than one sector, and a file that is one byte larger than a cluster requires another whole cluster.

The BPB ClusterSize value gives the number of sectors per cluster. On 360- and 720-KB diskettes, a cluster is two sectors. On the larger 1.2- and 1.44-MB diskettes, a cluster is a single sector. From the previous paragraph you'd expect larger floppies to have bigger clusters, but there's another factor at work.

The diskette's data area (everything other than the boot sector, FATs, and root directory) is divided into clusters that are numbered starting with 2. Listing 5 shows DiskCvtCluster(), which translates a cluster number into a logical sector number. DiskCvtSector() can then produce the physical track, head, and sector needed to read the diskette.



## Pow-R-Bar™ \$149<sup>00</sup> (msrp)

is an intelligent, programmable, six outlet power strip which connects to a computer's serial port and operates via RS-232 protocol. Pow-R-Bar™ is the perfect solution for controlling multiple AC outlets.

With Pow-R-Bar™ connected to a computer, each of the six AC outlets on the back of Pow-R-Bar™ can be turned on/off from the computer, by typing in a simple command or through custom programming.

Up to 26 Pow-R-Bar™s can be daisy chained together providing up to 156 outlets individually controllable from a single computer. With this system, an entire building can be automated.



**IMEG** International  
Micro Electronics  
Group, Ltd.

155 W. Tiverton Lexington, Kentucky 40503  
P.O. Box 25007 Lexington, Kentucky 40524  
800-274-8699 606-271-0017 Fax: 606-245-1798

## C COMPILER FOR PIC CONTROLLERS

- Integrated software development environment including an editor with interactive error detection/correction.
- Access to all hardware features from C
- Includes libraries for RS232 serial I/O and precision delays.
- Efficient function invocation mechanism allowing call trees deeper than the hardware stack.
- Special built-in features such as bit variables optimized to take advantage of unique hardware capabilities.
- Interrupt and A/D built-in functions for the C71
- Easy to use high level constructs:

```
#include <PIC16C56.h>
#use Delay(Clock=2000000)
#use RS232(Baud=9600,Xmit=pin_1,RCV=pin_2)

main 0 {
    printf("Press any key to begin\n");
    getc();
    printf("1 khz signal activated\n");
    while (TRUE) {
        output_high(pin_8);
        delay_us(500);
        output_low(pin_8);
        delay_us(500);
    }
}
```

PCB compiler \$99 (all 5x chips)  
PCM compiler \$99 ('64, '71, '84 chips)  
Pre-paid shipping \$5  
COD shipping \$10

CCS, PO Box 11191, Milwaukee WI 53211  
414-781-2794 x30

The File Allocation Table is an array of cluster numbers with one entry for each cluster on the diskette. Each entry is 12 bits long, which means there can be at most 4096 clusters on any diskette. Some cluster numbers are used as special codes, thus limiting the FAT to only 4078 clusters. Larger clusters mean fewer FAT entries for a given disk size, which means fewer sectors per FAT, which means more space is left for data. *That's* why smaller diskettes have larger clusters.

The first two FAT entries are reserved, which is why cluster numbers start with 2 instead of 0. The first byte of the first entry is a copy of the BPB **Media ID** value. The remainder of the two entries is always FF, although DOS may use the space for some arcane purpose while allocating files.

Now, here's why we need the FAT. The directory gives you the file's starting cluster number. You look that entry up in the FAT to find the *next* cluster number. The entry corresponding to that number gives you the next cluster and you repeat this process until you have located the entire file as defined by the directory's **File Size** field. If after reading the last cluster you look its entry up in the FAT, you'll see a value between FF8 and FFF; those magic numbers are end-of-file markers and don't indicate clusters.

The difficult part about FATs is two 12-bit FAT entries are packed into three 8-bit bytes. Listing 6 shows how to extract a FAT entry given its cluster number index. The **fp FAT** array contains far byte pointers, so the first line must cast the pointer to (**WORD far \***) to extract two bytes from the array. The **if** statement selects the high- or low-order 12 bits from that word.

Now you see why boot sector loaders typically require contiguous files! The directory tells you everything you need to know about a contiguous file, so the boot loader avoids this hassle.

After all that, reading a file is straightforward. Listing 7 is the inner loop of **DiskReadFile0**, which reads successive clusters, sector by sector,

Listing 7—Reading the file requires reading successive logical sectors in each cluster, accessing the FAT to locate the next cluster, and continuing until all of the bytes in the file are accounted for. This loop forms the core of the **DiskReadFile()** function.

```
while (pFile->CurrentPosition < Buffer-Size) {
    if (DiskReadSector(&SectorInfo)) { // bail out on error
        goto Done;
    }
    ++SectorIndex;
    if (SectorIndex >= DriveBlock[Drive].MaxSectorInCluster) {
        SectorIndex = 0; // step to next cluster
        ++pFile->RelCluster;
        pFile->AbsCluster = DiskGetFATEntry(pFile->AbsCluster, Drive);
        if (pFile->AbsCluster >= 0x0FF0) {
            break; // die on "can't happen" numbers
        }
        DiskCvtCluster(pFile->AbsCluster, &SectorInfo);
    }
    else {
        ++SectorInfo.LogicalSector; // next sector in cluster
    }
    SectorInfo.fpBuffer += DriveBlock[Drive].SectorSize;
    pFile->CurrentPosition += (DWORD)DriveBlock[Drive].SectorSize;
}
```

until the total number of bytes read equals or exceeds the actual file size. The loop terminates at that point, so the function will never (urn, *should* never) hit the FAT's end-of-file markers.

**DISKETTE.C** is over 7001 lines long, but you've seen the key parts. Download it from the BBS and spend a while tracing through the structures and logic. Reading a DOS disk file isn't as hard as you think.

Writing a file, on the other hand....

## FLIPPING THE SWITCH

If you haven't forgotten by now, **PM Loader** expects the file it reads into storage to be a 32-bit protected-mode program. The filename hardcoded into **PM Loader** helps ensure this. After all, **FFTS PM0** is hard to mistake for anything other than Firmware Furnace Task Switcher, Protected Mode Ring 0, and it's not a name you're likely to generate by accident.

**PM Loader** allocates a buffer on the far heap big enough for that file. Since the heap in conventional memory isn't a convenient location to run **FFTS**, **PM Loader** moves the code into extended memory at the 1-MB line. Protected-mode code isn't restricted to the first megabyte; I have some plans for that valuable real estate

down there and, well, it was easy to pull off.

Back in *CA/ 45*, I used a BIOS function to transfer dot patterns for the Game of Life into and out of extended memory. In this case, we'll move a 64-KB block of code and data, but the principle is the same. The details are in the source code this month if you need a refresher; I don't have the space to cover it again.

Once the file is in place, **PM Loader** sets up the GDT and IDT needed by the BIOS "Switch to Protected Mode" function. In addition, it creates two descriptors for RAM above the 1-MB line. **GDT [ 8 ]** is a code segment that covers the 64-KB block at 1 MB. **GDT [ 9 ]** is a data segment at 1 MB, but its descriptor has **G=1**, so it covers all of extended memory. Finally, we can use a *big* segment!

**PM Loader** then calls **PMEntry()** to handle the last few details before the switch. That function will never return because we can't run real mode C in protected mode. **PM Entry's** final instruction branches to offset 0 in the 32-bit code segment of **GDT [ 8 ]**.

The 32-bit PM code will set up a new GDT and IDT above the 1-MB line, create data and stack descriptors, and define interrupt gates for its handlers. **PM Loader** knows nothing of this, so we need not alter the real-

mode code when the protected-mode code changes.

The stub of protected-mode code in FFTS.PM0 this month simply puts up a progress indicator on the parallel port LEDs, shows "P3" on the FDB's LED digits, and blinks parallel port bit 5 to show that it's alive. Any errors will pass through the existing IDT on their way to the simple error handler that I discussed earlier.

The final flea is in full effect!

## RELEASE NOTES

The BBS files this month include revised boot sector loaders for the four common diskette formats (although I've only tested the 3% versions), the **PM Loader** program, and the stub version of FFTS. All of PMLoader's tracing options are turned on, so about two screens of diskette info squirt out COM1 during each boot.

Here's an extra credit project for those of you with a EPROM or RAM socket on your Firmware Development Board: convert **PM Loader** into a BIOS extension. This will eliminate the

need for a custom boot diskette and cut several seconds of loading.

If you need help, I covered ISA bus memory starting in *CAJ* 36 and BIOS extensions in *CAJ* 38, 40, and 41. Use the extension entry point at C800:0005 to prepare your code by hooking Int 19, then load the PM program from diskette when the BIOS executes Int 19. Don't forget a push-button escape hatch so you can boot ordinary real-mode program diskettes!

For you Micro-C users, Dave Dunfield wrote *MDCFS*, a Minimal DOS-Compatible File System using Micro-C version 3.0. As always, his "minimal" is entirely adequate for any reasonable task and includes far more functions than the code in this column. For example, you can actually write data to the diskette, which makes data loggers and such a snap.

Next month I'll expand the FFTS kernel by setting up housekeeping above the 1-MB line and adding some utility routines. From now on everything will be 32-bit protected mode code!

*Ed Nisley, as Nisley Micro Engineering, makes small computers do amazing things. He's also a member of the Computer Applications Journal's engineering staff. You may reach him at ed.nisley@circellar.com or 74065.1363@compuserve.com.*

## CONTACT

Dunfield Development Systems  
P.O. Box 31044  
Nepean, ON K2B 8S8  
(613) 256-5820

## SOFTWARE

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

## IRS

413 Very Useful  
414 Moderately Useful  
415 Not Useful

ary Display & (ram Download)

8-bit CPU Analog Input (Temperature) (Humidity) (Flow) (Etc.)

Twisted Pair - Extended System Communications

RS-485

HCS II BASIC SYST

PL-Link Module

TW523

PS123

Buffer/Terminator

4 Park Street, Suite 12 • Vernon, CT 06066  
Tel: (203) 875-2751 • Fax: (203) 872-2204  
Dealer inquiries invited

Energy Management  
Security and Alarm  
Coordinated Home Theater  
Coordinated Lighting  
Monitoring and Data Collection

Get all these capabilities and more with the Circuit Cellar HCS II. Call, write, or fax us for a brochure. Available in assembled or kit form.

Circuit Cellar HCS

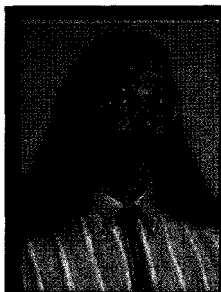
Module To

# FROM THE BENCH

Jeff Bachiochi

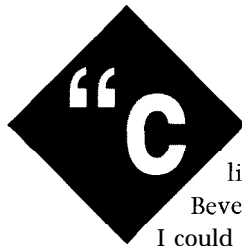
## Celebrate National Cannibalism Week

### Take Your Old Floppy Drives To Lunch



Most people have had to replace

a floppy disk drive at one time or another, but couldn't bring themselves to toss the old one. Jeff dives into some old drives to see just what he can find.



ome here and listen to this,"

Beverly pleaded.

I could tell just from the tone of her voice that I wasn't gonna be a happy camper. I could already hear the cash registers, a nest full of hungry birds screaming to be fed.

"What now?" I asked myself.

"Can't we pay off a few old bills first?" I muttered, this time raising my eyes

toward the skies looking for a little divine intervention.

There was, however, no time to listen for replies. For when I turned down the corridor leading to the laundry room, I heard a racket.

Rubba-rubba. Rubba-rubba.

"Hear that?" she asked, as if I was deaf. Rubba-rubba. Rubba-rubba.

"Phew, when did it start smelling like that?" I shouted. "Turn it off!"

Beverly opened the clothes dryer's door and the clattering subsided. I quickly looked inside as if to catch a glimpse of a leprechaun rapping the tub with his shillelagh.

"OK, I'll take a look," I submitted and left to get some tools.

"I don't want it fixed. I want a new one," Bev retorted. "You've already replaced the heating element twice and the drum bearing wheels once." She paused briefly to let the facts sink in and then returned with the finale. "It's fifteen years old, it's beginning to burn clothes, it sounds like it's going to take off, and it doesn't owe us a thing."

"I didn't know you knew what drum bearing wheels were," I paused to see if I caught her off guard, but no dice. "If it looks bad, we'll trash it," I said with my most honest face. She knew me better than that, but compromised, sort of.

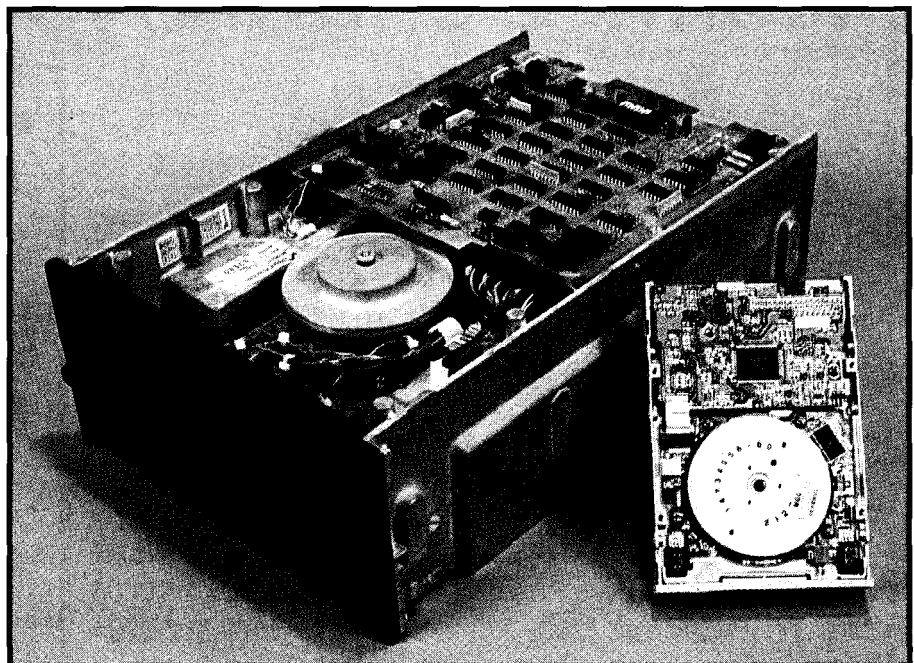


Photo 1—Over the years, large, discrete control electronics has been replaced by tiny, highly integrated circuits.

“One hour. That’s it, and then it’s outta here. Bulk pickup is tomorrow.”

Upon removing the 220-VAC plug, I grabbed the socket set and removed the twenty-odd screws which held on the dryer’s back panel. One look explained it all. The motor mount had broken loose, leaving the motor to vibrate wildly. A very simple arrangement of two tabs, one on each end of the motor mount, once held the mount to the dryer frame. The front tab fit into a slot while the back tab had a bolt hole (now empty) that secured it to the frame. Using a length of steel bar, I reattached the mount to the frame, sliding the front into the existing slot and a bolt into the back. After replacing the 30-odd screws (the things must be breeding!) on the back I turned it on for a test.

Rubba-rubba. Rubba-rubba.

“Thirty minutes!” shouted someone from the other end of the house.

Back off with the 40-odd screws. I play with the drum belt tension, I wiggle the motor shaft looking for worn bearings, I double check the

motor mount. Nothing I do reduces the wild vibrations.

I guess I’ve lost. Then I remember one comment, “It’s beginning to burn clothes.” “That’s strange,” I think to myself. The shaft moves freely.. . Let’s check the other end of the motor.

Off with the motor mount and a couple of extra screws holding it against the exhaust manifold. Removing the motor reveals the true culprit.

Clumps of lint and a single sock

are packed into the vanes of the impeller used to discharge the hot, moist air. The shaft spins free and balanced when spun by hand, but under power, centrifugal forces cause it to vibrate madly. I clean out the rotor and remount the motor. I replace all 50-odd screws holding the sheet metal back and turn it on.

Beverly came back cheerfully singing, “Time’s up. Out it goes.”

“OK, just let me turn it off first.”

Her mouth dropped open. You could hardly hear the dryer. It was no longer walking the floor on its own. She knew there would be no appliance shopping this week.

## INTERNET

Last week on the Circuit Cellar BBS, I read a message asking if there was anything worth saving in a

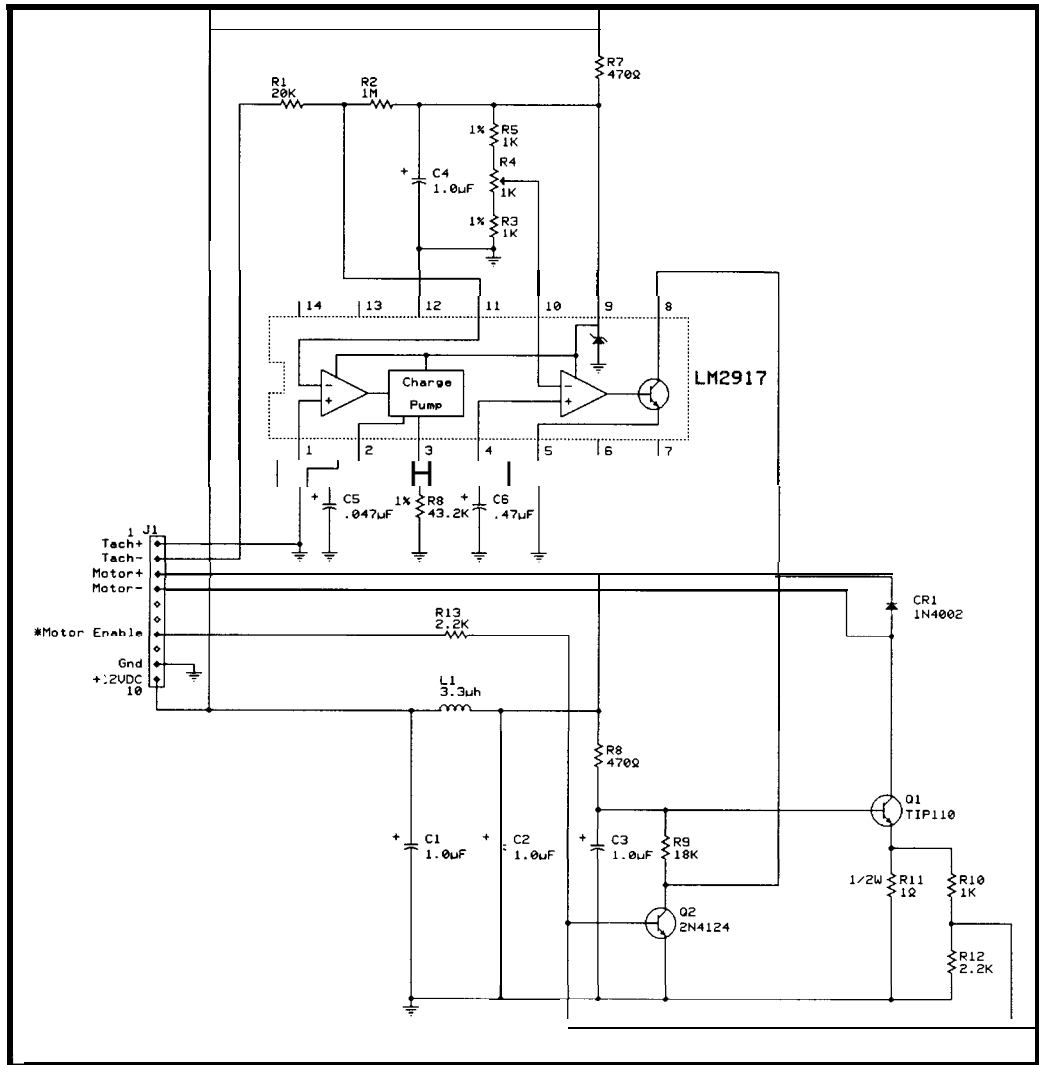


Figure 1—The DC motor control circuitry on an old Tanden disk drive uses an LM2917 F/V converter to automatically keep the motor spinning at the same speed regardless of load.

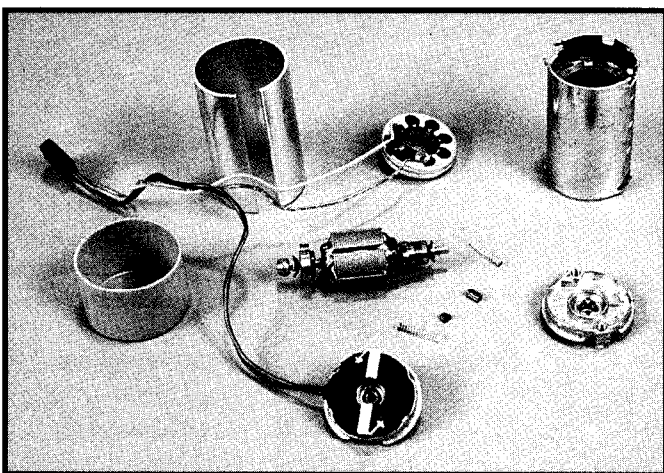


Photo 2—Two wires going to the drive motor in a Tanden drive supply DC power through a set of brushes to the seven rotor windings, and at the opposite end, a magnet spins past a pickup coil to provide a tachometer output.



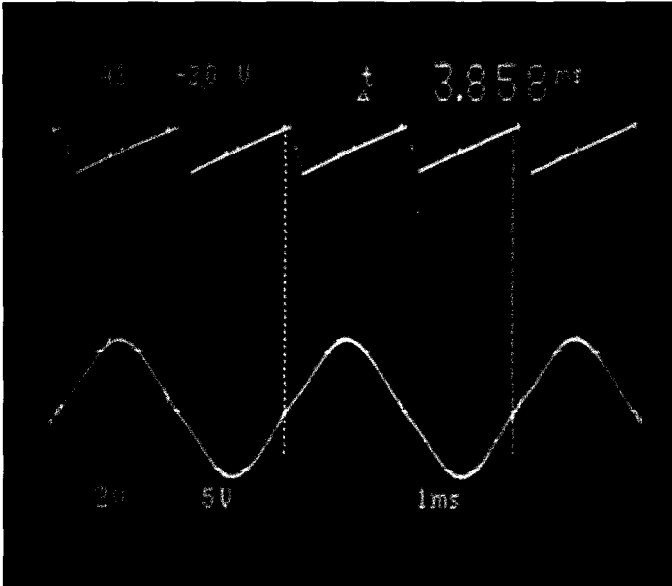


Photo 3—The tachometer output of the Tanden motor (top trace) is used to fine tune the motor voltage (bottom trace), closing the loop on the control circuit.

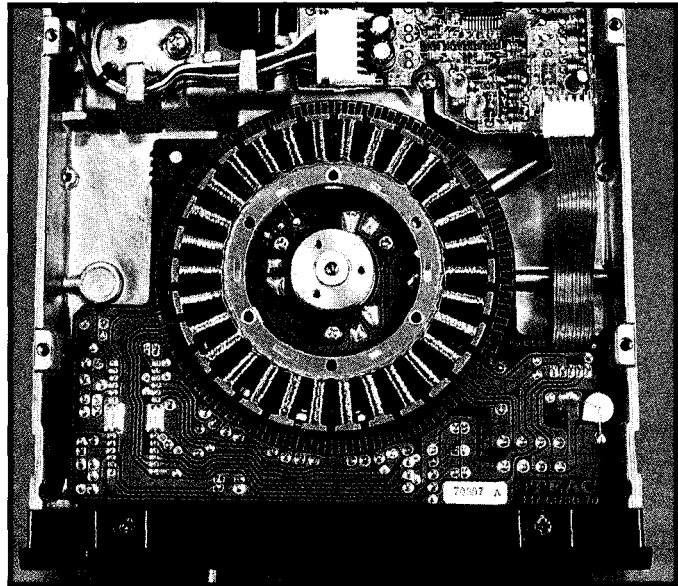


Photo 4—Newer half-height Teac drives replace the old Tanden spindle motor with a pancake motor, reducing the size and eliminating the brushes.

television destined for the junk heap. That started me thinking about recycling methods other than use as boat anchors.

Today's circuitry is of a much higher integration than the discrete circuits of the past. Nowadays you take it all or nothing. Modular designs are out, and the all-in-one designs are in. This means you can't easily save a portion, such as an audio amplifier, without the other stuff attached. If you've ever seen a Heathkit, you know what I mean—each function had its own separate PCB.

What is bigger than a breadbox and as small as a deck of cards?

*Hint: you can get them used for around \$5 at a computer show/flea market? They come in 8-, 5¼-, and 3½-inch formats?*

Yup, ye olde disk drive! These are a virtual storehouse of mechanical and electronic parts. If you've got a bunch hanging around collecting dust, don't deep six 'em—depopulate them. Let's take a closer look at just what you can get.

## DISSECTING THE DISK DRIVE

We kind of take floppy drives for granted

these days. They really do perform a great service for us. This brings to mind the Apple commercial showing a secretary talking to her computer. "Now, I want this back again!" she pleads, as she stuffs the diskette into its maw. Now, I realize a drive can tell when a diskette has been inserted. Spin it at precisely the right RPM to allow a magnetic pickup to search around through rings of magnetic patterns. And, snatch off just the right data, even if it has never seen the disk before. But it hardly has the intelligence to foil its operator.

It's been many years since I had a floppy disk drive fail. It seemed to happen all the time with my TRS-80

Model 1. Of course at that time, I was experimenting with double-sided drives and Percom disk doublers. And even then, failures amounted to nothing more than misalignments.

But, I've got to face facts. These old drives will not be used again. Improved MTBF, reduced current consumption and size, and higher densities combine to make these old dinosaurs obsolete.

Photo 1 illustrates the most obvious difference between the older and newer drive technologies. Older drives use discrete wiring for each subsystem while newer drives incorporate these subsystems into the PCBs. The elimination of wires, connectors, and the labor that goes into manufacturing each subsystem has made the largest contribution to lower drive prices.

The 8" behemoth of floppies is really quite a sturdy piece of equipment. Diskettes are spun by a 120-VAC motor which achieves a spindle speed of 360 RPM. No speed adjustment is used since it is locked to the line frequency. When a diskette is inserted into the

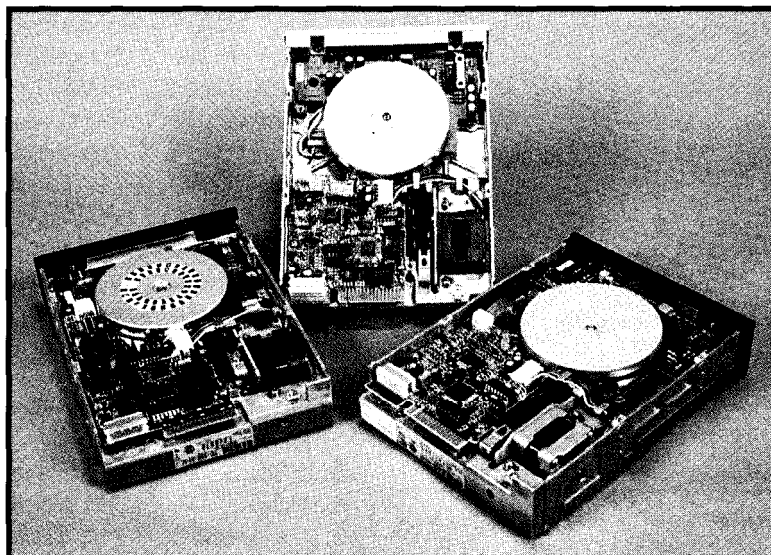


Photo 5—Within the TeacFD55 line, the electronics have steadily shrunk over the years.

drive, it cocks a release spring which kicks the diskette out when the drive door is reopened. The drive's diskette door is held in a locked position (indicated by an LED on the front panel) by a solenoid whenever a read/write function is in operation. Three optical sensors report diskette index position, sector position, and write protect status to the drive's circuitry. (Sector positioning was fixed by the sensors and not by timing from the index mark as in 5" and 3" drives.)

A stepper motor moves the magnetic heads along a twin-beam carriage centered on the diskette's centerline. The AC motor spins the diskette constantly, so a separate head load solenoid lowers and raises the magnetic heads. (This reduced head wear since, like a tape recorder, the magnetic heads were in contact with the media while in use.) Last, a giant PCB containing electronics to integrate all the subsystems.

The 5¼" drive uses many of the same ideas as its bigger brother. It has optical sensors, mechanical switches, and a head stepper motor. It doesn't have a sector sensor since sector spacing is done by software. Although some (older) 5" drives have head load solenoids, ingenious mechanics handle this task in the newer drives.

For the 5¼" and 3½" drives, the diskettes do not rotate continuously, so the head can be loaded whenever the drive door is closed without worry about needless head wear. Although you are allowed to physically pop out the diskette while in operation (there is no solenoid lock), damage to the data will likely occur.

Because the power for these smaller drives is DC, some kind of motor-speed control must be used. Closed-loop feedback of some kind is generally employed to ensure automatic and consistent control.

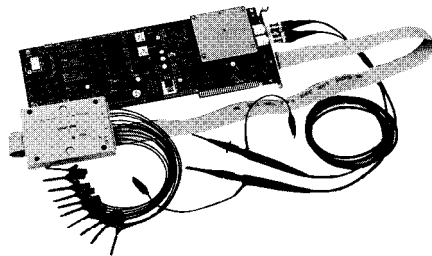
## STATS

In case you didn't disassemble your drives along with me, let's look at what I was able to recycle (Table 1).

Old Tandon drives are my favorite. The DC motor-control circuitry is contained on its own PCB mounted on the rear of the drive. Photo 2 offers an

## PC-Based Instruments 200 MSa/s DIGITAL OSCILLOSCOPE

**HUGE BUFFER  
FAST SAMPLING  
SCOPE AND LOGIC ANALYZER  
C LIBRARY W/SOURCE AVAILABLE  
POWERFUL FRONT PANEL SOFTWARE**



**\$1799 - DSO-28204 (4K)  
\$2285 - DSO-28264 (64K)**

### DSO Channels

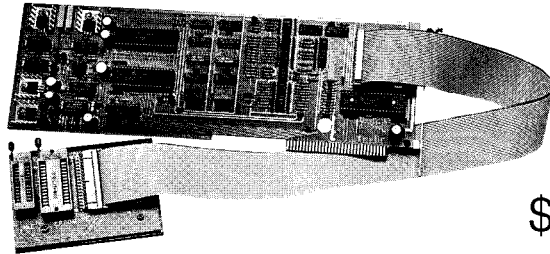
2 Ch. up to 100 MSa/s  
or  
1 Ch. at 200 MSa/s  
4K or 64K Samples/Ch  
Cross Trigger with LA  
125 MHz Bandwidth

### Logic Analyzer Channels

8 Ch. up to 100 MHz  
4K or 64K Samples/Ch  
Cross Trigger with DSO

## Universal Device Programmer

PAL  
GAL  
EPROM  
EEPROM  
FLASH  
MICRO  
PIC  
etc..

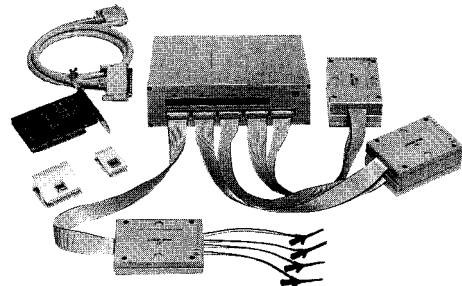


**\$475**

Free software updates on BBS  
Powerful menu driven software

## 400 MHz Logic Analyzer

up to 128 Channels  
up to 400 MHz  
up to 16K Samples/Channel  
Variable Threshold Levels  
8 External Clocks  
16 Level Triggering  
Pattern Generator Option



**\$799 - LA12100 (100 MHz, 24 Ch)  
\$1299 - LA32200 (200 MHz, 32 Ch)  
\$1899 - LA32400 (400 MHz, 32 Ch)  
\$2750 - LA64400 (400 MHz, 64 Ch)**

**Call (201) 808-8990**



**Link Instruments**

369 Passaic Ave, Suite 100, Fairfield, NJ 07004 fax: 808-8786

exploded view of the spindle motor. Two wires supply the DC power through a set of brushes to the seven rotor windings. At the opposite end of the shaft, a magnet spins within a pickup coil providing a tachometer output through a second set of wires. The AC signal, generated by the spinning of the motor's shaft, is fed to an LM2917 frequency-to-voltage converter (see Figure 1).

The proportional output voltage is compared to a second voltage, the speed control's adjustment potentiometer. If the adjustment voltage is higher than the F/V input, motor current is allowed to increase. For example, when a diskette is inserted, the motor load is increased and causes a reduced frequency output in the tachometer. The reduced proportional voltage causes an increase in motor current to counteract the added load. An equilibrium between the current and load is established based on the nominal 300 RPM set speed. The scope trace pictured in Photo 3 shows tachometer output and the controlled motor voltage.

Newer Teac drives use brushless DC pancake motors. The two-phase, bipolar DC drive signals energize stator poles in succession, like a stepper motor, thereby simulating the function of brushes without causing mechanical wear. The pancake motor's rotor is on the outside of the stator's poles. Its thin cylindrical magnet, built into the rim of the motor, also gives the light rotor a bit of inertia by adding weight to the rotor's edge. See Photo 4 for a view under the rotor.

Tiny, Hall-effect sensors mounted between the poles energize the stator coils in the proper order to ensure rotational direction. Unlike a driver for a stepper motor, there is no external step and direction control; it is a closed-loop system. Feedback on Teac's motor starts with the rotor's ring magnet serving a second function: it rotates over a zigzag tachometer pattern etched into the PCB. This pattern picks up the passing magnetic field and outputs pulses proportional to the speed. A F/V converter closes the loop and, like the Tandon, adjusts the motor's current to keep the speed

### 8" Shugart

1 SPDT microswitch	door closed sensor
2 IR LEDs	index and sector sensors
2 phototransistors	index and sector sensors
1 LED	in-use indicator
1 solenoid	door lock
1 solenoid	head load
1 optointerrupter	diskette sensor
1 optointerrupter	track 0 sensor
1 stepper motor	head movement
1 120-VAC motor	diskette rotation
1 PCB	discrete logic controls all functions

### 5" Tandon

1 SPDT microswitch	write-protect sensor
1 visible LED	in-use indicator
1 SPDT microswitch	track 0 sensor
1 IR LED/phototransistor pair	index-hole sensor
1 12-VDC stepper motor	read/write head movement
1 12-VDC brush motor with tach winding	diskette rotation
1 DC motor servo board	
1 PCB with stepper driver and read/write circuitry	

### 5" Teac

1 visible LED	in-use indicator
2 phototransistors	write-protect and index sensors
2 IR LEDs	write-protect and index sensors
1 12-VDC 2-phase bipolar brushless motor with tach pickup	diskette rotation
1 DC motor servo board	
1 12-VDC stepper motor	read/write head movement
1 PCB with stepper driver and read/write circuitry	
1 optointerrupter track zero sensor	
1 12-VDC solenoid head loading	

Table I—Dead disk drives are a virtual storehouse of useful raw parts for future projects.

correct. Early Teacs used separate Mitsubishi F/V and motor controller ICs. However, these functions are now integrated into a single chip.

Setting an accurate 300 RPM is easily accomplished thanks to 50-60 Hz lighting and a stroboscopic label stuck onto the spindle rotor. If the label rotates too fast, the spokes on the label seem to move forward because they are not in sync with the 50-60 Hz pulsations of the overhead lighting. Too slow, and the pattern rotates in reverse. Adjust the speed control to keep the pattern still and voilà, you're done.

You can see why I like the Tandon parts better than the Teac. Not only

are Teac's pancake motors larger, but the drive PCB is integrated into its motor design, increasing its physical girth even more.

## STEP RIGHT UP

Not much has changed in the technique of track stepping. The number of tracks has increased from 48 to 96 tracks per inch, but the song remains the same. And the split band movement has remained the most widely used technique. This is similar to the way the station indicator worked on (nondigital) AM/FM receivers. A taut band around a stepper motor's shaft moves the head by a pull-pull motion across the surface of a

Listing 1—It's possible to experiment with controlling a salvaged stepper motor using a PC's printer port and a few lines of code.

```

10 X=0
20 A=&H378 :REM Set to addr of your port (&H278, &H378, or &H3BC)
30 PRINT "Hit 'S' to step, 'D' to change direction"
40 I$=INKEY$: IF I$="" THEN GOTO 40
50 IF (I$="S" OR I$="s") THEN PRINT"Stepping once..." :
    OUT A,(X+1): OUT A,X: OUT A,X+1
60 IF (I$="D" OR I$="d") THEN PRINT"Changing direction..." :
    X=(X XOR 2)
70 GOTO 40

```

diskette. Other methods, like rotating cams or feedscrews, are also used. The same parts are used on both 48- and 96-TPI drives, except for the stepper motor's step angle and the magnetic head's track width, both of which are reduced to increase track densities.

The stepper motors may be driven from the associated PCB using the direction and step inputs located on the disk drive connector. You can connect your PC's printer port to the disk drive's edge connector and step the motor with simple print statements to toggle the Step and Direction inputs. Make a simple three-wire cable using the connections shown in Figure 2 and the code in Listing 1.

The stepper and read/write circuitry is heavily intertwined on these PCBs. You may wish to refer to past issues of the *Computer Applications Journal* or other fine magazines for articles on steppers and stepper circuits, or you may even wish to design your own interface.

Notice in Photo 5 that the stepper drive and the magnetic read/write circuitry takes on a whole new level of integration with each model and revision. Eight-inch drives used total

discrete logic for every function. Five-inch drives, although born with discrete logic, quickly developed larger forms of integration to lower manufacturing costs and improve MTBF stats. Ultimately, this gives the user a superior product at a lower cost.

## TWO HEADS ARE BETTER THAN ONE

I'll save the magnetic read/write heads and associated circuitry for now, but I suspect I won't create a project that uses them—unless I get itchy for a magnetic card reader, a cassette tape backup unit, or a recovery probe for CRC errors. Hmm...that last one might need to wait until April.

In the meantime, please your significant other and throw out some trash. Just make sure you don't scrap the good stuff, and remember to pick that carcass clean. ☐

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

**includes product design and manufacturing. He may be reached at** [jeff.bachiochi@circellar.com](mailto:jeff.bachiochi@circellar.com).

## REFERENCES

Maintenance manuals were used from the following companies:

Teac America  
(213) 726-0303

Amdek Corp.  
(312) 364-1180

Siemens Energy and Automation  
(404) 740-3000

BASF Corp.  
(201) 397-2700

Tandon Corp.  
(805) 523-0340

μ-sci  
(no phone number available)

## IRS

416 Very Useful

417 Moderately Useful

418 Not Useful

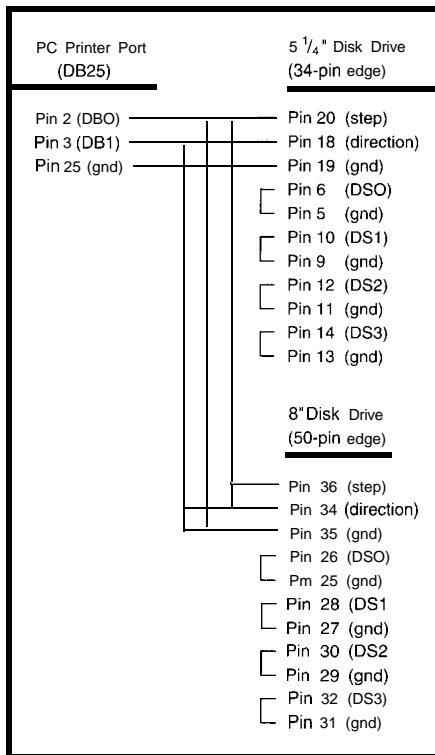


Figure 2—Three wires are all that's needed between a PC's printer port and a junk disk drive to move the drive's head back and forth.

# The Real Logic Analyzer



Test your Logic circuits with the printer port of your IBM or compatible computer!

- El 5 Input capture channels via printer port
- High Speed 64K input capture buffer
- Glitch capture and display
- Full triggering on any input pattern
- Automatic time base calibration
- El 4 cursors measure time and frequency
- Save, print or export waveforms (PCX)

The Real Logic Analyzer is a software package that converts an IBM or compatible computer into a fully functional logic analyzer. Up to 5 waveforms can be monitored through the standard PC parallel printer port. The user connects a circuit to the port by making a simple cable or by using our optional cable with universal test clips. The software can capture 64K samples of data at speeds of up to 1.2uS (Depending on computer). The waveforms are displayed graphically and can be viewed at several zoom levels. The triggering may be set to any combination of high, low or Don't Care values and allows for adjustable pre and post trigger viewing. An automatic calibration routine assures accurate time and frequency measurements using 4 independent cursors. A continuous display mode along with our high speed graphics drivers, provide for an "Oscilloscope-type" of real time display. An optional Buffer which plugs directly to the printer port is available for monitoring high voltage signals.

**LOGIXELL**  
ELECTRONICS

61 Piper Cr.  
Kanata, Ontario  
Canada K2K 2S9

Requires 266, or higher with EGA or VGA display.

LA20 Software Only \$79.95us  
Software With Test Cable \$99.95us  
BUFF05 Buffer \$39.95us

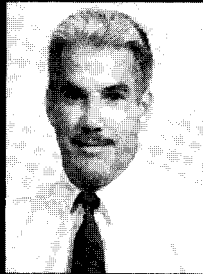
Tel: (613)599-7088  
Fax: (613)599-7089



# SILICON UPDATE

Tom Cantrell

## Flash of Inspiration



The latest craze in nonvolatile storage is flash

memory, but adding it to a system using PCMCIA cards has its shortcomings. Tom checks out a unique solution that requires virtually no extra hardware or software.



much (perhaps too much) has been written about the imminent demise of disk drives at the hands of flash memory. Yes, in principle, it seems reasonable that chips of sand will inevitably surpass the mechanical complex—motors, platters, actuators, and so on—that make up a disk drive.

Unfortunately, for the flash marketeers, there are a couple of problems with their rosy scenarios.

First is the simple matter of price. The fact is flash disks still cost nearly 100 times their spinning counterparts (e.g., 50¢ vs. \$50 per MB). Continuing software bloat and evermore porcine programs magnify the disk drive's cost advantage. Furthermore, both camps' price trends exhibit similar slopes (down, of course), pushing any cross-over far into the future.

The second problem isn't really flash's fault, but rather the vehicle that's been chosen to deliver it. I'm referring, of course, to PCMCIA which might better be called the "bus that never stops-changing, that is." What began as a simple memory-card concept has grown willy nilly, and what-with the latest attempts to make it a master bus (with the PCI folks getting their two cents in)-has become an unwieldy mess.

I'm reminded of the Winchester Mystery House, a local attraction named after a turn-of-the-century eccentric who believed she wouldn't die as long as she kept adding on to the house. A song describing Ms. Winchester's pad-and PCMCIA at

this point might aptly be titled "Stairway To Nowhere."

Enter EUROM, a company that brings a healthy dose of reality to the flash market by adding a dab of ASIC and a dash of hybrid packaging. The result is chips that are both easy and sensible to use.

### DISK0 FEVER

Consider a PC that only needs a small amount of flash disk, say 1-2 MB. Or, how about an embedded PC that calls for minimal chip count, power consumption,

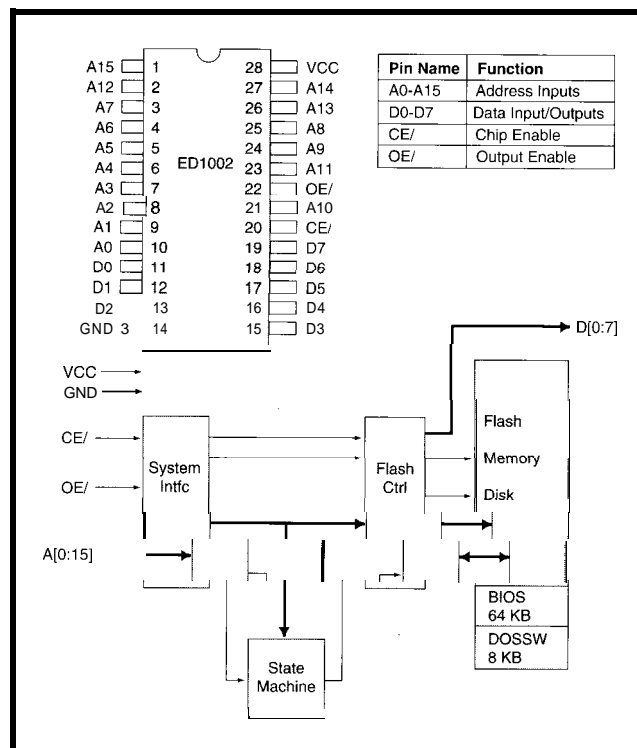


Figure 1—The DiskOnChip hybrid consists of an ASIC that performs housekeeping tasks and flash memory that holds the DiskOnChip software, BIOS, and flash disk.

size, and cost? In either case, a PCMCIA flash-disk approach proves problematic. Cost, form factor, and packaging issues—not to mention the need for a confusing variety of drivers and card (dis)services—make it a nonstarter.

Instead, consider EUROM's ED1002 DiskOnChip (Photo 1), a device that exploits hybrid surface-mount technology (Photo 2) and combines a control ASIC and flash memory chip into a single BIOS-ROM-compatible chip.

As shown in Figure 1, the ASIC performs myriad housekeeping tasks while the flash holds the DiskOnChip software, BIOS, and flash disk. Versions with 1, 2, and 4 MB of flash are available in 1000s at \$87, **\$157**, and \$257, respectively.

On the surface, the concept is blessedly simple. Just replace your BIOS ROM with the equivalent pinout DiskOnChip and voilà, instant flash disk. No need for special drivers and so on. The flash disk operation is managed transparently and is a simple INT13 call away, just like any other disk. Importantly, for embedded applications, it's even bootable, which means that otherwise diskless (floppy and hard) applications are supported.

It sounds simple, but under the hood, a lot of tricky stuff is going on. Right off the bat, you may notice the chip (like a BIOS ROM) doesn't have a WE' (write enable) pin. Obviously, since the world doesn't need a read-only disk, some cleverness is taking place.

Following the system's operation from power on will give a better idea of what's happening. At first, the POST routine is executed as always. Once it passes, the DiskOnChip starts into action. (Note that this is before the BIOS bootstrap loader kicks in, which is why the chip is bootable.)

The first step is to copy 8 KB of DiskOnChip software (also known as DOCSW) into the top of conventional memory. Yes, it cuts into available application space, but 8 KB seems a small price to pay since the DOCSW performs a number of critical tasks.

First of all, DOCSW redirects INT13 calls. If the call is deemed for a

spinning disk, it's passed on to the regular BIOS INT13. Otherwise, the DOCSW DFS (DOC File System) goes to work, handling the mapping of sectors into flash addresses, and reading or (magically) writing the data. Though the details aren't public, DFS presumably does try to minimize the flash write-endurance problem (100k cycles) using some or all of the popular erase minimization, wear leveling, and bad sector remapping techniques.

Since the BIOS contains the interrupt handler, one gotcha associated with combining flash disk and BIOS in the same chip is interrupts. What happens if an interrupt comes in during a relatively lengthy flash write (9  $\mu$ s) or erase (300 ms) cycle? The answer is that DOCSW includes an interrupt-handler front end that decides how best to deal with the situation. An interrupt during a write will be held off until the write completes. However, an interrupt during an erase has priority; the erase opera-

tion is suspended and does not resume until the interrupt is serviced.

A similar trick handles reset during a write/erase. In this case, the DiskOnChip force-feeds the CPU a branch loop until the flash operation completes and the normal startup sequence begins again.

Despite EUROM's best efforts, the DiskOnChip does impose a few restrictions on system design. First of

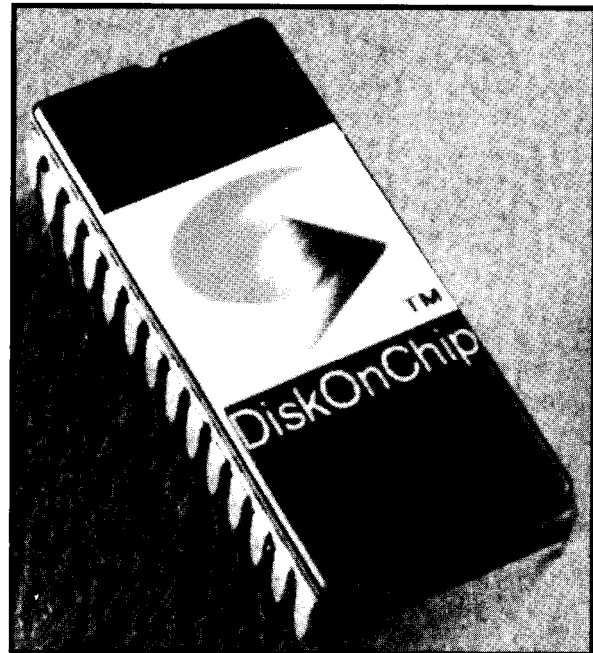


Photo 1—The DiskOnChip is a pin-compatible, drop-in replacement for many PC BIOSs that adds flash disk support with no extra hardware or software.

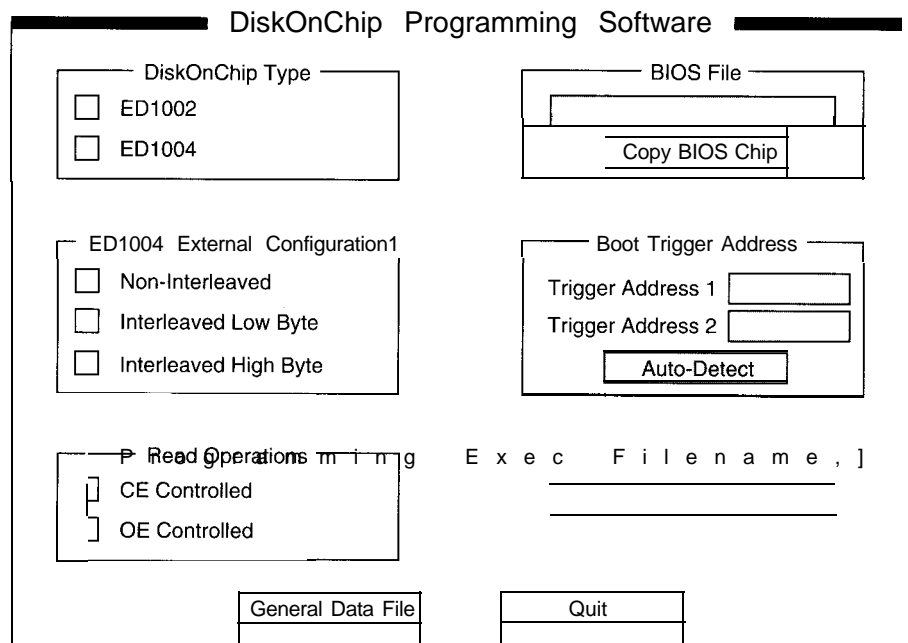


Figure 2—The DiskOnChip configuration software specifies hardware configuration, BIOS file, and boot trigger addresses.

all, the ED1002 only works with a 64-KB BIOS and is incompatible with systems that use a 128-KB BIOS (or two 32-KB BIOS chips).

Furthermore, the widely used "Shadow-RAM" approach, in which the BIOS is copied from ROM into RAM, is a no-no. Obviously, if the CPU is talking to the BIOS in RAM, the DiskOnChip is left out in the cold.

Fortunately, I believe the ShadowRAM is a CMOS-setup option in most systems.

EUROM does offer another version, the ED1004, that can address the 128-KB, two-chip, and Shadow-RAM issues. It also features a WE\* input for direct writing. However, it's packaged in an 84-pin PLCC and calls for a special PCB and more expensive socket.

Setting up a DiskOnChip is a three-step process. The first step is handled by a EUROM supplied program (see DOCMENU in Figure 2) which specifies the hardware configuration (chip and control line), BIOS file, and boot trigger addresses (used by the DiskOnChip to detect the occurrence of INT19/[boot DOS]). The output of this step is saved to a file.

Next, the file saved in step 1 is programmed under the control of a second EUROM supplied program, DOC P R. The resulting chip is ready to be plugged into a target system.

Finally, the flash disk must be initialized. Highlighting the transparency of the DiskOn-Chip approach, our old friends F D I S K and FORMAT do the trick. Similarly, getting files on/off the flash disk is as easy as COPY.

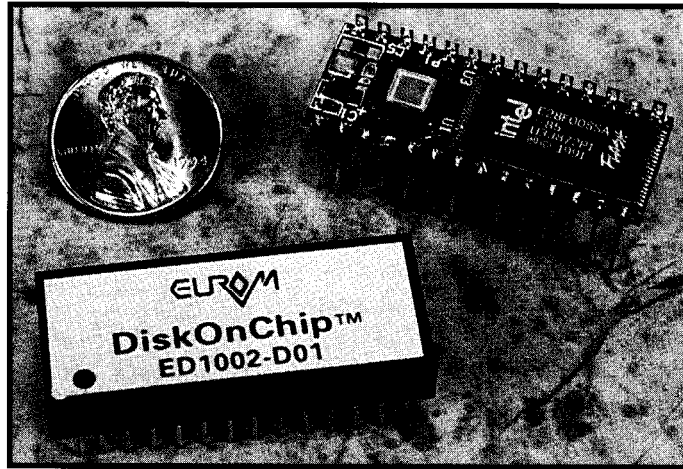


Photo 2—The DiskOnChip uses hybrid surface-mount technology to combine off-the-shelf parts into a unique module.

### TALE OF TWO PINS

Like the DiskOnChip, the ES 10 16 Smart Flash ROM (SFROM) combines the ASIC and flash chips in a single module. Featuring 150-ns access time, the SFROM is available in versions from 256 Kb to 8 Mb. The price for the lower-density versions is a little steep, dominated by the multichip assembly cost. For example (in 1000s), the 256-Kb chip is \$3 1, while the 1-Mb version is only \$32. Furthermore, the 8-Mb

chip is only \$61, less than twice the 1-Mb part, so go ahead and splurge.

The ES1016 pinout (Figure 3) sure looks familiar. In fact, the lower (physically) 32 pins are the standard byte-wide JEDEC pinout we know and love. It's the top two pins (SDIN and SDOUT) that make the ES1016 unique.

Indeed, ignoring the extra pin pair, at power up the ES1016 defaults to normal mode in which it operates exactly like standard flash. Reads

proceed in the usual way, under control of CE\* and OE . .

Writes take place via WE\* and a command register. First, a program command (a special sequence of bytes) is written to "unlock" the chip, and then each byte is written in a loop (Figure 4). The byte-loop timing is handled by the data-polling technique. After each byte is written, subsequent reads return the complement of D7 until the write completes. When D7 matches what was written, the loop proceeds to the next byte.

It's when a wakeup sequence is detected on SDIN that interesting stuff starts to happen. The ES1016 switches to serial mode, and control of the flash switches to the on-chip ASIC and the JEDEC pins are ignored. Note the potential system design hazard—what a CPU connected to the JEDEC pins sees when the ES1016 is in serial mode isn't defined. You need to make sure the CPU is held in reset or DREQ (DMA request) or is executing out of another memory chip lest it get lost in the woods.

The SDIN and SDOUT pins implement a standard UART (8N1) and automatically detect

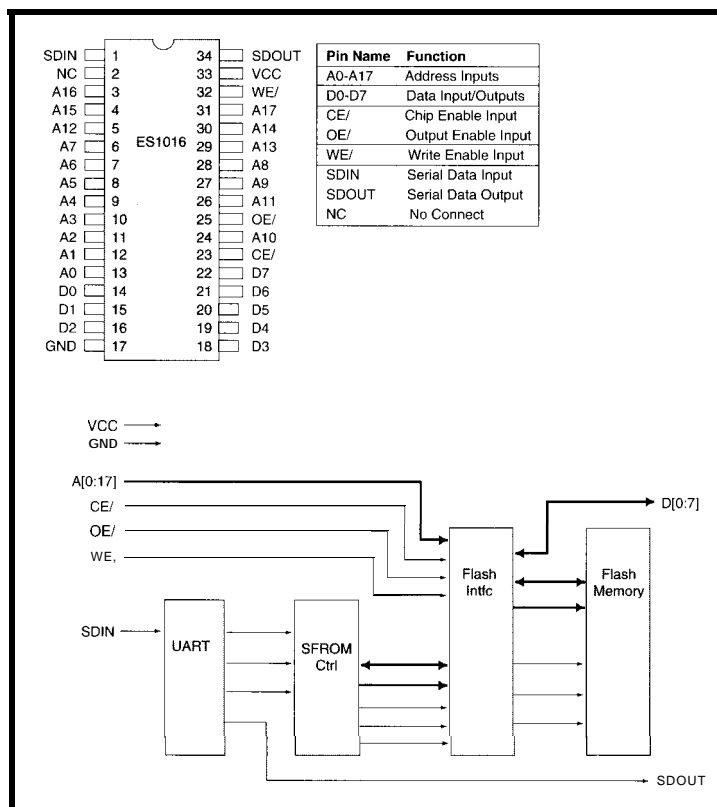


Figure 3—The ES1016 Smart Flash ROM (SFROM) combines the ASIC and flash chips in a single module and can be programmed either byte-wide or serially.

the baud rate (up to 75 kbps). Notably, they are daisy chainable, so you can string together an arbitrarily long chain of SFRoMs (up to 256). Each chip's SDOUT is connected to the next one's SDIN with the first SDIN and last SDOUT connected to the host.

Of course, the SDIN and SDOUT pins are TTL, not RS-232, compatible. If you want to connect to a PC serial port, level shifters are required. On the other hand, TTL compatibility can work in your favor by switching the connection to the PC parallel port, using the time-honored hack of coercing the Strobe and Busy lines into acting as a soft UART.

EUROM supplies software (SPS) that handles just such a connection. It automatically detects the number of SFRoMs in a chain and allows data to be uploaded or downloaded to any selected device. SPS also includes a simple hex-file editor and can convert between Intel hex, Motorola S records, and binary formats.

The serial-mode command set is shown in Figure 5 and can be roughly divided into initialization, register access, and data transfer commands.

The Wakeup, PowerDown, and Enable/DisableFlash commands are used as transitions between normal and serial modes. In this context, the PowerDown and Enable/DisableFlash commands really refer to the serial control logic and not to the entire chip. In other words, PowerDown shuts off the UART and Enable/Disable switch between serial and normal (JEDEC) mode.

The power-up algorithm is important since it must handle the challenge of initializing the daisy chain (Figure 6). WakeUp1 gets the first chip out of bed, and it is assigned an ID using the WriteChipNumber1 command. Next, the WakeUp2 and WriteChipNumber2 commands are used to set up subsequent chips. Note that the 1 and 2 variants of commands

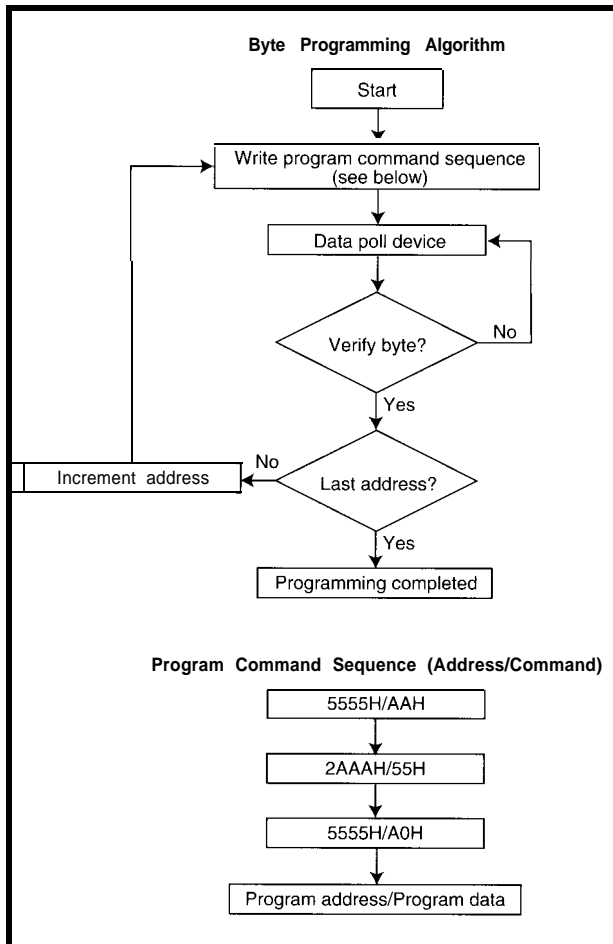


Figure 4-The ES1016 byte-wide programming algorithm is similar to that used when programming an EPROM, though requires extra steps to send the chip commands

all deal with the difference between a first (or last) chip access and a selected chip access. Most non-1 or -2 commands are global, affecting all devices in the chain.

The benefits of the SFRoM may not be obvious, but could prove quite helpful under certain design scenarios. Specifying a 34-pin (vs. 32-pin) socket and 3-pin (SDIN, SDOUT, and GND) header may be wise and certainly doesn't cost a lot.

Most obvious, the SFRoM provides a serial programming port in a system which is short a UART. In particular, for a minimal chip count, space-constrained design, eliminating a single extra chip could make a 20-33% chip count per area advantage.

However, more subtle SFRoM advantages may actually have more impact. For instance, consider a hypothetical application that needs a serial flash disk. Designer Joe Goofus, having fallen prey to the PCMCIA

hype, might build a complicated gizmo with a single-chip CPU + UART and PCMCIA interface chip along with its expensive connector and open-to-the-elements slot. Jane Gallant, on the other hand, builds a nice simple hermetic box with a string of SFRoMs and.. nothing else!

Don't forget software. Or better yet, do forget it. Joe will be spending nights in the lab tweaking code to match the vagaries of low-level flash programming while Jane—thanks to SPS and on-chip logic-can relax.

In previous articles, I've mentioned the production advantages of in-system programming (i.e., speed production flow by eliminating the programming step). The idea is to simply take chips from tube to PCB without extra handling, loading the latest and greatest code release right before the system goes into the box.

Regular flash chips, however, may encounter the bootstrap problem (i.e., how can the system CPU program the self-same flash chips from which it is executing?) The SFRoM's serial port provides a clean and easy solution for production-line programming.

The same idea-separating the program from the CPU access port—may also be of interest to high-security types who don't trust a password as far as they can throw it. A system that is self-programmable tends to be trashable, hackable, and vulnerable (i.e., a recipe for disaster). The spy-versus-spy solution is to leave the WE\* disconnected and to physically secure the board and SFRoM header with a padlock, big dog, or M16-toting GI.

## CONVERSATIONAL COMBO CHIP

Last year I wrote about the voice record/playback chips from ISD ("Talking Chips," CAJ 36). Since then, you may have noticed some impressive design-ins for ISD, including those "Say 'hi' to Grandma" vocal greeting



Command	Sequence	Comments
Wake Up 1	D9	
Wake Up 2	25 03 70 D9	
Power Down	04 00 23	
Enable Flash Array	0D	
Disable Flash Array	0C	
Clear Status Register	07 00	
Read Status Register 1	25 03 70 07 06	
Read Status Register 2	06	
Clear Checksum Register	01 00	
Read Checksum Register 1	25 03 70 01 00	
Read Checksum Register 2	00	
Write PageLength Register	15 xx	Write xx to PageLength Reg
Read PageLength Register 1	25 03 70 15 14	
Read PageLength Register 2	14	
Write Segment Register	25 01 71 xx	Write xx to Segment Reg
Read Segment Register 1	25 03 70 11 10	
Read Segment Register 2	10	
Write Offset Register	13 xx	Write xx to Offset Reg
Read Offset Register 1	25 03 70 13 12	
Read Offset Register 2	12	
Write ChipNumber Register 1	03 01	Write xx to ChipNumber Reg
Write ChipNumber Register 2	25 03 70 03 25 03 70 xx	
Select Device	04 00 04 xx	Select Device xx
Page Read 1	0F 00 25 03 70 15 14 25 03 70 17 1A	
Page Read 2	0F 00 1A	
Page Load	17 xx1...xxN	Write xx to SFROM control data buffer
Page Write	0F 01 1B	
Page Verify	0F 00 1D	
Erase Chip	25 55 55 AA 25 AA 2A 55 25 55 55 80 25 55 55 AA 25 AA 2A 55 25 55 55 10	

Figure 5-h serial mode, the ES1016 supports a whole host of commands, which can be categorized as initialization, register access, and data transfer.

cards, the digital answering machine in Motorola cellular phones, not to mention a variety of chatty toys. The ISD chips are talking all right-and the message is "call your broker."

Needless to say, ISD's success is attracting a lot of attention. EUROM's response is a forthcoming (projected for first quarter '95) chip, called the ChipRecorder. Unlike the DiskOnChip and SFROM, the DiskRecorder is a standard monolithic IC designed to work with external flash chips.

As shown in Figure 7, the ChipRecorder includes a plethora of functions. Overall operation is controlled by the EUP (EUROM Proprietary Processor), a small RISC fed by on-chip ROM (2 KB x 8) and RAM (256 bytes). Serial access is provided via both UART and I<sup>2</sup>C ports. Unique interfaces include 4 x 4 matrix-

encoded keyboard and 4-digit (7-segment) LCD driver. The ears and vocal cords are handled by an 8-bit ADC and DAC, conditioned with a programmable-gain op-amp.

The ISD chip's strength is (thanks to their DAST [Direct Analog Storage Technology] scheme) focused on the low end. There is simply no cheaper way to provide minimal voice I/O, which explains their success in

greeting card and toy applications.

However, the other side of the coin is that the ISD chips are intrinsically limited by the on-chip memory capacity. Their first devices held 20 s or so, with a few minutes foreseeable down the road.

The DiskRecorder can't compete at the low end, but by working with off-chip flash, offers high capacity (EUROM forecasts about 20 minutes per MB).

One advantage of the digital (versus DAST) approach is that we can haul out a variety of digital-data compression techniques. Indeed, by my figuring, the 20-minutes-per-meg spec implies the EUP is delivering about a 10: 1 compression ratio, which completely offsets ISD's 8: 1 analog-storage-density advantage (to be fair, respective sound quality also needs to be compared). Perhaps the ISD approach is also amenable to compression, but I suspect it's a lot more tricky and may not pay back.

While initially targeted at the voice I/O arena, remember that the function of the EUROM chip is dictated by the on-chip RISC, ROM, or RAM. It seems to me that such a combo chip could be programmed for a wide variety of useful data acquisition tasks, offloading an otherwise fatigued CPU. The UART, I<sup>2</sup>C, keyboard, and LCD interfaces might come in handy as well. Such a nonvocal role is beyond the ISD counterpart's capabilities, largely due to the fact that the data is never digital, but analog in and out.

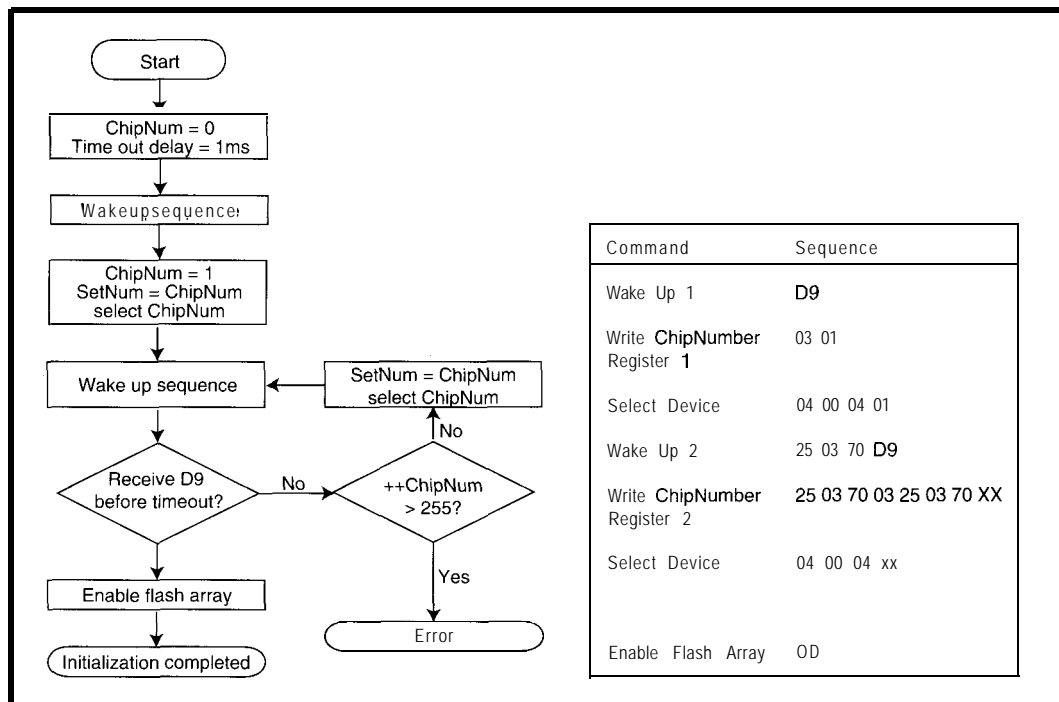


Figure 6—Multiple ES1016s may be daisy-chained for more storage, so the powerup algorithm must initialize all the chips in the chain

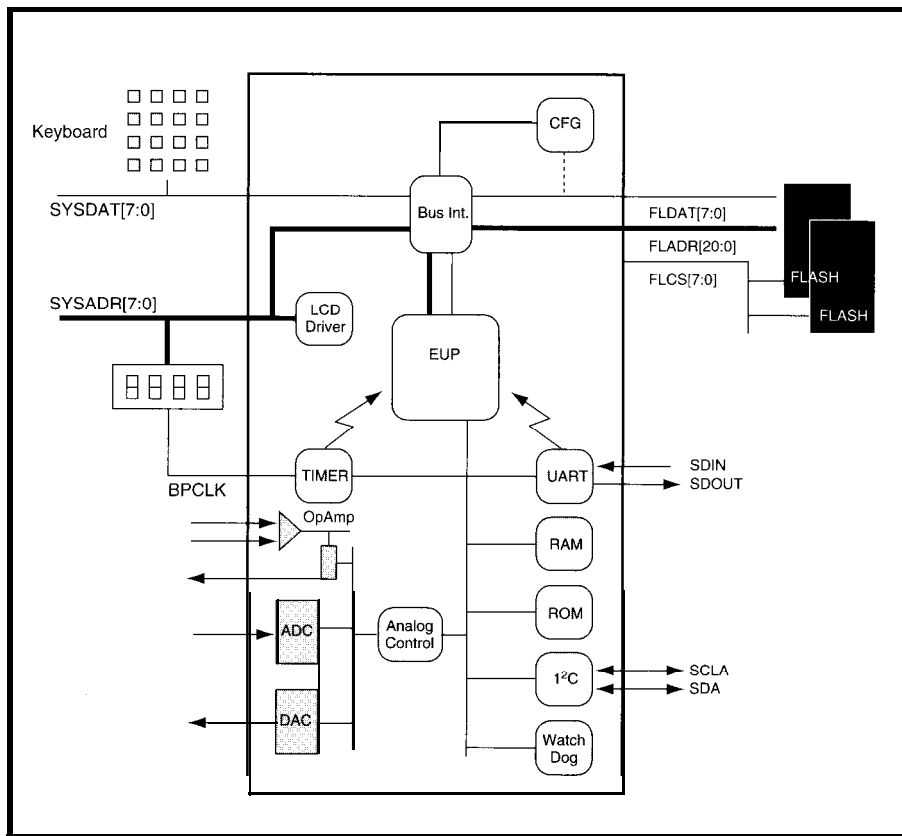


Figure 7—TheChipRecorder uses external flash memory to store recorded audio and can directly interface to a display and keypad.

While cashing in on the voice I/O frenzy, I propose EUROM consider a version of the DiskRecorder with flash or RAM replacing the on-chip ROM. No doubt, there are quite a few designers—you know who you are—who could stuff some clever code in there. □

*Tom Cantrell has been an engineer in Silicon Valley for more than ten years working on chip, board, and systems design and marketing. He can be reached at (510) 657-0264 or by fax at (510) 657-5441.*

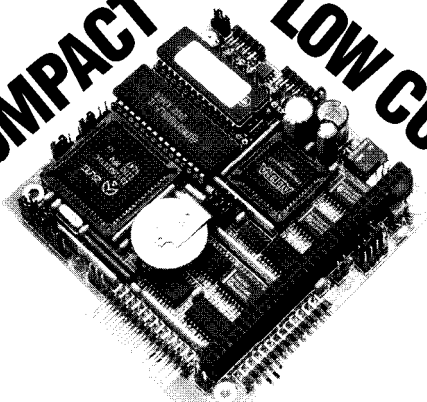
### CONTACT

EUROM Flashware Solutions, Inc.  
4655 Old Ironsides Dr., Ste. 200  
Santa Clara, CA 95054  
(408) 748-9995  
Fax: (408) 748-8408

### IRS

419 Very Useful  
420 Moderately Useful  
421 Not Useful

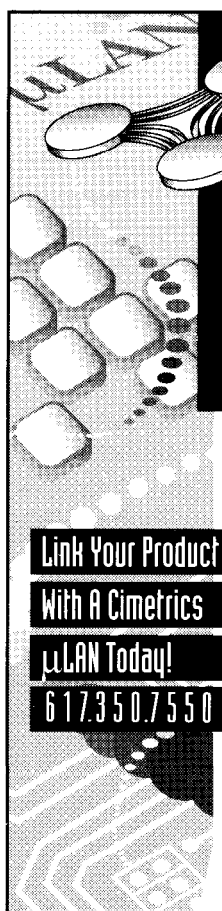
**COMPACT** **LOW COST**



## C-Programmable Controllers

Use our controller as the brains of your next control, test or data acquisition project. From \$149 qty one. Features I/O to 400 lines, ADC, DAC, RS232/RS485, printer port, battery-backed clock and RAM, keypads, LCDs, enclosures and more! Our simple, yet powerful, Dynamic C™ makes programming a snap!

24-Hr AutoFAX: 1724 Picasso  
916.753.0618. Davis, CA 95616  
Call from **your** FAX. 916.757.3737  
Request catalog 18. 916.753.5141 FAX



**Cimetrics**  
TECHNOLOGY

*Linking Microcontrollers.  
Breaking Boundaries.*

### The 9-Bit Solution

The Cimetrics Technology 9-Bit Solution is a complete microcontroller network (µLAN) that supports the 8051, 68HC11, 80C186B/E/C, and many other popular processors. The 9-Bit Solution takes full advantage of microprocessor modes built in to microcontrollers. The 9-Bit Solution allows simple and inexpensive development of master/slave multidrop embedded controller networks.

**Link Your Product  
With A Cimetrics  
µLAN Today!  
617.350.7550**

- 8051, 68HC11, 80C186B/E/C compatible
- A full range of other processors supported
- Up to 250 nodes
- 16 Bit CRC error checking with sequence numbers
- Complete source code included

55 Temple Place • Boston, MA 02111-1300  
Ph 617.350.7550 • Fx 617.350.7552

# EMBEDDED TECHNIQUES

John Dybowski

## ec.32 Wrap Up

**O**ne elusive issue of processor throughput involves many subtleties. Some

processor architectures are simply better at performing certain operations than others. On the other hand, certain applications require specialized CPUs.

If processing needs are really stringent, you've no choice but to find just the right processor for the job. However, most applications aren't so restrictive and favor a general-purpose processing engine as a natural choice. Nonrestrictive applications change everything and may make sticking with a familiar architecture the dominant concern.

Here possession is nine-tenths of the call. If you already possess the knowledge base, own the development tools, and have access to a significant body of working software, it's very difficult to justify a move to a superior processor architecture.

But, what happens when you finally exceed the performance capabilities of your CPU? This can prove to be disturbing, especially if you're faced with adopting something totally different than you're accustomed to. Switching CPUs is what usually happens when we succumb to the need-real or imagined-to increase the processor bandwidth which, almost invariably, implies an increase in bus width. This unavoidably imposes a learning curve, and

performance gains in some areas may be offset by losses in others.

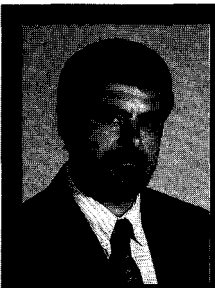
For example, the 80C186 is fairly popular in a certain class of embedded applications. Although it comes with many desirable attributes, it's not without its deficiencies when it comes to real-time processing. It seems the designers have ballasted its interrupt control unit with lead.

As is usually the case, there are different ways to address a given problem. As I've shown in previous columns, Dallas Semiconductor, through a combination of architectural refinements and sheer clock speed, has given the basic 8031 architecture a significant boost while still keeping us on familiar turf. This is no mean feat considering the meager resources available in the fundamental 8031 framework. But remember, despite its problems, this is the architecture that keeps them coming back for more.

This leads us to the predicament: "if you fix it too much, you lose the reason you kept it around so long." This may seem obvious, but it has gone unheeded by certain manufacturers in their unbridled quest for performance gains. The very latest transmogrified 8031s from Intel and Signetics are said to possess greatly improved CPUs and execute 8031 code. The only catch is that the code must be recompiled to run on the new hardware. This may not come as a surprise from Intel since they seem to have all but lost touch with the 8-bit controller arena, but it's a bit more perplexing with Signetics. I think these guys are seriously overestimating the amount of work engineers are willing to do for the upgrade. Let's watch to see what happens.

So, to come up with a faster processor, you can increase the bus width, streamline the instruction set, or just make the thing run significantly faster. The 80C320 uses the latter two methods since maintaining full 8031 compatibility is apparently one of Dallas's prime directives.

With peripherals, particularly data converters, it is a lot more cut and dry. If you need greater analog resolution, then you get a chip that simply resolves more bits. But even here,



John finishes his series on the DS80C320-based ec.32 board by covering the analog I/O, digital I/O, and power supply sections. Once complete, the ec.32 makes an ideal high-speed, lower-power embedded controller.

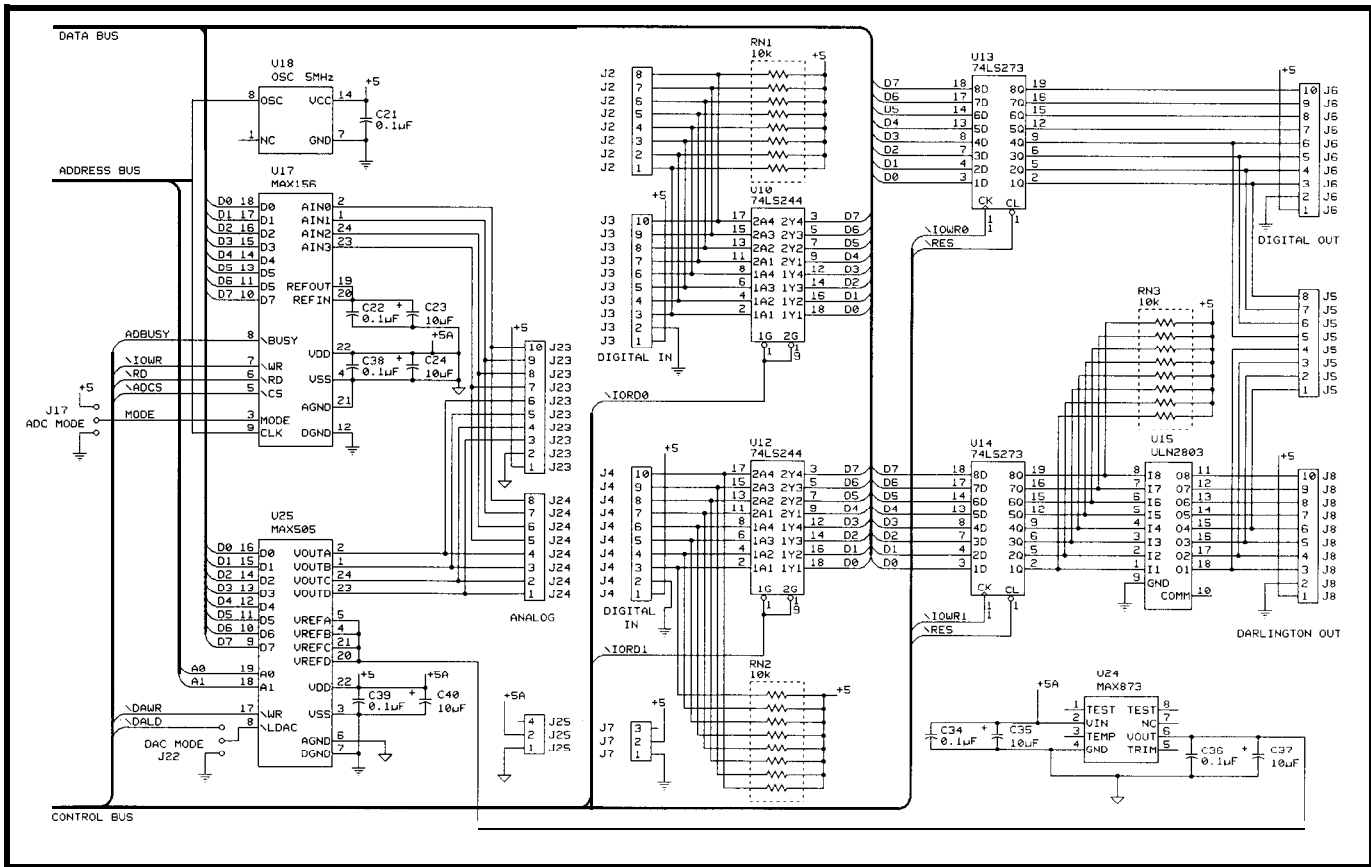


Figure 1—The MAX156 ADC and MAX505 DAC provide analog I/O on the ec.32 while ULN2803 high-current drivers buffer the board's output ports.

there are tricks you can use if you are so inclined. (Think of how you could get an extra bit of resolution out of an ADC using two successive differential conversions of opposite polarity.) Primarily, you need to avoid the mistake of associating limited conversion resolution with a lack of functional refinement. A survey of the parts available reveals that the 8-bit converter arena is full of parts designed to handle general purpose as well as a variety of special needs.

When developing a product to meet a specific set of requirements, the peripheral selection criteria should be well understood, so the decision-making process is relatively straightforward. But, things are different when designing a general-purpose instrument like the ec.32. Obviously, it's impossible to second guess what uses the product will ultimately be put to, even though the built-in feature set dictates to a great extent the range of possible applications the system is suited for. The ec.32 attains surprising levels of performance using an advanced 8-bit processor. Why not

surround this processing core with a set of equally adept 8-bit peripherals?

### ANALOG I/O

Figure 1 shows the ec.32's I/O section. Analog I/O is supplied in the form of a four-input ADC and a four-output DAC. Both converters resolve to 8 bits, are relatively fast, and contain special features that make them particularly useful when used in conjunction with a high-speed processor such as the 80C320. They both operate over an input span of 0-2.5 V. The ADC has a built-in reference whereas the DAC's reference voltage is externally supplied.

The MAX156 ADC handles the analog inputs, converting at a rate of 3.6  $\mu$ s per channel with a 5-MHz conversion clock. Simultaneous sampling of all inputs is accomplished with the inclusion of a built-in track-and-hold circuit for each channel. This capability eliminates timing differences when simultaneous multichannel sampling is a necessity.

A conversion is started on the falling edge of the write strobe, which

also causes all the channels to be sampled. This simultaneous sampling takes place regardless of the number of channels actually taking part in the subsequent conversion. On the rising edge of the write pulse, the mux configuration data is loaded, and \BUSY goes low indicating a conversion is in progress. The conversion proceeds sequentially, starting with the lowest channel. When all channels have completed their conversion, \BUSY goes high and the converted analog data is held in an internal 4 x 8 RAM. The processor can now access the RAM contents with consecutive \RD pulses, and a new conversion can be started at any time by pulsing \WR.

Typically, the MAX156 is set up via firmware before being used. The setup is performed by writing to a configuration register. Items set up by bits in this register include selecting the channel to be configured, unipolar or bipolar operation, single-ended or differential front-end, and single- or multichannel operation. When fully configuring the chip, these parameters would be specified for each channel.

This may sound like a lot of overhead, but in fact, the chip only needs to be set once at power-up. The setting remains in effect until the processor intentionally modifies the configuration register or until power is lost.

Although the overhead associated with setting up the MAX156 is minimal, and you only have to figure out how to do it once, you might still not want to wade through the myriad options. This is especially true if your application only uses the converter. And, the MAX156, despite some of its special features, makes a fine general-purpose converter.

Well, I'm a big fan of handling simple requirements simply, and the MAX156 aims to please. Looking again at Figure 1, you see a jumper block connected to the MODE pin. For simple applications, the MODE pin can be hardwired to specify the type of conversion within certain, somewhat limited, constraints. Pulling MODE low, a four-channel single-ended conversion is selected, whereas with the MODE pin high, a two-channel differential conversion will be performed. Just like in the programmable mode, the conversion is initiated on the falling edge of write, but in this case, any data on the I/O pins is ignored. Leaving the MODE pin open requires the firmware-based configuration steps that I outlined above.

The MAX505 4-channel DAC complements the MAX156's simultaneous sampling capability with its ability to update all of its outputs simultaneously. And, although it is an excellent converter, it is basically a dumb peripheral and is in no way programmable (unlike the MAX156).

The MAX505 is a 4-channel, voltage-output DAC whose internal buffer amplifiers can swing rail-to-rail. Although the converter provides independent reference inputs to each DAC, the ec.32 doesn't make use of this feature and presents a single reference voltage to the entire chip.

Using the MAX873 2.5-V reference, the MAX505 is wired for unipolar operation over a 0-2.5-V voltage swing. Double-buffered logic inputs enable all outputs to be simultaneously updated using an asynchro-

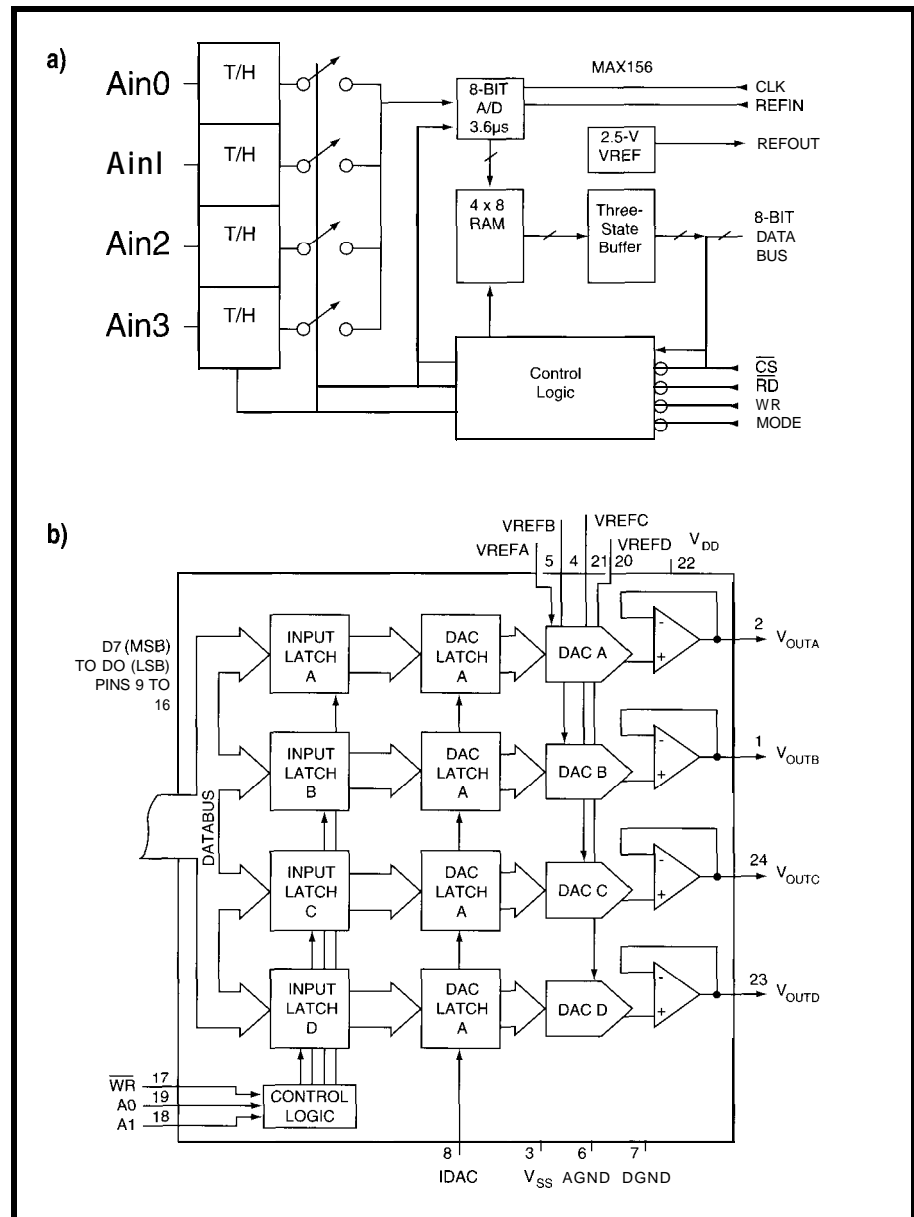


Figure 2—The ec.32's analog I/O is handled by Maxim's MAX156 analog-to-digital converter (a) and their MAX505 digital-to-analog converter (b).

nous load DAC (\LDAC) signal. Initial data is latched into the appropriate channel using the \WR strobe. \LDAC is the control pin to the secondary transparent data latch that is used to actually feed the buffer amplifiers. When all the channels have been set up with individual \WR strobes, \LDAC is pulsed to update the final stages simultaneously. To defeat double buffering (and eliminate a step from updating the DAC), just pull the \LDAC pin low using a jumper as shown in Figure 1.

For such a sophisticated data converter, you may have noticed that its digital interface is rather primitive,

consisting of an unqualified \WR input and \LDAC input. These signals must be qualified externally with a valid chip select before being presented to the DAC. This was depicted as part of the chip-select logic I covered last month. Figure 2 shows the general structure of the MAX156 ADC and the MAX505 DAC.

## DIGITAL I/O

The ec.32's digital I/O is configured in a somewhat utilitarian arrangement composed of two 8-bit input buffers and two 8-bit output latches. All inputs are terminated with pull-up resistors to +5 V. Of the outputs, 8 bits

are brought out at TTL levels, the other 8 bits are buffered using high-voltage, high-current Darlington drivers. Buffering is performed using a ULN2803 octal Darlington array. The ULN2803 is specifically designed to operate as an interface between low-level circuitry and high-power loads. The array is ideal for driving relays, solenoids, motors, or displays. Each individual driver can sink 500 mA, but you must be careful to observe the maximum package current of 3 A. This limit is described in the specifications as the maximum allowable current through the ULN2803's return pin.

The actual power dissipated by such a Darlington power IC is the sum of the individual driver dissipations, where each individual dissipation is the product of the collector-emitter saturation voltage, the collector current, and the duty cycle. The collector-emitter saturation voltage is primarily determined by the collector current, and to a certain extent by the operating temperature. What you have to remember is that, although the device specification gives you maximum voltage, current, and power ratings, these parameters are mutually exclusive. You must take care not to apply them simultaneously!

ULN2803 is just one member of a family of Darlington arrays each having an input stage optimized to interface to a particular logic family. The ULN2803 series has a series base resistor optimized for interface to a TTL driver. Figure 3 shows a schematic representation of one of the Darlington pairs contained within the ULN2803. Having described the specifications for a typical Darlington output, it might be helpful to take a look at its input characteristics. Let's examine how the driver behaves when operated in a typical environment and interfaced directly to a TTL output.

Since the ULN2803 is essentially a transistor (actually a pair), its drive capability is dependent on how it is driven. A 200-mA saturated output sink can be attained when driven with a worst-case 2.4-V TTL-logic-1 output. With a 3-V drive, which is more

representative of what you'd normally encounter, the Darlington pair will sink at least 300 mA.

Working from the ULN2803 specification, the maximum Darlington input current is stipulated at 1.35 mA. This level is not directly defined as a TTL-output parameter and falls between the defined 400- $\mu$ A logic 1 condition and the maximum output short-circuit current of about 20-55 mA. Extrapolating the corresponding

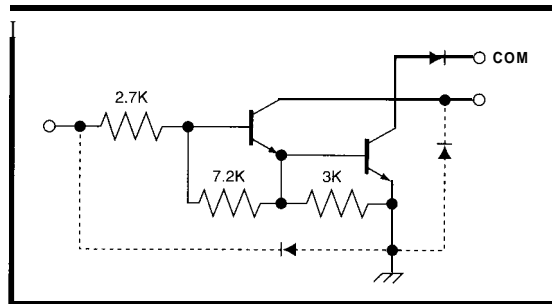


Figure 3—The Darlington pairs within a ULN2803 provide a high-current interface for the ec.32's outputs.

available voltage at this level of current yields a figure of about 3.85 V. To avoid a possibly ambiguous situation when driving very heavy loads, the ec.32 furnishes pull-up resistors to ensure adequate base current into the Darlington pair.

Finally, all of the 80C320's port 1 and most of the port 3 signals are terminated on headers. Many of these easy-to-use I/O bits have alternate functions assigned to them, including functions such as the I<sup>2</sup>C data and clock, program load control, RS-485 control, and other such functions. Since many (probably most) applications won't make use of all of these functions, all of the lines are available for alternate purposes. This not only opens up a lot of bidirectional I/O pins, but also makes a number of the 80C320's external interrupts available as well as a high-resolution event counter and timer/capture system. Timer-related functions suddenly take on a higher level of capability since the master clock to each individual timer can be independently set for a 480- or 160-ns period.

## POWER

In my opinion, one area that is recurrently given less than enough

regard in the design of single-board computer is the power supply. Most often, regardless of the power demands of the system, a simple pass stage is all that's provided. Sometimes no regulator is provided at all, requiring users to supply their own +5-V power. Although there's nothing wrong with a pass regulator, you should observe some commonsense limits about how much power you're willing to dissipate as heat.

Now, most of my designs have had to operate under rather significant power constraints. I've gotten into the habit of shaving milliamps off my operating current and microamps, and even nanoamps, off my standby requirements. Taken in this context, the ec.32 obviously doesn't qualify as fly-powered circuitry, although neither is it a contributor to global warming.

The ec.32 must be able to accommodate user-supplied loads. These loads consist of sensors and probes that hook up to the various headers and connectors or circuitry placed in the pad-per-hole prototype area. Because of this, it's important to provide some reserve power or you end up just paying lip service to your promise of expandability and extensibility. Typical pass regulators can deliver amperes of current, but that's not the problem. It's usually the power that will do you in.

To illustrate the predicament, take the ec.32 as an example. If we accept that the current consumption falls in the 300-mA range and that the raw DC comes in at a nominal 12 V, it follows that the regulator will dissipate over 2 W during normal operation. Most inexpensive, wall-mounted power packs run hot, even at their rated current, so 14-15 V is probably a more realistic figure. If this is the kind of margin you're working with to begin with, then it should be obvious you're just playing it too close.

Also, it's important to remember that the ec.32 is only a building block. In most real applications, it will only represent a portion (although a significant portion) of the total system current consumption. Although

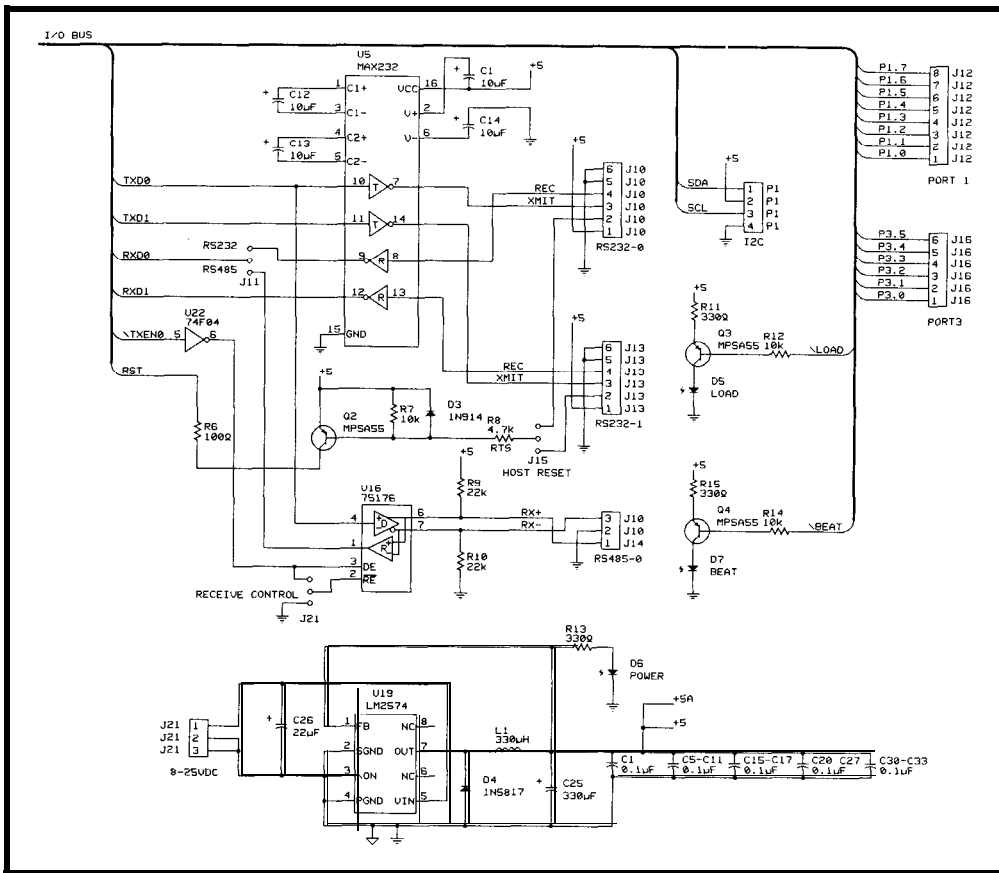


Figure 4—The ec.32 power supply section is based on the LM2574-5 switching regulator. The board also features one **dedicated** RS-232 port and a **second** serial port that can be set up for either RS-232 or RS-485.

capable of handling the basic ec.32 requirements (with adequate heatsinking), a simple pass regulator would obviously be hard pressed to handle any additional burden. And, if you wanted to run at the increased input voltage that would be desirable in a centrally powered system, your pass regulator would disintegrate.

The choice between the two approaches basically boils down to a question of whether you want to operate in the realm of constant current or constant power.

Luckily, switch-mode regulators are getting better all the time. No longer do you have to be a specialist in a variety of fringe disciplines to be able to enjoy the benefits of this superior approach to power conversion. A continuous stream of new monolithic integrated circuits is appearing on the market. Parts are available that encapsulate all of the active functions for any of the popular conversion topologies onto a single chip. Although the perceived cost differential between pass-mode and switch-mode regulators

may, at first, appear to be significant, you have to weigh all the associated costs. Heat sinks, wasted board space, and heavier front ends tend to make the differential less than you would first think. Most importantly, cooler operation enhances long-term product reliability, especially if your operating environment doesn't permit amenities such as ventilation holes.

With so many different switching regulators on the market, you could easily get bogged down. In selecting one, it pays to be familiar with the different manufacturers and what they offer. National Semiconductor has led the way in providing easy-to-use switchers with their "Simple Switcher" product line. This family encompasses a variety of topologies, requires a minimum number of support components, and even comes with PC software to help you tune the optimal configuration for your particular application. I've used various parts from this series with good results.

For the ec.32, the maximum current need not exceed 0.5 A. This

modest current requirement permits the use of the LM2574-5, a regulator with the distinct advantage of coming in an inexpensive, 8-pin plastic DIP. The regulator epitomizes electrical refinement versus mechanical brute force. That is, you can do away with the cumbersome TO220 package, the associated heat sink, and obligatory screws, nuts, and washers, and instead go with a handful of electrical parts.

The LM2574 series is a buck (step down) regulator available with fixed output (3.3 V, 5 V, 12 V, **15 V**) or adjustable voltages. It can drive a 0.5-A load with very good line and load regulation and represents the modern approach to switched-mode power conversion. It requires only four external components: two capacitors, a catch diode, and a standard inductor. The LM2574 operates at a fixed frequency of 52 kHz and can handle an input level all the way up to 40 V.

Like many general-purpose switchers, the LM2574 can be set to operate in either a continuous or discontinuous mode of operation. The inductor current differentiates these two modes. In discontinuous operation, the inductor current drops to zero. Continuous operation requires current to flow continuously through the inductor. The 330- $\mu$ H coil maintains a continuous mode of operation with inherently better load regulation, lower peak-switch currents, and lower ripple voltage. It is only slightly more complicated than a standard three-terminal pass regulator. The efficient LM2574 with its support circuitry appears in Figure 3.

## KEEP IT SIMPLE

Do the job as simply as possible with as few parts as possible. An admirable objective, but one that is sometimes difficult to fulfill. Often products are more complicated than they have to be. Less frequently but

equally disastrous is the attempt to make something simpler than possible. Between these two extremes lies a fairly narrow band in which the right measure of function and economy exists. Unfortunately, this zone gets more nebulous when designing something that is, by definition, to accomplish different things for different people.

The ec.32's primary focus on data acquisition and control should be clear. In this regard, the system's major constituents—the analog I/O and digital I/O subsystems—hold a preferential position: they get to ride the high-speed parallel data/address bus. It would have been a shame, however, not to find a way to accommodate other secondary functions as well. Had these been handled in a traditional fashion, the component count would have escalated, compromising the goal of reasonable simplicity. Reliability may also have been compromised due to excessive bus loading and unnecessarily burdening the power supply (33-MHz operation is right around the corner).

Having defined a secondary peripheral set, the best solution was to provide it with a secondary bus, one capable only of the moderate throughput needed by these peripherals. This is provided using the serial I<sup>2</sup>C that meets these objectives using just two processor pins and a bit of firmware. If I<sup>2</sup>C is not needed for a given application, these I/O pins can be reclaimed for use as general-purpose I/O.

Locally, the I<sup>2</sup>C supports an RTC, programmable interrupt interval timer, 256 bytes of RAM, and 512 bytes of E<sup>2</sup>PROM. An RJ11 connector carries the bus off-card where peripheral devices such as Mid-Tech's LCD/Keypad module can be used. This two-wire peripheral module supports a 4 x 20 LCD, 4 x 4 keypad, beeper, and several I/O lines. I<sup>2</sup>C provides the answer; functions that would have been prohibitive using conventional means become available almost for nothing.

## MAPPING THE ec.32

Often the usefulness of an embedded computer is measured in its I/O

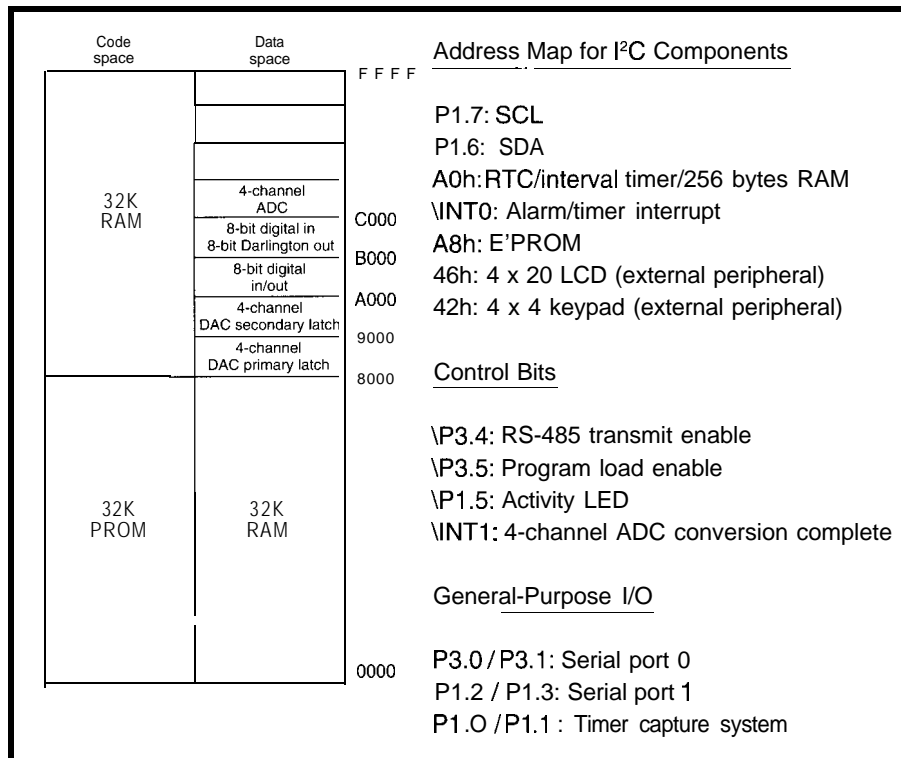


Figure 5—The memory and I/O map for the ec.32 neatly summarizes the board's features.

capacity. The ec.32 is certainly no slouch in this regard. To gain a better understanding of the system's capabilities, refer to the system memory and I/O map in Figure 5.

## TEST DRIVE THE DS80C320 ON US

The beauty of the 80C320 is its similarity to the 8031. Its strength comes through architectural improvements, higher clock rate, and totally new capabilities.

Because it goes quite a way beyond what a standard 8031 is capable of, you might find it advantageous to get your feet wet before actually cobbling together some hardware. Look to the file area of the Circuit Cellar BBS for SIM320, a PC-hosted 80C320 simulator. Capable of simulating at more than 500,000 instructions per second on a 486/33 MHz machine, this fully functional "windowed" simulator supports all the new features of the 80C320 CPU.

And, if you decide to migrate to the ec.32 as a development platform, you will be pleased to find that MON320, the compatible PC-hosted debugger, has an identical user interface. Learning curve = 0. ☺

*John Dybowski is an engineer involved in the design and manufacture of embedded controllers and communications equipment with a special focus on portable and battery-operated instruments. He is also owner of Mid-Tech Computing Devices. John may be reached at (203) 684-2442 or at john.dybowski@circellar.com.*

## SOURCES

For elements of this project, contact:

Mid-Tech Computing Devices  
P.O. Box 218  
Stafford Springs, CT 06075-0218  
(203) 684-2442

Individual chips are available from:

Pure Unobtainium  
13 109 Old Creedmoor Rd.  
Raleigh, NC 27613  
Phone/fax: (919) 676-4525

## IRS

422 Very Useful  
423 Moderately Useful  
424 Not Useful



# CONNECT TIME

conducted by Ken Davidson

The Circuit Cellar BBS  
300/1200/2400/9600/14.4k bps  
24 hours/7 days a week  
(203) 871-1988—Four incoming lines  
Internet E-mail: [sysop@circellar.com](mailto:sysop@circellar.com)

*With the ever increasing popularity of battery-operated equipment, rechargeable battery care is always an issue. NiCds are still the most popular kind of battery in use, and our first discussion deals with how they should be treated and how to breathe new life into near-dead cells.*

*Next, we look at the AC, ACT, HC, and HCT logic families and briefly talk about their characteristics and proper use. In the third thread, we try to locate the ideal natural gas detector. Finally, high-density static RAM comes in various packages that can be confusing to the uninitiated. We try to shed some light on the different kinds of RAM available.*

## NiCd battery care and feeding

**Msg#:20443**

From: MIKE TRIPOLI To: ALL USERS

I'm about to open that can of worms dealing with the cycling of camcorder batteries. I have a small stack of Sony NP-66H, 77H, 6-volt, 1400-2400-mAh battery packs. I tried deep discharge with a resistor. I then bought a "discharge/charge" unit (which had a big resistor inside). However, the things hold a charge for less and less time. I would like to try this business of "zapping" the battery to blow away the little whiskers inside. I figure it's worth a shot before trashing the things. Has anyone ever done this? What kind of voltage, current, time duration is needed? Any help is appreciated. Thanks to all.

**Msg#:20485**

From: RUSS REISS To: MIKE TRIPOLI

I have no idea if this is kosher, proper, legitimate, legal, safe, warranted, (you get the picture)...but I've brought many a pack back to life by locating the faulty individual cells within the pack and zapping them with a rather high voltage (as much as 30-50 volts-II have no idea why so much is needed since it's a current-limited power supply) set to a limit of perhaps 1-2 amps. The dead short will fold back the power supply to nearly zero volts, but after some time (minutes to hours) they sometimes do "recover" and start to take a charge once again. I'd then run a "fast charge" of say 1/5-1/3 the rated capacity for a few hours. Do this with each cell that needs it (or just fast charge the good ones). Then discharge and recharge the whole pack on

a 1/10 charge for 12-15 hours. Hey, it works for me. That's all I can say on the subject.

**Msg#:20514**

From: MIKE TRIPOLI To: RUSS REISS

Man, I swear, nothing is easy these days. I have to say you've got me scared. You've never had a battery blow up! How did you get the battery pack open? These appear to be either glued or welded closed. Have you ever tried your technique on a full pack (all cells at once?). First, I'm going to try to get the case open. Next, I'll give a shot at the high voltage treatment. Third, I'll trash the whole mess and buy new batteries. ;-)

Here's another question: Let's say you start with brand new fresh batteries. My fancy Hi-8 Sony only discharges the batteries so far before giving up. How do you deep cycle them to extend the life?

**Msg#:20542**

From: RUSS REISS To: MIKE TRIPOLI

Yes, the biggest problem is opening some of those cases. They really don't make it easy to get to the individual cells. My experience has shown that typically you cannot bring back bad cells by charging or zapping the entire pack, unfortunately. Sometimes, if you can determine the placement of the cells, you can drill tiny holes (carefully!) to access the terminals of each cell, but that's tricky too. Sorry I cannot offer any simple method.

As for deep cycling to extend life, I just don't know. Some have said that's an old myth that no longer applies to the latest technology. Others swear by it. I would think the discharge/charge unit you have would do the job, though. Good luck.

**Msg#:20607**

From: MIKE TRIPOLI To: RUSS REISS

I can't fault the charge/discharge unit yet. I got it after the batteries started crapping out. I'm going to buy one new battery and make a test of it. By the way, for everyone, while on the subject of batteries, I've been using the Renewal rechargeable alkaline system for a couple of months. [Doing toy design, we go through a TON of these puppies.] It is really working well, I recommend it. Now, if they would just come out with a 9-volt version.. .

# CONNECTIME

## Msg#:20623

From: LEE STOLLER To: MIKE TRIPOLI

I suppose everyone has their own experiences with NiCd's, but I'd like to give you mine:

NiCd "cycling," "memory effect," whatever you want to call it, was a problem 20 years ago with early cells. The NiCd manufacturers solved that one not long afterward.

"Memory" is not a problem anymore. It's been my experience that, whenever a NiCd battery will no longer hold a charge, it's gone bad. Replace it.

I've also experimented with zapping "shorted" cells. I've found that, while you may bring the cell back temporarily, it won't last. Usually, if one cell is doing this, the others are close to the same condition. The battery is bad. Hire a new one.

There is a way to get the most out of a NiCd battery. Don't keep it in the charger unless you have to. Most people use a device for a short while, then put it in the charging stand. What they should know is that, once that battery is charged up, the electrical energy from the charger gets converted to heat, which dries the cells up, and so shortens life. It is better to use the device, without recharging, until the battery is well discharged; then charge it up and remove it from the charger. (This sounds superficially like the old advice about "conditioning" or "equalizing" the cells, and is probably responsible for perpetuating the old myths. It's not the same thing.)

## Msg#:20726

From: RUSS REISS To: LEE STOLLER

My personal experience with "zapped" cells is not the same as yours. I've brought many back to life that way, and they have continued to function for a long while after (year or so, anyway). Whether or not it's worth the effort is a personal judgment. If the cells are easy to access, it sounds rather cost-effective if you don't have deep pockets to be buying new packs all the time. I'm sure you are correct though, that in general it's a sign of reaching the end of life.

## Msg#:20630

From: MIKE TRIPOLI To: LEE STOLLER

Yeah but, yeah but... You sek, I do charge and remove the battery. The problem is, the camcorder only uses the battery to a certain level, then dismisses it (i.e., starts flashing this cute little dead battery icon, then just closes up shop). I would \*almost\* prefer that the camera ran and just all of a sudden shut down, but it doesn't. I've been told I could get a couple of hours from a 2400-mAh battery. Even when new, I got something like 3/4 hour. Maybe something like one of those Ghost Busters' nuclear reactors strapped to my back, but then I'd have to get permits.

## Msg#:20938

From: LEE STOLLER To: MIKE TRIPOLI

Two possibilities suggest themselves to me:

1. The battery is indeed bad. A common battery failure mode is for one of the cells to short out. The voltage under load will be a little more than one volt low, so the low-battery indicator comes on prematurely.

2. The camcorder's low-battery circuit is out of calibration. The thing is shutting down before it ought to.

I would suggest trying to measure the battery voltage while in use. See what the voltage is when the camcorder shuts down.

## Msg#:23391

From: PELLERVO KASKINEN To: MIKE TRIPOLI

My experiences with the NiCd batteries has been somewhat limited, but here is what I believe to be essential.

When deep discharging, it \*has\* to be done on individual cells, not a series-connected stack. If you do a stack, the cell that becomes empty before the others will, on continued discharge, actually reverse in polarity. This is something that the cell definitely doesn't do gracefully. This is also the reason for many a "Low Battery" indicator. But if there is no actual action taken, like in case of the power switch forgotten in the on position, the warning does no good.

About the zapping of shorted cells into new life: the very same rule of doing only individual cells applies. In fact, let's look at the physics. A shorted cell typically has something like a whisker that makes the low resistance path inside from one electrode to the other. Good cells have the electrolyte resistance. If you force a "zap" on a stack of cells, the energy is dissipated by the resistances.  $P = I^2 * R$ . The shorted cell has next to no resistance, so all the "zap" energy is dissipated on the good cells. No, you indeed have to do any zapping on the single, shorted cell only.

I have quite successfully used this to revive NiCd battery packs for both a digital voltmeter and for my laptop computer. Of course, I had to open the packs enough to make electrical contacts to the individual cells, at first to measure which one was shorted and then to apply the "medicine" to that cell.

What I used was a 10,000- $\mu$ F capacitor charged up to 50 V (these values came from the available components and lab power supply). Then I held the end of a 12-AWG wire leading to one end of the dead cell in place, while I quickly brought another similar wire in contact. If it sparked strongly, I knew that the energy was dissipated in the spark and a new attempt had to be made. Typically it took 3 to 5 times before I got a silent contact. After that I could check the cell voltage again and notice a nonzero reading (200 to

# CONNECT TIME

600 mV). Then I could charge that cell with DC from the lab power supply and monitor its behavior. If I got the charging voltage up to 1.37 V in short time and with a low current, then I would disconnect the charging and leave the voltmeter on, to check the discharge rate. If that was too rapid, then I provided one more "zap." This generally was all it took to recondition the cell for several more months of service.

However, don't be fooled into believing that this is anything else but a temporary fix. The same cell can be fixed only a few times, maybe twice or three times. Then it is as dead as dead battery can be.

## Msg#:23392

From: PELLERVO KASKINEN To: RALPH WILLING

I recently bought a Renewal charger and eight AA batteries. So far I have not got any positive charging experience. Four of the batteries keep serving OK with the original charge. The other four showed dead on my electronic flash after no use. I tried to charge them in the Renewal charger. Only one battery indicated any acceptance of charge by the appearance of the LEDs. Once the cycle was completed, I tried the batteries again. No output! Finally I checked them individually and only one of the four was in operating shape. 8- I guess, there still is something to be learned.

---

## AC, ACT, HC, and HCT logic families

## Msg#:20072

From: DALE NASSAR To: ALL USERS

I am designing a circuit using National Semiconductor's ACT logic, and the data book only specs for  $V_{cc}=4.5\text{ V}$  or  $5.5\text{ V}$ . Why isn't  $5.0\text{ V}$  (what I need) listed? I can't figure this one.

## Msg#:20128

From: RUSS REISS To: DALE NASSAR

1. You won't see vast differences in performance at the 4.5/5.5-V levels.
2. When you supply "5 volts" as you indicate, realize that it is at some tolerance. If that happens to be  $\pm 10\%$ , then you could (and will sometimes) be operating at 4.5 or 5.5 volts! These charts/tables let you know what's going to happen at the extremes.
3. This is an example (simple one) of "worst case" design... something few new engineers are used to, but should learn quickly: Design your circuit for whatever extremes will or could be encountered. That taken care of, the "middle ground" usually (not always, though) takes care

of itself-as the world is pretty monotonic/continuous in nature. Hope that helps.. .

## Msg#:20160

From: GARY L. LEAR To: DALE NASSAR

Standard high-speed CMOS (HC) comes in two flavors: HC and HCT. Advanced CMOS comes in equivalent forms: AC and ACT. Before I directly answer your question, I would like to digress a moment. You see, you've come very close to one of my pet peeves.

HC and HCT devices (nearly all of my comments apply to AC and ACT devices as well) are nearly identical to one another. Regrettably, one of their greatest differences is that blasted "T"! The \*only\* other difference is the geometry of one of the input transistors (for each TTL-compatible input). The output stages and everything in between are identical.

The input stages are modified to change the switching thresholds to be compatible with TTL. But in addition to this, the modification also affects noise immunity, switching speed (yes, I know what the specs seem to say: HCT is faster; read on), and power consumption.

Noise immunity is drastically reduced by the obvious consideration of going from CMOS levels of 45% of  $V_{cc}$  to standard TTL levels. Switching speeds are not greatly different (a few nanoseconds), but the advantage is with HC and not HCT. The reason this comes about (despite what the data books seem to say) is that a delay is produced by the modified input stage. Spec sheets sometimes appear to show HCT switching sooner than HC. If you look closely, they are normally talking about driving the input with a 3-3.5-volt signal here. (I have most commonly seen this with AC and ACT specs.)

Most of us are still working with 5-volt-only systems, and I must admit I couldn't make much sense of these particular graphs. And since HC and HCT both generate rail-to-rail output swings... not much good anyway.

Finally, the power consumption is affected. This arises due to the fact that the modified input stage is always biased on to some degree. Standard HC inputs only conduct during transitions (as well as a very small leakage current [ $< 1\mu\text{A}$ ]). As a result, you should never use HCT in situations where minimal standby current is required.

The silly part about all of this is that HCT was never intended to be anything but a temporary measure. HCT parts were introduced as an aid to designers to make the transition from TTL to CMOS easier. Pull-up resistors could be used instead, but they are somewhat slower than HCT. All RCA, TI, and others did was include level translators at each input pin. Would you design a system that required a level-translator chip between each device? Especially when it wasn't needed and adversely affected

# CONNECTIME

system performance? Of course you wouldn't! It would be absurd. But I have seen systems that employ only HCT devices and that is exactly what they are doing.

Some engineers I have spoken to about this seem to have acquired the mistaken impression that HCT stands for high-speed CMOS, TTL version. That must be better than straight CMOS. Arrghhh.. The upshot of all of this is: don't use HCT anywhere you do not absolutely have to. A new design should be almost totally HC, with only interfaces possibly having a HCT part.

Finally, an answer to your question. Maybe you have already deduced the reason: the voltage specs for ACT devices are given at  $\pm 10\%$  of 5 volts. In reality, the parts will function correctly over the same Vcc range as AC devices. However (there is always a rub), the switching levels will no longer be TTL compatible. CMOS devices base their switching thresholds on the supply voltage. This is why TTL-compatible inputs are only specified over a restricted Vcc. Or to be even more specific, only over standard TTL voltages. Bear in mind that since power consumption varies as the square of Vcc, power consumption could vary considerably. It is also directly proportional to switching speed. This is in addition to the static power consumption.

I apologize if I took too much space to answer your question. But, I hope I did answer it, and maybe someone else can use the other information. Good luck.

**Msg#:20579**

From: BOB PADDOCK To: DALE NASSAR

NS is no longer going to make the AC/ACT stuff, end of problem! Try looking at the graphs in the front of the book to see if it will help you out.

Get their "VHC/VHCT Advanced CMOS Logic Data book."

---

## Natural gas detectors

**Msg#:22011**

From: C.D. PRITCHARD To: ALL USERS

I need a natural gas detector for residential use. Something which sounds an alarm at, say, 10% of the lower flammable limit. Battery backup and operation on 220 VAC, 50 Hz would be nice but not required. Any leads or recommendations would be greatly appreciated!

**Msg#:23538**

From: GEORGE NOVACEK To: C.D. PRITCHARD

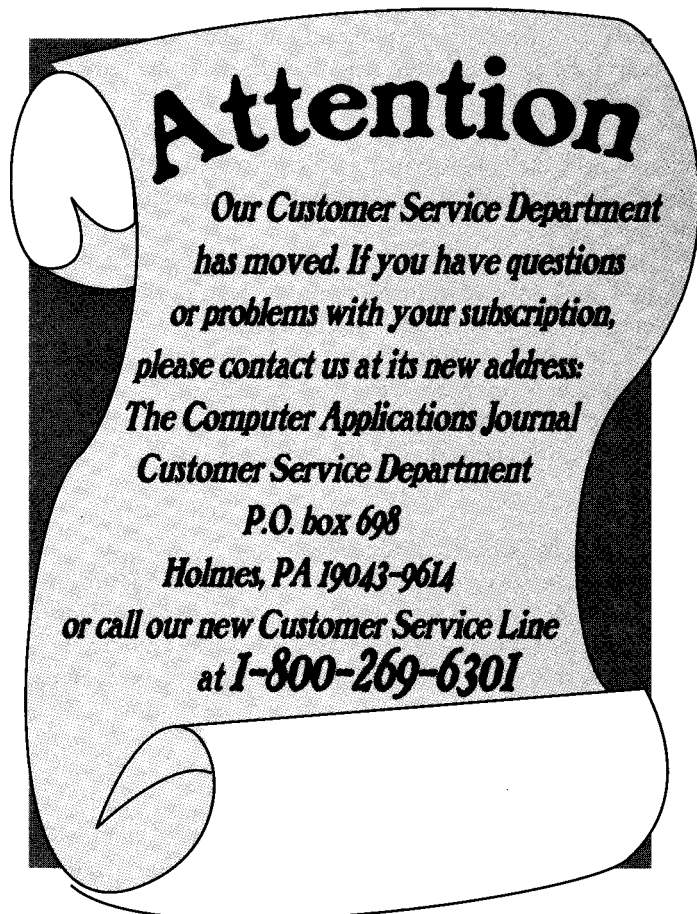
Probably the most common hydrocarbon sensor is the Taguchi sensor. It comes in several models, such as opti-

mized for natural gas or propane use. The U.S. distributor is Figaro Inc. of Chicago (wholly owned by Figaro of Japan). The sensor is a sintered crystal which is heated by a filament. It reacts with hydrocarbon molecules and changes conductivity. The output change is such that in a crude, simplest detector you could use the sensor itself to trigger an SCR. But add a comparator; it'll work better.

There are quite a few commercially available detectors using the Taguchi sensor. I believe First Alert (the smoke detector people) makes one costing about \$20. Also, marinas carry them. They are installed in bilges to detect gasoline fumes and to turn on fans automatically.

The man who runs Figaro told me once the reason Mr. Taguchi developed the sensor was that in Japan, the majority of cooking is done on gas stoves. Because of frequent earthquakes, the gas distribution is done by rubber hoses. So, while they won't have explosions due to ruptured pipes, they get them because rats love to chew the rubber. I sidetracked here. Suffice to say the popularity of the gas detectors in North America never reached that in Japan (the smoke detector situation is exactly opposite).

The major drawback of the Taguchi sensor is that the



# CONNECT TIME

filament draws a lot of power. I haven't got the catalog handy, but believe it is 5 V at 300 mA. Consequently, battery backup is a problem. There have been numerous people claiming new detectors with similar or better detection capabilities than Taguchi at significantly less power. I had been interested in prototypes, but they have been coming "real soon now" for the past fifteen years. So far I am not aware of one.

**Msg#: 23632**

**From: BOB PADDOCK To: C.D. PRITCHARD**

Ask your local gas company, or check in the phone book for a nearby propane dealer. When I was heating with LP, every few months the LP people would try to sell me one of these detectors.

---

## High-density static RAM

**Msg#: 8587**

**From: MIKE GANN To: ALL USERS**

I need some help regarding a 6585 12 5 12K x 8 pseudostatic RAM. I was looking for a 5 12K x 8 static RAM device to use in a 2M x 8 ROM emulator. I saw this part's pinout in a magazine which neglected to show the refresh input. I ordered the parts along with a data sheet.

Now I see on the data sheet "pseudostatic RAM" and three timing diagrams showing how to refresh it. If any of you are familiar with this part I could use some answers to the following:

1. I don't see how I can use these for an emulator, since I would not know when I could refresh them not knowing when it was going to be accessed by the host.
2. I think I understand how to refresh it, but I seem to be missing how often it needs to be refreshed.
3. Does anybody know of a normal 512K x 8 static RAM that's affordable [I need four] and available.

**Msg#: 8623**

**From: PAUL SHUBEL To: MIKE GANN**

I recently priced out some 5 12K x 8 static RAMs from various vendors. They seem to run in the \$150 range. Static RAM modules might work in your application and they are about two-thirds the price of the monolithic versions. Hope this helps.

**Msg#: 8749**

**From: MIKE GANN To: PAUL SHUBEL**

Thanks for your reply. Could you possibly elaborate on "static RAM modules." Exactly what they are and maybe availability and vendors.

**Msg#: 8758**

**From: PAUL SHUBEL To: MIKE GANN**

A "static RAM module" is a miniature PCB the size of a normal monolithic 512K x 8 SRAM. On this miniature PCB they mount four 128K x 8 SRAMs in TSOP (very thin) packages. They also add a 1-of-4 decoder to break the 512K address space into four 128K blocks. These modules were developed before the 512K x 8 monolithic chip was available because in some applications, circuit boards require very dense memory. Because the monolithic 5 12K x 8 chip is out, the module is reaching the end of its life. That is until the price of the monolithic chip reaches parity with the old "module version." Incidentally, now that 5 12K x 8 chips are available, some manufactures have already mounted them on a module to create the super-dense 2048K x 8 SRAM. I shutter to think what this must cost.

**Msg#: 8802**

**From: PETER HAND To: MIKE GANN**

Paul explained what they are, I think. One thing to watch out for if space is tight is that sometimes they overhang the pins by a half inch each end, so get a sight of the device (or at least the data sheet) before you part with money.

---

*We invite you call the Circuit Cellar BBS and exchange messages and files with other Circuit Cellar readers. It is available 24 hours a day and may be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps. For information on obtaining article software through the Internet, send E-mail to [info@circellar.com](mailto:info@circellar.com).*

---

## ARTICLE SOFTWARE

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

To order Software on Disk, send check or money order to: The Computer Applications Journal, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your VISA or Mastercard and call (203) 875-2199. Be sure to specify the issue number of each disk you order. Please add \$3 for shipping outside the U.S.

---

## IRS

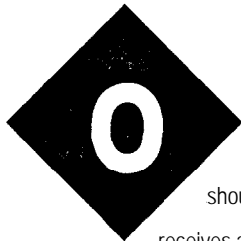
425 Very Useful

426 Moderately Useful

427 Not Useful

# STEVE'S OWN INK

## Of Patentable Value?



One of the big questions that most independent engineers face at one time or another is whether they should patent a discovery or invention they are working on. We've all heard stories about the guy who receives a %cent royalty on every aerosol dispenser or the pop-top opener on cans. The mere thought of such rewards leads some designers to approach their work as if buying a lottery ticket.

When or when not to declare ownership of an idea is as much a personal issue as a legal one. Not only do you need the intelligence to uncover something unique, but you must have the fortitude and resources to deal with the legal complications involved in declaring that ownership. For as many tales about huge royalties on existing mass-produced products, there are an equal number of horror stories about large companies creating endless legal hurdles for the small inventor. Like many aspects of life today, sometimes it's only the lawyers who triumph.

Technical publishing presents a unique dilemma. While we use the same engineering techniques and often arrive at the same conclusions, authors generally consider themselves contributing to the knowledge base rather than staking out personal ownership. Because we cover many diverse subjects and think of our presentations as technical opinion rather than documentable research, we often overlook the patentable aspects of those exhibits. Publishing a technical idea does not preclude proprietorship. In fact, in some instances, it solidly documents it for all time. Registering lasting ownership while publishing the same idea is mostly a legal question of timing. Patenting does not have to come before publishing.

A more altruistic approach to technical publishing defines technology as an adventure shared with everyone and that which is published as public domain. In all the years that I have been presenting projects, my attitude has been that technical publicity takes precedent. This conviction is not without its costs, however.

It turns out that there have been patentable ideas among those projects—some involving multimillion-dollar consumer products. One example was a lawsuit between two electronic game giants where the defense's claim that the other company's patented reset circuit was not unique and should not have been granted a patent since I had published how to build it in *Build Your Own Z80 Computers* some time before. The circuit idea had become public domain.

Presently, I'm involved in another patent fight between companies that most likely involves millions in licenses and royalties. Now, get this! Again, the defense's primary claim is that there is no patent because the concept appeared first in a *Circuit Cellar* project. It became public domain because we didn't patent it. *C'est la vie.*

My reason for illustrating these uncollected possibilities is not to gain sympathy. What is not sought as a goal cannot be viewed as a lost opportunity. Then, and still today, my primary objective is to publish technically relevant ideas and make them public.

Only in retrospect, and with fleeting perception of any ideas of ownership, should we all realize that some of what we present here is *truly* worth a million bucks.

