CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

REAL-TIME PROGRAMMING

Object-oriented Programming in Embedded Systems

Multitasking Under DOS

Z8 Forth Revisited

Direct Analog Storage Technology

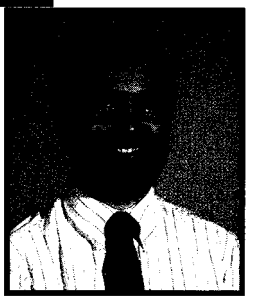
July 1993 — Issue #36



07

$3.95 U.S.

EDITOR'S INK



Code on the Fly

When we sat down to consider ideas for this month's cover, we asked ourselves, "What is real-time programming?" The obvious answer was that it is the art of writing programs that must deal with real-world situations and must react in real time to a variety of external stimuli.

I then twisted the topic around and wondered, what about "programming in real time?" An interesting idea to toss around. When you think about the traditional programming process, you conjure thoughts of Twinkie-driven late-night sessions banging at the keyboard (and banging your head against the wall) trying to find an elusive bug that only affects one small facet of your code. A short program run typically represents hours of effort.

My new idea elicits visions of a super programmer, hands poised over the keyboard, ready to spit out reams of code in response to some real-world stimuli to control a reaction as the action takes place.

Ludicrous? Perhaps not. Take the example of training a robot. You manually direct it to perform some action and the robot dutifully records all your actions. When you play back the "program," the robot repeats the task exactly as you trained it. In the strictest sense of the phrase, you just programmed the robot in real time.

Moving back to reality, our first feature article this month delves into one of the latest crazes in programming circles: object-oriented programming. If you thought OOP's usefulness was limited to windowing environments, think again. OOP is simply a methodology that is equally helpful in generating solid embedded programs that run in real time.

Another misnomer in programming circles is that you can't do multitasking under MS-DOS. Using the parent-child technique described in our second article, you can manage those pesky interrupts that often crash a machine attempting to do two things at once.

Following up on last month's modem introduction, Technical Editor Michael Swattzendruber finished his Gemini modem with some actual hardware.

In **our columns,** Ed adds some nonvolatile memory to the embedded '386SX project; Jeff looks at an old friend and breathes new life into its limited existence; Tom explores a nifty new technology that directly stores analog data without the digital middleman; John starts a pair of articles on the mysteries of recharging batteries; and Russ pulls up patents covering topics such as in-seat aircraft passenger flight information, encoding information on a video signal, and an electronic still-video camera.

Ken

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Michael Swartzendruber

ASSOCIATE EDITOR
Robert Rojas

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
John Dybowski & Russ Reiss

MEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

GRAPHIC ARTIST
Joseph Quinlan

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

Cover Illustration by Bob Schuchman
PRINTED IN THE UNITED STATES

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Susan McGill

CIRCULATION COORDINATOR
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK. THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated. 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second class postage paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U.S.A. and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95. All subscription orders payable in US. 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 7694, Riverton, NJ 08077 or call (609) 786-0409.
POSTMASTER. Please send address changes to The Computer Applications Journal, Circulation Dept., P.O. Box 7694, Riverton, NJ 06077.

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

NORTHEAST
Debra Andersen
(617) 769-8950
Fax: (617) 769-8982

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

SOUTHEAST
Christa Collins
(305) 966-3939
Fax: (305) 985-8457

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

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

Circuit Cellar BBS—24 Hrs. 300/1200/2400/9600/14.4k bps, 8 bits, no parity, 1 stop bit, (203) 871-1988; 2400/9600 bps Courier HST. (203) 871-0549

All programs and schematics in *Circuit Cellar INK* have been carefully reviewed to ensure 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 © 1993 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 Object-oriented Programming in Embedded Systems
by Mike Podanoffsky

26 PC Parent-Child Programming: A Path to Multitasking Under DOS
by H. Bradford Thompson

38 High-speed Modem Basics: The Working Hardware
by Michael Swartzendruber

46 ❑ Firmware Furnace
Memories are Made of This: The '386SX Project Goes Nonvolatile
Ed Nisley

56 ❑ From the Bench
Breathing Life into an Old Friend: Revisiting the Z8
Jeff Bach&hi

62 ❑ Silicon Update
Talking Chips
Tom Can trell

70 ❑ Embedded **Techniques**
The Art of Battery Management
John Dybowski

INSIDE ISSUE 36

2 **Editor's INK**
Ken Davidson
Code on the Fly

6 **Reader's INK**
Letters to the Editor

8 **New Product News**
edited by Harv Weiner

78 **Patent Talk**
Russ Reiss

ConnecTime 85
Excerpts from the Circuit Cellar BBS conducted by Ken Davidson

Steve's Own INK 96
Steve Ciarcia
PC Clone $5,000

Advertiser's Index 81

READER'S INK

TEMPERATURE MONITORING

Your article on Temperature Monitoring in the February '93 issue of ***The Computer Applications Journal*** was outstanding. For a person like me who has spent many hours considering the same subject, the information you presented was well received.

Since you have helped me, let me try to make you see your project in a different light. The thoughts I am about to present are not solely about heating a solarium but about home automation. I am a member of EIA's CEBus Working Group putting the standard together. I personally do not believe automating devices is what home automation is about. I doubt that such products will ever sell to the American public. The reasons are so numerous and so subjective that it would really waste my time to list them and yours to read them. This letter is on home automation: a different viewpoint.

The task you have undertaken is not stated as a goal. I have not read your prior articles but I surmise that, at least initially, you wanted to add a solarium to your home to enjoy a few of the benefits of Mother Nature such as solar heating and unobstructed views of outdoor life in Connecticut. I've gone through the same thinking process but I haven't had the guts to go as far as you.

Like a living room, your apparent thinking has focused on keeping the temperature at a constant 72" F. This is as expected, but let me list some other requirements you could have added. They represent system-level requirements for a home automation system.

The home has to be looked at as an entity. When initially designed by an architect, a complete thermal analysis was made to assure that the structure would provide a comfortable environment over the four seasons. Consideration was given to orientations on the building site, shading, wall thickness, insulation, HVAC sizing, window locations, colors of roof coverings, and so forth. The number of building parameters that come into play is extensive. If a builder or an architect can do this, why can't that information be placed in a computer and that machine be used to manage the complete thermal situation for the household?

This is what I want my Thermal Manager to do:

*know the thermal profile I wish for each room in the house

*continuously compute heat transfer to and from the room based on physical characteristics of the room, the heat supplied or removed from the room by in-house heat/cooling sources, and the heat transferred to or from the room due to condition of the outside environment

*follow the thermal profile based on sensor data that provides actual temperatures; occupancy; time of day, week, and year; weather conditions; forecasted weather conditions; and knowledge of any special events programmed

*control operation of all heat/cooling systems on site including monitoring and recording time active, heating/cooling fluid output and input temperatures, rate of fluid flow, energy source, and rate of energy absorption

•compute heat transfer for each room, each wall (or ceiling), and the building

*determine the heat/cooling supplied to the building and compute power plant efficiencies

*control ceiling fans and motorized window blinds to support attaining the desired thermal profile

I have carefully left the definition of the Thermal Profile to your imagination. It can get as sophisticated as one wishes. However, information on heat transfer, power plant efficiencies, and solar heating capability are numbers that every automated home should have. These are valuable byproducts of the system described. They indicate how well your systems are functioning, their deficiencies, your home's deficiencies, and also give clear indication when they are failing. It is one form of Integrated Diagnostics.

I am what Jeff Fisher referred to as an OOZ: an Object Oriented Zealot. He clearly pointed out that CEBus was not completely object oriented. The type of system I have described requires the application of object-oriented technology. I have built a model of a hotel building in Las Vegas to demonstrate the ease of computing heat transfer. The model could be attached to building HVAC systems to provide control and history information. That is the basis of my approach to home automation. All control of hardware originates in software models that capture the hardware functionality and "know" how the hardware should respond to current operational conditions-and does just that.

CEBus is needed to send messages from node to node in the home environment. These messages are seen by most as moving from device to device; from the controller to the controlled. In the system that I visualize, the messages move from software structure to software structure; from object to object. Control software in the node determines how a node should function and does so.

I wish you well on your solarium project. There are more people out there trying to do the same type of thing than you probably realize.

Frank Edden
The Workhouse, Huntington, N.Y.

READER'S INK

EMBEDDED DEVELOPMENT ON A MAC

I just read "Steve's Own INK" in the March issue. I agree most vigorously with his article "PC Trials," about what a pain the PC is to set up and make run in the Windows environment. I have watched the pain of one friend who tries to keep a dozen '386s running Windows limping along in the local junior college. These computers are only used for instructing classes on DOS, word processing, databases, and spreadsheets.

I have another friend in the same college who keeps the Macintosh computer lab running. His Mac network has a larger number of computers of tremendous variety. The interesting thing is, just like in the commercial Steve cites, the Mac users have few problems.

Now for my question: I use Macintoshes for everything I can in my business, but I have noticed a real lack of cross-compilers and development systems that will run on my Mac II. Much of the hardware that I would like to use is designed for the PC parallel port, not a nice serial port that I could use. How about an article on what software and hardware is available for people who would prefer to use their Macintosh to develop software controllers?

Fred Johnson
Knoxville, Ill.

We, too, have noticed a distinct lack of microcontroller development tools designed to run on the Mac. Perhaps one of our readers has more information that they'd be willing to share with us.-Editor

We Want to Hear from You

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

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



The bug stops here

And the headaches, cold sweats and other symptoms associated with debugging real-time embedded applications. Paradigm DEBUG offers you choices: • Intel or NEC microprocessors • Remote target or in-circuit emulator support • C, C++ and assembler debugging • Borland, Microsoft and Intel compatibility.

Kickstart your embedded system with the only debugger family to have it all. Give us a call for Paradigm DEBUG . before it's too late!

800-537-5043

PARADIGM DEBUG

Proven Solutions for Embedded C/C++ Developers

PARADIGM: (607) 748-5966
FAX: (607) 748-5968

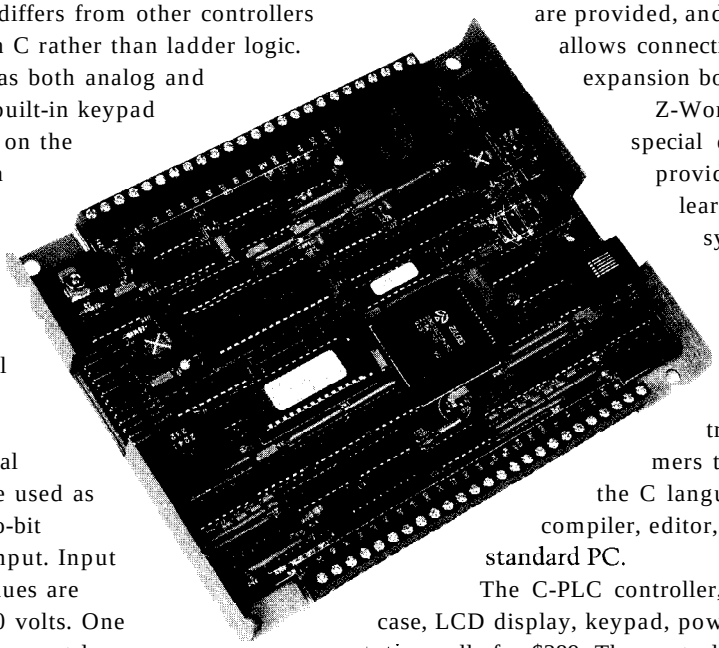
#103

NEW PRODUCT NEWS

Edited by Harv Weiner

C PROGRAMMABLE CONTROLLER

The C-PLC, a C-Programmable Logic Controller from Z-World Engineering differs from other controllers because it is programmed in C rather than ladder logic. It is a self-contained unit, has both analog and digital I/O, and includes a built-in keypad and LCD. The unit is based on the Zilog Z180 processor with a 6.144-MHz clock, programmable timers, time/date clock, watchdog timer, power-fail detector, as well as EPROM, SRAM, EEPROM, and serial ports.



The C-PLC standard features include six universal inputs, each of which can be used as a 0-lo-volt analog input (lo-bit resolution), or as a digital input. Input threshold and hysteresis values are adjustable between 0 and 10 volts. One input can receive a 20-mA current loop without an external resistor. Seven digital inputs can accept voltages from -48 to +48 volts with the logic threshold at 2.5 volts. Also included are two counters, a precision analog input, two analog outputs, two relays, ten lines each capable of driving inductive loads up to 300 mA, and a 10-volt reference output.

The standard operator interface includes a 2-line by 20-character liquid crystal display and a 12-key keypad. The interface features the ability to scan multiple menus and change parameters with only five keys, plus one additional help key. Keypad legends are easily customized. A full-duplex RS-232 port and RS-422/RS-485 port are provided, and an expansion header allows connection to custom I/O expansion boards.

Z-World's Dynamic C and a special device-specific library provide a powerful, easy-to-learn software development system for the C-PLC. Extensions for ladder logic and function block programming have been added to make it easy for traditional PLC programmers to make the transition to the C language. The interactive compiler, editor, and debugger run on a standard PC.

The C-PLC controller, complete with metal case, LCD display, keypad, power supply and documentation sells for $389. The controller board alone sells for $289 and the Dynamic C software sells for $195.

Z-World Engineering
1724 **Picasso Ave.**
Davis, CA 95616
(916) 757-3737
Fax: (916) 753-5141

#500

ASM UTILITY LIBRARY

EMS Professional Shareware is now shipping an updated version of its ASM **Utility Library**. The new version has 368 public domain and shareware products for professional assembler programmers. The products are compressed onto nine 1.44M-byte disks or one CD-ROM. All products in the library are described in an indexed database which accompanies the library. When the programmer needs to locate a particular type of assembler product, they can find it quickly by vendor, name, type, or by using a free text search across descriptions. The library contains a variety of file types, including: arrays, BIOS, code, communications, compression, date/time, debugger, diagnostic, disassembler, disk I/O, DOS, driver, editor, environment, format, graphics, hardware, interrupt, keyboard, language interface, library, lookup, math, memory, menu, mode, mouse, MS Windows, network, OOP, patch, print, reference, screen, strings, toolkit, TSR, tutorial, utility, video, XASM, and other types.

The library sells for $59.50 on diskette or CD-ROM and has a 30-day money back guarantee.

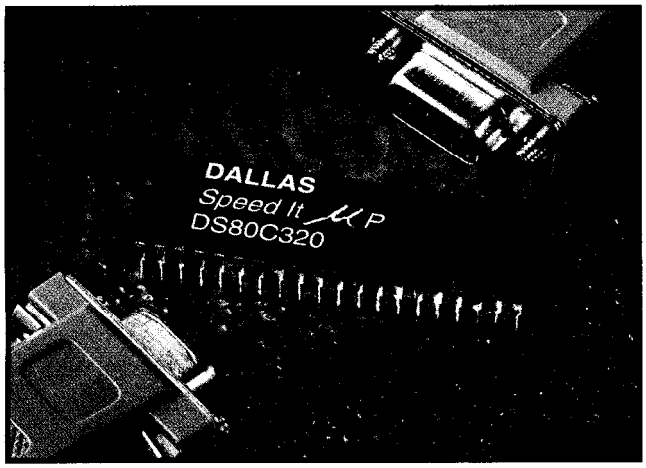
EMS Professional Software
4505 Buckhurst Ct.
Olney, MD 20832-1830
(301) 924-3594
Fax: (301) 963-2708

#501

NEW PRODUCT NEWS

8051-COMPATIBLE HIGH-SPEED MICROCONTROLLER

Dallas Semiconductor has announced a new microcontroller that is a drop-in replacement for the ubiquitous 8051. The **DS80C320** High-speed Micro runs at clock speeds up to 25 MHz and over 6 MIPS throughput. With the High-speed Micro, older designs can be



updated without changing processor architecture, software, or development tools.

The DS80C320 maintains full compatibility with the original 805 1. It uses the same instruction set and is pin-compatible with the 80C3 1 and 80C32. Any existing software development tools, such as assemblers or compilers, can still be used. In addition, the DS80C320's internal timers can be run at their old speed, allowing real-time software to function correctly when the chip is dropped into an existing design.

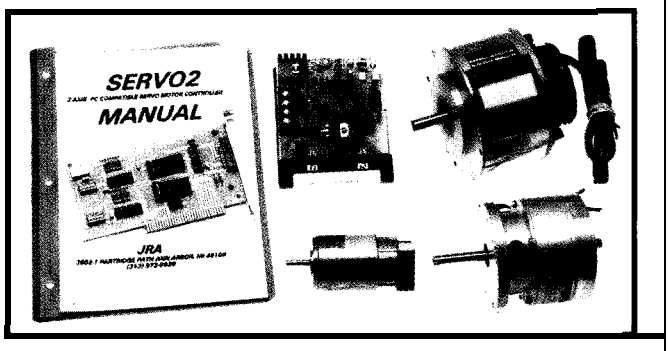
The DS80C320 provides several extras in addition to greater speed. These include a second full hardware serial port, seven additional interrupts, a programmable watchdog timer, and power-fail interrupt and reset. The DS80C320 also provides dual data pointers (DPTRs) to speed block data memory moves. It can also adjust the speed of off-chip data memory access to between two and nine machine cycles for flexibility in selecting memory and peripherals.

Like any CMOS product, the DS80C320 draws less power when run more slowly. Since it is more efficient than a standard 805 1, it can do the same job running at less than half the frequency. By simply changing the crystal, a designer can reduce the power consumption of an 80C32 design, with no loss in performance, by using the DS80C320.

The DS80C320 will be available in 40-pin plastic DIP, 44-pin PLCC, and 44-pin PQFP. In large quantities, the DS80C320 DIP package sells for $6.50.

Dallas Semiconductor
4401 South Beltwood Pkwy. . Dallas, TX 75244
(214) 450-0448 . Fax: (214) 450-0470 **#502**

TWO-AXIS SERVO MOTOR CONTROLLER



A PC-compatible, two-axis servo motor controller has been announced by JRA. The SERVO2 interface plugs into any PC-compatible motherboard and provides independent control over two motors. Feedback is provided by an optical encoder on the motor shaft.

The controller operates in two modes. In position mode, the user enters maximum motor speed in RPM and final position in encoder counts. The motor shaft will remain locked at this position until a new command is issued. In velocity mode, the user selects motor speed and direction. In either mode, motor velocity may be changed during the move.

Using JRA motors, the SERVO2 will accurately control speed from 0 to 2700 RPM with a position resolution of 0.17". Motor movement is smooth and vibration free throughout the velocity range. The system offers an affordable, low-vibration replacement for stepping motor systems in XY positioning and robotic applications.

The SERVO2 software allows the user to read motor position or velocity in real time, run both motors simultaneously, create motion sequences by entering motor velocity and destination, plot position versus time, and teach moves via the keyboard. The software is written in Quick Basic and program listings are provided.

The SERVO2 controller is priced at $350 with manual and software.

JRA . 3602-I Partridge Path • Ann Arbor, MI 48108 • Voice/fax: (313) 973-0928 **#503**

NEW PRODUCT NEWS

ANALOG VOLTAGE MEASUREMENT VIA PARALLEL PORT



The **AD1010** from B&B Electronics allows an IBM PC or compatible computer to be connected to the outside world using the computer's parallel port. Its eight analog inputs have a voltage range of -5 to +5 VDC, with a conversion time of less than 5 µs per channel. The speed, resolution, and flexibility of this unit make the AD1010 ideal for measuring voltages from lab experiments, potentiometers, sensors, and various other devices.

The AD1010 can operate in three different modes: single-ended, differential, and pseudo-differential. In single-ended mode, the eight input channels are converted with respect to a reference. In differential mode, the inputs are grouped in pairs. The voltages of the inputs are converted with respect to the other input of the pair. In pseudo-differential mode, all of the inputs are converted with respect to one input. Pseudo-differential mode is helpful when there is a variable DC offset voltage applied to a group of inputs.

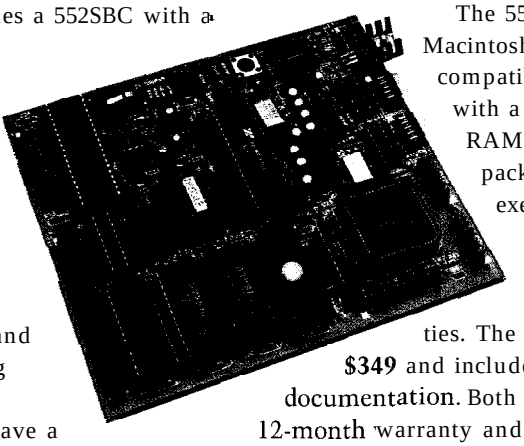
The AD1010 requires a DC supply capable of providing IO-18 V at 50 mA. It features a resolution of 10 bits plus a sign bit. The conversion time (10-bit plus sign) is 4.4 µs max and (8-bit) is 3.2 µs max. Reference output voltage error is 2%. The unit measures 3.8″×2.4″×0.9″.

The AD1010 comes with an instruction manual and diskette containing demonstration programs written in Quick Basic, Pascal, and C. The demonstration programs can be used to test or monitor the ADIO10. The source code for these programs is included on the diskette, and the routines can easily be modified for a specific application.

B&B Electronics Manufacturing Co.
4000 Baker Rd. P.O. Box 1040 . Ottawa, IL 61350 . (815) 434-0846 . Fax: (815) 434-7094 **#504**

LOW-COST SINGLE-BOARD COMPUTER

A single-board computer based on the Signetics 80C552 microcontroller has been announced by HiTech Equipment Corporation. The **552SBC** is available in two versions-a standard development board and an OEM board. The development board includes a 552SBC with a debug monitor and documentation.



The 552SBC board includes an 8-channel, 10-bit ADC; two PWM D/A outputs; 40 digital I/O lines; and three independent RS-232 outputs. Two of the serial ports can be configured to use RS-422 or RS-485 protocols by changing chips. These peripherals are complemented by a battery-backed, real-time clock and up to 16K bits of EEPROM for storing configuration information.

The four 28-pin JEDEC sockets have a flexible GAL address decoding scheme. Two of the sockets can be configured as bank-switched ROM areas with simple GAL equation changes. One of the sockets has battery back up ability through the DS1210 chip.

The monitor on the development board allows the designer to conveniently debug code in real time. Software drivers for keyboard scanning, LCD interface, and serial port communication are available as linkable assembly language code modules.

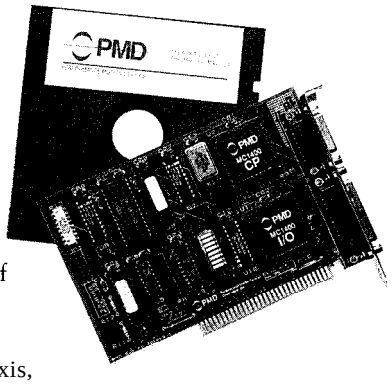
The 552SBC can be used with an Apple Macintosh, or an IBM PC/AT [or 100% compatible) running DOS 2.0 or higher with a minimum of 5 12K bytes of RAM. Any standard communication package is required to download executable code.

The OEM version of the 552SBC single-board computer sells for $149 in single quantities. The development version sells for $349 and includes the debug monitor and documentation. Both products are backed with a full 12-month warranty and unlimited technical support.

HiTech Equipment Corp.
9400 Activity Rd. . San Diego, CA 92126
(619) 566-1892 . Fax: (619) 530-1458 **#505**

NEW PRODUCT NEWS

MOTION CONTROL DEVELOPER'S KIT



The MC1400 Developer's Kit from Performance Motion Devices simplifies the task of designing MC1400-based systems. The MC1400 is a multiaxis, DSP-based, motion control chip set that provides four axes of servo control and includes advanced features such as complex contouring, velocity feedforward, servo filtering, and electronic gearing.

The MC1400 Developer's Kit consists of an ISA-bus compatible board and a software package that can exercise all of the features of the MC1400. The kit can be used as a platform to develop software for use with the MC1400 chip set, or as a stand-alone MC1400 exerciser.

The PC board fits into a standard half-length board slot and accepts four axes of incremental encoder inputs with index pulses. Two types of motor drive signals are provided: sign and magnitude outputs for use with PWM amplifiers, and analog voltage signals for use with analog amplifiers. The analog outputs are provided by on-board D/A converters and are available with a 5-volt or a 10-volt range. A generic eight-bit input port is also provided.

The software package has an easy-to-use, menu-oriented interface. It provides direct low-level access to all chip functions as well as convenient higher-level routines to perform various integrated functions such as trapezoidal moves and filter parameter changes. A variety of C-source code libraries is also included.

Interfacing to external components is accomplished through DB-15 and DB-25 connectors. With lo-volt D/A converter output, the board requires an external 15-volt supply; with 5-volt D/A converter output, the board is self-powered by the ISA bus.

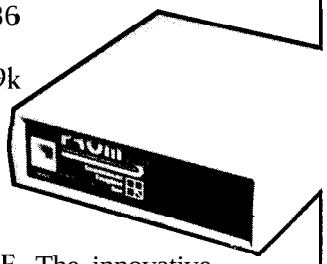
The MC1400 Developer's Kit sells for $1495. The MC1400 chip set is available for $99 in quantities of 100.

Performance Motion Devices
11 Carriage Dr. . Chelmsford, MA 01824
(508) 256-l 913 . Fax: (508) 256-0206 **#506**

68xxx, 80x86, 29k, Z80 Embedded System Developers

IF YOU *USE*...

Debug/RT	80x86/x88
CV Tools	80x86/x88
SoftProbe/386	803861486
MiniMON	Am29k
XRAY	68xxx/29k
FreeForm	68xxx
Quickfix	68xxx
CMICE	Z80
Spectra	68xxx



Then you need PROMICE. The innovative emulator that recognizes all of these.

Call us today at 1-800-PROMICE (776-6423) for your free information packet.

Grammar Engine Inc.
921 Eastwind Dr., Suite **122**
Westerville, OH **4308 1**
614/899-7878
Fax 614/899-7888

#104

Cross-Development Tools

from $50.00

Cross Assemblers

- Extensive arithmetic and logical operations
- Powerful macro substitution capability
- Unlimited include file capability
- Selectable Intel hex or Motorola hex object file format

Simulators

- Ten userdefinable screens
- Unlimited breakpoints and memory mapping
- Trace file to record simulator session

Disassemblers

- Automatic substitution of defined label names for all jumps and branches
- Automatic insertion of supplied comments and expressions

Broad range of processor specific tools Intel, Motorola, Zilog, RCA, Rockwell ,,,
All products require an IBM PC or compatible. MS DOS 2 1or greater
Same day shipment VISA, MasterCard, American Express, and COD
Unlimited technical support Thousands of satisified customers worldwide

PseudoCorp
716 Thimble Shoals Blvd.
Newport News, VA 23606
(804) 873-1947 FAX:(804) 873-2154
BBS (804) 873-4838

#105

NEW PRODUCT NEWS

LOGIC ANALYZER

A low-cost, software-based logic analyzer has been announced by F&J Associates. **Logic Analyser** gives an instant picture of what is happening. Run it, set a viewpoint, then move around in memory. The clock handler or any program in memory can then be observed.

Logic Analyser supports all Intel CPUs up to and including the i486, and disassembles code which contains standard and protected mode instructions. All orthogonal 32-bit address modes and all floating point instructions are supported, including i486 embedded instructions. Code can be displayed in any combination of 16-bit and 32-bit address and operand sizes.

Logic Analyser's general probe design allows generic events to be specified in Breakpoint, Tracepoint, and Watchpoint commands. Logic Analyser uses debug hardware built into '386 and i486 processors to allow specification of memory access breaks, input or output tracing, and interrupt watching. When all hard breaks are used, soft breaks take over-there is no limit to the number that can be defined or set.

Standard debugging is also supported, so when a problem occurs, it can be readily solved. Breakpoints can be set, and step or step over commands can be used to allow incremental execution. Event capture can be enabled and the resulting history played back. Registers, stack, and data displays are format selectable in 16- and 32-bit forms.

Logic Analyser features overlapped or tiled windows to give full control of screen display. Logic Analyser also features a Clone key which allows any window to be instantly cloned. This allows the ability to set up one or more alternate viewers, tracers, logic analyzers, or symbol tables. Most tools are clonable and can be assigned to different process threads data structures, or any object of interest.

Logic Analyser requires MS-DOS 3.1 or later, a standard 640K base memory or 1 MB of extended memory. It runs with any IBM PC/XT, AT, PS/2 or compatible with 8086, 80286, 80386, or i486 CPU. Logic Analyser sells for $199.

F&J Associates • P.O. Box 62539 . Scarborough, Ontario . Canada MI R 5G8 . (416) 438-2720 . Fax: (416) 438-8408 **#507**

How Can You Catch Nasty Race Conditions That Take All Night to Happen Without Waiting All Night?

CodeProbe.™

The new high-performance software analyzer that captures, time-stamps, and records software and hardware interrupts, DOS calls, BIOS interrupts, and user-defined events **in real-time** for analysis of race conditions, interrupt activity, and service times. CodeProbe gives you the hard facts you need to fix the big one that stands between system test and shipping your product.

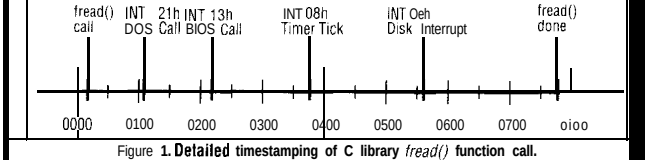


Figure 1. **Detailed timestamping of C library** *fread()* **function call.**

If **CodeProbe** can break-down a library function call into its components (above), imagine how you'll see context switches, device interrupts, and other asynchronous code

Call ***today*** *for free technical* ***specifications!***

GENERAL SOFTWARE™

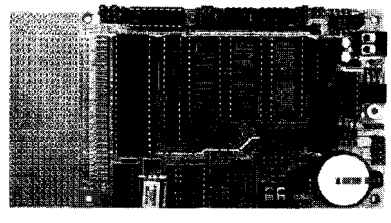
P.O. Box 2571, Redmond, WA 98073

Tel 206.391.4285
Fox 206.557.0736
BBS **206.557.4BBS**

Copyright (C) 1993 General Software, Inc All rights reserved General Software, the GS logo, and CodeProbe are trademarks of General Software, Inc

#106

8051 EMBEDDED CONTROLLERS
With Lots of Extras!



We offer a full line of low cost 80C32 embedded controllers and software tools which are ideal for developing products, test fixtures and prototypes.

Features Include:

- Low power CMOS design
- Up to 60K of code space and up to 60K of data space
- 5 to 15 volt operation
- Small form factor (3.5" * 6.5") with prototyping area
- System diskette includes application notes
- Start at $100

Available Options:

- Multifunction Board adds A/D, 24 I/O lines and more!
- BASIC-52 or Monitor/Debugger in EPROM,
- C Compiler $100 or BASIC Compiler for $300

Iota Systems, Inc.
POB 8987 . Incline Village, NV 89452
PH: 702-831-6302 . FAX: 702 831-4629

#107

NEW PRODUCT NEWS

PRINTER TESTER

A device that tests and troubleshoots standard parallel interfaces and dot matrix and daisy wheel printers has been announced by Sibex Inc. The **LP-1 Printer Tester** performs two basic series of tests that are designed to test the communication and data transfer capability between the devices. A built-in microprocessor allows the LP-1 to test printers, cables, and interface boxes without the need for a computer.

The LP- 1 verifies that the printer is receiving the correct parallel data or commands from the computer, as well as verification that the printer is issuing the proper commands to the computer.

The Printer Tester attaches in-line between the printer and computer and performs two series of tests. It monitors the communication between the computer and the printer. The **LP-1** displays (on LEDs) the line status and pulses as they change. High-speed signals and pulses are latched or stretched to make them easily readable for troubleshooting. This test verifies proper operation of the computer interface, interconnect cabling, and the printer output.

The second sequence tests the printer's ability to generate text if it receives the correct instructions. The LP- 1 Printer Tester incorporates a microprocessor which has been programmed to simulate a computer output. In this mode, the LP-1 causes the printer to initially print each character in the alphabet and then print two lines of preprogrammed text. The test sequence will repeat continuously until stopped. This repeat tests for intermittent problems and provides sufficient time for analysis of the pulsing signals.

The LP-1 is housed in a hand sized plastic case and comes with male and female DB-25 connectors. An external 110 VAC power supply and detailed instruction manual are provided with each unit. The LP-1 sells for $249.

Sibex, inc.
1040 Harbor Lake Dr.
Safety Harbor, FL 34695
(813) 726-4343
Fax: (813) 726-4434

#508

µLAN

The 9-Bit Solution is the Answer to your Embedded Network Needs

PC/XT/AT
8051
8096
80C186EB/EC
68HC11
68HC16
Z180

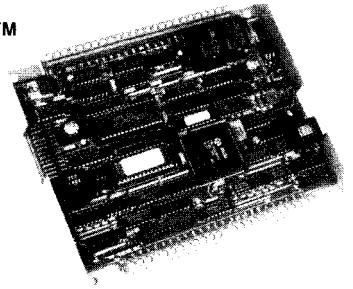
The Cimetrics Technology **9-Bit** Solutron is a complete **microcontroller** network **(µLAN)** that supports the 6051, **68HC11, 80186,** and **many other popular processors.** The **9-Bit Solution takes** full advantage of multiprocessor modes built into **microcontroller serial** ports. Our flexible software and hardware allow devalopers to **create powerful,** vet inexpensive master/slave **multidrop** embedded controller **networks.**

- Up to 250 nodes
- 16-bit CRC error checking with sequence numbers
- Low network overhead and low resource requirements
- RS-485 interface card far the PC
- Complete source code included
- Comprehensive documentation

CIMETRICS TECHNOLOGY
120 West State Street, Ithaca, NY 14850
TEL: (607) 273.5715 FAX: (607) 273.5712

#108

C-PLC™ $289!



- New C Programmable miniature controller
- Seven 1 O-bit analog inputs
- Seven digital inputs
- 1 O-bit DAC: voltage or current output
- Twelve digital/relay driver outputs
- RS-232/RS-485 serial ports
- Enclosure with LCD/Keypad available
- Expansion bus for additional, low cost I/O
- Easy to use **Dynamic C™** development software only $195!

Z-World Engineering
1724 Picasso Ave., Davis, CA 95616
(916) 757-3737 Fax: (916) 753-5141
24 hr. Information Service: (916) 753-0618
(Call from your fax and request data sheet #24)

#109

FEATURES

14 Object-oriented Programming in Embedded Systems

26 PC Parent-Child Programming: a Path to Multitasking Under DOS

38 High-speed Modem Basics: the Working Hardware

FEATURE ARTICLE

Mike Podanoffsky

Object-oriented Programming in Embedded Systems

The quest for the perfect programming methodology never ceases. The newest craze, OOP, provides experts and beginners alike a different perspective in the art of writing code. Read on to get the scOOP.

Object-oriented programming is more than just the newest technical jargon. It is a well-organized way of viewing programming which delivers, by my criteria, the ability to break down complex problems into smaller, more manageable solutions.

Object-oriented programming (OOP) is a programming concept. It is an approach intended to clarify the way you think about a problem. Because of this, you can use object-oriented concepts with any computer language including C, assembler, or BASIC. The OOP paradigm is not limited to just languages like C++ or Smalltalk, which have devoted themselves to object construction techniques.

Object-oriented code provides some of the clearest, most modular, and easiest-to-test code you'll write. OOP provides this because of the way in which the code is organized from the start. As you read what OOP is really all about, you'll discover that you may have already used some object-oriented concepts before without knowing it.

For most programmers, OOP begins to work for them when they begin to actually get a handle on the code itself. 1'11 try to demonstrate OOP

```
int Greeting(int Message )

    switch ( Message ){
        case HANG-UP:
            StopGreeting();
            RewindGreeting();
            break:

        case PHONE-ANSWER:
            RewindGreeting();
            PlayGreeting();
            break:
```

Figure I--The *behavior of the Greeting tape object is encapsulated within fhe Greeting module.*

concepts and ideas by explaining some of the key concepts in terms of objects (no pun intended) in the real world and by developing an XMODEM program.

OBJECTS

In OOP, you describe your program in terms of *objects.* Objects represent real-world components which interact with each other. For example, a CAR object moves about on HIGHWAY objects and strikes TREE objects. In OOP-speak, the behavior an object exhibits is called the object's *method.*

Take, for example, a telephone answering machine. The objects in this system are the incoming phone line, the greeting tape, the message recording and playback tape, and each of the buttons that a user may press to activate replay, rewind, and so forth. Basically, anything that has a behavior that must be modeled must be represented as an object.

The INCOMING-PHONE-LINE object detects the ring signal, answers the phone, senses voice, and detects hangup. The GREETING-TAPE object models the behavioral requirements of the greeting tape; namely, recording and playing back an announcement.

Objects communicate with each other by sending and receiving *messages.* The messages objects send to each other can consist of anything, the only limitation being their content must meet the specific requirements of your application. You can see how messages can be coded in Figure 1, which shows a simplified object routine for handling the processing of a phone greeting.

In the telephone answering machine example, when the INCOMING-PHONE-LINE object detects the ring signal and answers the phone, it sends a "Play Greeting Tape" message to the GREETING-TAPE object. In turn, the GREETING-TAPE object sends a message to the RECORD-TAPE object to begin recording.

If, at any time during these processes, the INCOMING-PHONE-LINE object detects a hangup, it sends the "hang up" message to all objects, which perform cleanup behavior appropriate to each of their methods. The GREETING-TAPE object rewinds the greeting tape and the RECORD-TAPE object stops the recording. Every object resets itself for the next message instance.

While objects may communicate with each other, they must not interfere with each other's internal processes or structures. Therefore, objects cannot change variables in data structures or set global modes. An object's method is totally self contained. You may, however, set variables and send them as a message to an object.

This tenet is essential and an important consideration in designing true object-oriented systems. It is important that an object and its behavior be totally encapsulated.

All too often in traditional programming, there has been a tendency to develop a single routine that behaves slightly differently because of modes or states. OOP tries to avoid this by developing objects with a single and distinct behavior, which means the message carries modal or state-specific information for which the object will have a single response. Of course, you are still free to break this rule and make your code just as complex and as difficult to understand as before. I can lead you to the river, but I cannot make you drink, as the saying goes.

```
typedef  struct{

    int  fof:              /* friend or foe status       */
    int  id;               /* plane's id for future ref. */
    int x, y;              /* current location in grid */
    int altitude:
    int  speed;
    int x-heading;         /* x, y, and Z headings       */
    int y-heading;
    int z-heading;

    } Plane:

Plane friend, foe:         /* two planes                 */

int Flight ( int message, Plane far * plane )

    switch ( message ){
        case UP:
            ++plane->z_heading;

        case  DOWN:
            --plane->z_heading;
```

Figure 2-I *he simple message handling for flight in this hypothetical Flight Simulator example demonstrates OOP principles.*

WHERE'S THE BENEFIT?

How is the OOP model more efficient than other programming paradigms? The functionality of the answering-machine objects is delineated by function, that is, specific objects handle specific functions. There is no behavior in the answering machine system that is not defined in

some object. This differentiation leads to cleaner code than code produced using top-down design methods, because to locate and correct any problems in the system, you isolate the problem to the object responsible for that process and correct its behavior or method. To test or debug an object, you send it a message with consistent content, and you are done when the object returns the correct response to that message.

The system created by your code becomes more functional when there is an introduction of a new object in the system. There should not be substantial changes in any existing objects (if they are truly encapsulated) since they and their message structures are not changed by the addition of the new object. The only thing that may change is the order of messages flowing through the program, or the addition of other valid message types. In other words, just add the behavior of the new object, define where and when messages should be sent to it, and how it responds to the calling object.

Listing I--Two objects make *up the XMODEM data transfer program, one for reading and processing the file (XmodemObject) and the other for performing the serial data communications (DataCommObject).*

```
/* The Xmodem Protocol:

SENDER                                          RECEIVER
command issued to send file

(sender may send any text to
 receiver including file size,
 expected time to transmit,
 or other info )
                                                command issued to
                                                receive file
                                          <-    NAK
SOH 01 FE Data[128] CKSUM                 ->
                                          <-    ACK
SOH 02 FD Data[128] CKSUM                 ->
                                          <-    ACK
SOH  03  FC Data[100] NULL[20]  CKSUM     ->
                                          <-    ACK
EOT                                       ->
                                          <-    ACK
*/

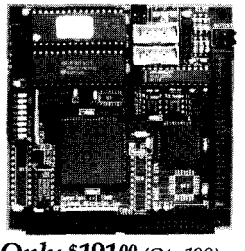
#include <dos.h>
#include <dir.h>
#include <stdio.h>
#include <string.h>

enum    {
```

(continued)

We're **Small, We're** Powerful, And We're Cheaper.

4 ***MMT-188 EB***
- 2 *serial I/O ports*
- 3 *programmable parallel I/O ports*
- *1 Meg RAM/ROM capable*
- *powerfail detect interrupt and reset*
- *counter-timers*
- *watch dog timer*
- *expansion connector*



Only $191.00 *(Qty 100)*

ALSO AVAILABLE: MMT-Z180, MMT-196, MMT-HC11, MMT-EXP

In fact, you'll get the best product for about half the price. If you're interested in getting the most out of your project, put the most into it. For the least amount of money.

Call us today for complete data sheets, CPU options, prices and availability.

Custom Work Welcome. Call or fax for complete data sheets

2308 East *Sixth Street*
Brookings, SD 57006
Phone (605) 697- 8521
Fax (605) 697- 8109

MIDWEST Micro-Tek

WE'RE SMALL BUT WE'RE POWERFUL

#110

DSP

HARDWARE

Checkoutourcompleteline of DSP boards based on powerful floating-point processors like the AT&T DSP32c (25 MFlops) and the Analog Devices ADSP-21020 (75 MFlops). Several analog interface modules are available. DSP boards start at just $995.

SOFTWARE

We have everything you need to do DSP software development, including C compilers, assemblers, source-level debuggers, algorithm development tools, and many example programs. Data can be transferred between the DSP board and host at up to 3 Mbytes/sec with the host interface library (source code included).

SOLUTIONS

Call our friendly, knowledgeable staff to discuss your applications and we'll show you how easy it is to take advantage of DSP technology.

800-848-0436

BITWARE RESEARCH SYSTEMS

400 East Pratt Street • 8th Floor • Baltimore • MD • 21202
More Info: 800-848-0436 • FAX: 410-783-7375

#111

Real- Time Multitasking with DOS

for Microsoft C, Borland C, Borland/Turbo Pascal

Develop Real-Time Multitasking Applications under MS-DOSwith *RTKernel!*

RTKernel is a professional, high-performance real-time multitasking kernel. It runs under MS-DOS and supports Microsoft C, Borland C++, Borland/Turbo Pascal, and Stony Brook Pascal+. RTKernel is a library you can link to your application. It lets you run several C functions or Pascal procedures as parallel tasks. RTKernel offers the following advanced features:

- pre-emptive, event-/ interrupt-driven scheduling
- number of tasks only limited by available RAM
- task-switch time of approx 6 µsecs (33-MHz 486)
- performance is independent of the number of tasks
- use up to 64 priorities to control your tasks
- priorities changeable at run-time
- time-slicing can be activated
- programmable timer interrupt rate (0.1 to 55 ms)
- high-resolution timer for time measurement (1 µsec)
- activate or suspend tasks out of interrupt handlers
- programmable interrupt priorities
- inter-task communications using semaphores, mailboxes, and message-passing
- keyboard, harddisk, andfloppydiskidlettmes usable byothertasks
- interrupt handlers for keyboard, COM ports, and network interrupts included with source code
- supports up to 36 COM ports (DigiBoard and Hostess boards)
- full support of NS16550 UART chip
- supports math coprocessor and emulator
- fast, inter-network communication using Novell's IPX services
- runs under MS-DOS 3.0 to 6 0, DR-DOS, LANs, or without operating system
- perform DOS calls from several tasks without re-entrance problems
- supports resident multi-tasking applications (TSRs)
- runs Windows or DOS Extenders as a task
- supports CodeView and Turbo Debugger
- ROMable
- **full source code available**
- **no run-time royalties**
- **free technical support by phone or fax**

RTKernel-C (MSC 6.0/7.0/8.0, BC++ 1.012.013.x) $495 **(Source Code: add $445)**
RTKernel-Pascal (TP/BP 5.x/6.0/7.0, SBP 6.x) $ 4 4 5 **(Source Code: add $375)**
For international orders, add $30 for shipping and handling
MasterCard, Visa, check, banktransfer, COD accepted.

MasterCard VISA

FREE DEMO DISK

On Time MARKETING
Professional Programming Tools

Karolinenstrasse 32 20357 Hamburg Germany
Phone +49 - 40 - 43 74 72 · Fax +49 - 40 - 43 51 96 · CompuServe 100140,633

#112

Free Demo Disk! Call 1-800-221-6630

What is C_thru_ROM?

*ROM Your Borland or Microsoft **C/C++Code.***

C_thru_ROM **is the complete ROM development software tool kit. It lets you run Microsoft. and Borland C and C++ programs on an embedded 80x86 CPU without using DOS or a BIOS.**
C_thru_ROM **saves you money. There are no DOS or BIOS royalties to Pay for your embedded systems.**
C_thru_ROM **is complete! It includes the following and much more:**

***Supports Borland's Turbo Debugger.**
***Remote Code View style source level debugger.**
- **ROMable startup code brings CPU up from cold boot.**
- **ROMable library in source code.**

***Flexible 80x86 Locator.**

COMPLETE PACKAGE ONLY $435. 3-Y MONEY GACK GUARANTEE,

Datalight®

107 N. OLYMPIC, SUITE 201 • ARLINGTON, WA 98223 • (206) 435-8086 • FAX: (206) 435-0253

#113

Changing the method-the code that executes and thereby creates an object's behavior-should have less impact on the overall program because the code for an object is localized. However, changing the messages an object produces could have a serious ripple effect since other objects would have to be changed to understand how to respond to the new message type.

Consider the telephone answering machine example that I outlined above. To implement a toll-call savings feature where a machine allows two rings if there are messages waiting, but won't answer for four rings when there aren't any messages waiting (to save you money when you call in for your messages] changes the method of the `INCOMING-PHONE-LINE` object, and only that object. As long as the messages flowing in and out of the `INCOMING-PHONE-LINE` object do not change, this should be the only object requiring any changes.

You can legitimately claim that OOP itself doesn't bring any advantages that a well-structured program wouldn't have provided. Still, OOP forces you to naturally develop a well-delineated, concretely defined messaging protocol and interobject interface structure.

OOP is designed to create objects once so you can reuse them in many different programs, not unlike a programming library. For example, once you've developed a communications object, potentially you may reuse it repeatedly just by sending it messages.

OBJECT DATABASE

I once wrote a database system that stored documents that contained both text and images. The entire database access was an object. The interface to the database was either a "Store Element" or a "Retrieve Element" message together with a pointer to an element, which was either some text or an image.

The database itself was considered the object which responded to store or retrieve messages. The method used by the `DATABASE` object to store elements was fully encapsulated within `DATABASE` object.

Listing 1-continued

```
    INITIALIZE = 1,
    CONNECT,
    TRANSFER,
    DISCONNECT.
    CANCEL,
    } CommMessages;

enum    {
    ASCII = 0,
    XMODEM = 1,
    } ProtocolMessages;

enum    {
    ERR_NOERRORS = 0,
    ERR_BADMESSAGE = 1
    ERR_CANTCONNECT,
    ERR_USERHANGUP,
    ERR_USERCANCEL,
    ERR-TIMEOUT,
    } ErrorMessages;

typedef  struct{
    int       file;
    char far *filename:
    char far *telephone:
    char far *buffer;
    int       bufferlen;
    int       commPort;
    int       baudRate;
    int       parity:
    int       dataBits;
    int       stopBits;

} DataCommControl;

#define FAR_ADDRESS(s,p)
    (void far *)((unsigned long)((unsigned long)((s))
                        << 16) + (unsigned long)((p)))

#define TimeOut(t,p)
    (*Time  (long)(t) > (long)(p)*18 )

static int maxerrorcount = 0:
static long far * far Time = FAR_ADDRESS(0x0040,0x006C);

#define CONTROL-Z  26 /* control z                                    */
#define  OVERWRITE  1   /* define for normal overwrite                */
#define  MAXERRORS  10 /* max number of times to retry one block*/
#define  OVRWRITIM  10 /* time to pause (none if OVERWRITE)          */
#define  BLOCKSIZE  1'28 /* transmission block size                   */
#define SENTIMOUT  80 /* timeout time in send                        */
#define TIMEOUT    99
#define  SLEEP      30 /* timeout time in recv                       */

/* Protocol characters used */
#define  SOH     0x01    /* Start Of Header                          */
#define STX     0x02    /* Start Of Text                             */
#define EOT     0x04    /* End Of Transmission                       */
#define  ACK     0x06    /* ACKnowlege */
#define  NAK     0x15    /* Negative AcKnowlege                       */
#define  CAN     0x18    /* Cancel                                    */

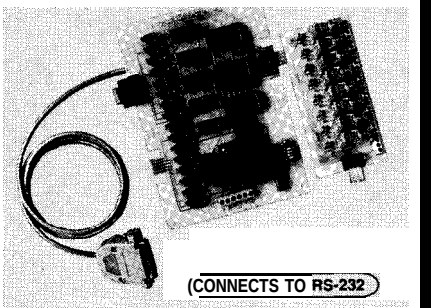
#define DEBUG 1

////////////////////  Prototypes  ////////////////////
char RecvChar(int port);
int SendChar(int port, char ch);
```

(continued)

RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

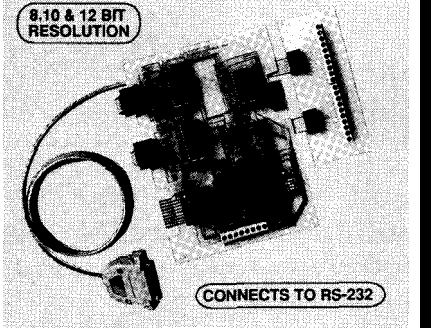


(CONNECTS TO RS-232)

AR-16 RELAY INTERFACE.....$ 89.95
Two 8 channel relay output ports are provided for control of up to 16 relays (expandable to 128 relays using EX-16 expansion cards). Each relay output port connects to a relay card or terminal block. A variety of relay cards and relays are stocked. Call for more info. RS-422 available (distances to 4,000 feet). PS-4 port selector may be used to control satellite AR-16 interfaces. (up to 16,384 relays)
RD-8 REED RELAY CARD (8 relays, 10 VA).....$ 49.35
RH-8 RELAY CARD (10 amp SPDT 277 VAC).....$ 69.95
EX-16 RELAY EXPANSION CARD (16 channel)...$ 59.95

ANALOG TO DIGITAL

8,10 & 12 BIT RESOLUTION



CONNECTS TO RS-232

ADC-16 A/D CONVERTER (16 channel, 8 bit)..$ 99.95
input temperature, voltage, amperage, pressure,energy usage, energy demand, light levels, joystick movement and a wide variety of other types of analog signals. Inputs may be expanded to 32 analog or 128 status inputs using the AD-16 or ST-32 expansion cards. 112 relays may be controlled using EX-16 expansion cards. Analog inputs may be configured for temperature input using the TE-8 temperature input conversion. RS-422 available. PS-4 port selector may be used to connect satellite ADC-16 interfaces (up to 4,096 analog inputs/16,384 status inputs and 14,336 relays). Call for info on 10 & 12 bit converters. (terminal block and cable sold separately)
ST-32 STATUS EXPANSION CARD.....$ 79.95
Input on/off status of relays, switches, HVAC equipment, thermostats, security devices, smoke detectors and other devices including keypads and binary coded outputs. Provides 32 status inputs (opto isolators sold separately).
TE-8 TEMPERATURE INPUT CONVERSION..... $49.95
Includes 8 temperature sensors & terminal block. Temperature range is minus 40 to 145 degrees F.
PS-4 PORT SELECTOR (4 channels RS-422).....$ 79.95
Converts an RS-232 port into 4 selectable RS-422 ports.
TOUCH TONE DECODER and other serial interfacing products available. Call for free information packet.

- **FULL TECHNICAL SUPPORT**...Provided over the telephone by our staff. EACH ORDER INCLUDES A FREE DISK WITH PROGRAMMING EXAMPLES IN BASIC, C AND ASSEMBLY LANGUAGE. A detailed technical reference manual is also included.
- **HIGH RELIABILITY**...engineered for continuous 24 hour industrial applications. All ICs socketed.
- Use with IBM and compatibles, Tandy, Apple, Mac and most other computers with RS-232 or RS-422 ports. All standard baud rates and protocols may be used (50 to 19,200 baud).

Use our 800 number to order FREE INFORMATION PACKET. Technical Information (614) 464-4470.

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

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

#114

When I first implemented the DATABASE object, I didn't have time to write a complete database code with indexing. I just needed to save objects for subsequent retrieval. Initially, each store message just created a DOS file where the object, text, or image was stored. Eventually, I went back and changed the DATABASE object to a fully functioning database. But the message interface remained the same. Ah, the beauty of fully encapsulated objects!

FLIGHT SIMULATOR

Let's continue our look at what constitutes objects and messages. Flight simulation is as interesting to me as flight itself. A decent flight simulation shows various views from the cockpit and flight instruments such as altimeters, compasses, fuel gauges, flaps, and so forth. To write a good object-oriented flight simulator, each instrument gauge has to be a clearly defined and independent object. Each object, of course, will have to have its own behavior.

One object is the plane itself. As the PLANE object "flies," it updates its position represented as three coordinates, x, y, and z (height), based on its "speed" and rate of ascent. The PLANE object receives several messages from a variety of sources. It receives wind and turbulence information, periodic time information to update its position, and keyboard messages. The person using the keyboard acts as the pilot and would use the keyboard to tilt the plane left or right or pull the nose up or down. These keyboard actions are passed to the plane object as messages to direct the flight path.

The instruments and gauges need to represent the current state of the flight. The PLANE object sends an "update" message to all instruments and controls. In turn, they each update the graphical representations of their instruments. Each instrument or gauge has a different method and representation, but they all respond to the same "update" message. Figure 2 shows a trivial skeleton for a Flight Simulator. It is meant to demonstrate OOP principles.

Polymorphism is the term used to describe the situation when different

Listing *1-continued*

```
int sendstring( int port, char far * string);
int sendbuffer( int port, char far * buffer, int bufferlen);
int CancelTimedOutJob( DataCommControl far * DataComm):
int DataCommObject(int Message, DataCommControl far * DataComm);
int XmodemObject (int Message, DataCommControl far * DataComm):
int ParseCommandLine(int argv, char far * far *args,
                              DataCommControl far *DataComm);

char RecvChar(int port)              { return NAK; }
int SendChar(int port, char ch)      { return ERR_NOERRORS; }

int sendstring(int port, char far * string)

    int Err = ERR_NOERRORS:

    while (*string)
        if ((Err = SendChar(port, *string++)) != ERR_NOERRORS)
            break;
    return  Err:


int sendbuffer(int port, char far * buffer, int bufferlen)

    int Err = ERR_NOERRORS;

    while (bufferlen- > 0)
        if ((Err = SendChar(port, *buffer++)) != ERR_NOERRORS)
            break;
    return  Err:


int CancelTimedOutJob(DataCommControl far * DataComm)

    DataCommObject(CANCEL, DataComm);
    return ERR-TIMEOUT;


int DataCommObject(int Message, DataCommControl far * DataComm)

    int Err = ERR_BADMESSAGE;
    int port = DataComm->commPort;
    switch (Message)

        case INITIALIZE:
            return ERR_NOERRORS;
        case CONNECT:
            Err = sendstring (port, DataComm >telephone);
            #ifdef _error_connect_
            if (status(port, . . .) != CONNECT
            Err = ERR_CANTCONNECT;
            #endif
        break;
        case TRANSFER:
            Err =
            sendbuffer(port,DataComm->buffer DataComm->bufferLen)
        break;
        case CANCEL:
            SendChar(port, CAN);
        break;
        case DISCONNECT:
            Err = sendstring(port, "ATHO")
        break:

    return  Err;


int XmodemObject(int Message, DataCommControl far * DataComm)
```

(continued)

Listing 1-continued

```
{
    int i;
    int Err = ERR_BADMESSAGE;
    int port = DataComm->commPort;
    int errorcount = 0;
    char buffer[3 + 128 + 11;
    switch (Message)
    {
        case XMODEM:
        if ((Err=DataCommObject(CONNECT, DataComm))
            == ERR_NOERRORS)
        {
            int blocknumber = 1;
            int NotEndOfFile = TRUE:
            long timeout;
            char checksum;
            int  retry:
            char far *  readbuffer = &buffer[3];
            char  ch;
            errorcount = 0;

////////////////////// wait for initial NAK /////////////////
            timeout = *Time;
            while (TRUE)
            {
                if (TimeOut(timeout, 80)) // 80 seconds max
                return ERR-TIMEOUT;
                ch = RecvChar(port);
                if (ch == CAN)
                return ERR_USERCANCEL;
                if (ch == NAK)
                break:
            }

            //// send data blocks ////////////////////////////
            while (NotEndOfFile)
            {
                buffer[0] = SOH:
                buffer[1] = blocknumber;
                buffer[2] = -blocknumber;
                NotEndOfFile = (read(DataComm->file,
                    readbuffer, BLOCKSIZE) == BLOCKSIZE);
                for (i = 0; i < BLOCKSIZE: ++i)
                    checksum += readbuffer[i];
                readbuffer[BLOCKSIZE + 11 = checksum:
                DataComm->buffer = buffer:
                DataComm->bufferLen = sizeof(buffer);
                retry = TRUE;
                while (retry)
                {
                    Err =
                    DataCommObject(TRANSFER, DataComm);
                    retry = FALSE:
                    if (Err != ERR_NOERRORS)
                    {
                        DataCommObject(CANCEL, DataComm);
                        return  Err:
                    }
                    timeout = *Time:
                    while (TRUE)
                    {
                        if (TimeOut(timeout, 10))
                                // 10 seconds max
                        return CancelTimedOutJob(DataComm);
                        ch = RecvChar(port);
                        if (ch == CAN)
                            return ERR_USERCANCEL;
```

(continued)

(multiple) objects respond differently to the same message. The word comes from the Greek "poly" for multiple and "morphism" for change. Another example of a polymorphism is in graphical applications such as AutoCAD. When an object is resized, the object is sent a "resize" message. Each object responds to the same message but in different ways because a rectangle resizes and draws differently than a circle or a picture.

OOP IN DISTRIBUTED SYSTEMS

Distributed systems are especially well suited for OOP techniques because, by their very nature, distributed systems are message-based systems. Distributed systems use messages that are not unlike messages passed between objects. A medical equipment monitor system is one example of a distributed system. One machine is situated by a patient, monitoring several vital signs while another part of the system is at the nurse's station. They not only communicate with each other, but a failure to receive periodic update information is a signal to the nurse that the equipment (or connection) may be faulty.

With an OOP design, the patient monitor doesn't care where it sends messages. That is, it should be unaware of whether its messages are sent between two object modules or between machines physically disparate. Instead, it just treats the nurse's station as an object that it sends messages to. In the early development of the product, or during debugging, there need not be physically a remotely connected nurse's station—only an object to receive and send messages.

As the development evolves into two physically separate (distributed) systems, the method of the NURSE'S-STATION object (which formerly was a device or program that only emulated the nurse's station) evolves into a full-featured component that communicates with the monitoring station. Furthermore, to make the system even more modular, create the monitoring station using two objects: a CONNECT-TO-NURSE'SSTATION objectanda MONITOR-PATIENT-STATION object.

When distributing the objects, the dynamics of the system change. Not all debugging and performance issues are resolved. For example, there are now connect time, data communication performance, speed of data transfer, and loss of data connection issues introduced when physical data communications is introduced. OOP does not eliminate these physical concerns.

INHERITANCE

One final OOP-speak term, *inheritance,* refers to an object inheriting the behavior [and messages) of another object that is already defined. This is a very commonly used technique within OOP programs and you will no doubt use it where appropriate even if you didn't know the technique had a name.

For example, suppose that several buttons in a given problem behave in exactly the same manner: when pressed they will click and continue to repeat the click until released. As each button is held down, it increments or decrements the value in a variable it controls. For example, the volume or channel control buttons on a television remote control.

Now suppose you needed a button that exhibits this kind of behavior, but must also perform other functions. One example might be a color-select button which must change hue or contrast.

Your obvious choice is to create a HUE object in which you perform any color change in response to the button being down. The HUE object sends or receives messages from its BUTTON object that handles the tasks common to all buttons. In this example the HUE BUTTON inherits the behavior common to all general buttons.

OBJECT-ORIENTED APPROACH TO DATA COMMUNICATIONS

The original intent of this article was to create an XMODEM data transfer program using object-oriented principles. The XMODEM data transfer program should be given a message that contains a telephone number, a communication port's address, and a data file. The data

Listing I-continued

```
                    if (ch == NAK)

                        retry = (++errorcount <= 10);
                        if (!retry)

                            DataCommObject
                            (CANCEL, DataComm);
                            return  Err:

                        break:

                    if (ch == ACK)
                     break:

                t // while retry
            }// while not end of file.

            //// send End Of Transmission ////////////////////
            retry = 0;
            errorcount = 0:
            while (retry)

                DataCommObject(EOT, DataComm);
                timeout = *Time;
                while (TRUE)

                     if (TimeOut(timeout, 10)) // 10 secs.  max
                        return CancelTimedOutJob(DataComm);
                    ch = RecvChar(port);
                    if (ch == NAK)
                    {
                        retry = (++errorcount <= 10)
                        break:

                    if (ch == ACK)
                        break:



        break:
        case DISCONNECT:
            DataCommObject(DISCONNECT, DataComm);
        break;

    return  Err:
}

////////////////////////////////////////////////////////////////
int ParseCommandLine(int argv, char far * far *args,
                            DataCommControl far *DataComm)

    if (argv < 2) return FALSE;
    else if ((DataComm->file = open(args[1], 0x8000)) == NULL)
        return FALSE:
    else {
        if (argv > 3)
            DataComm->telephone = args[2];
            strupr(args[argv-1]);
        if (strncmp(args[argv-1], "COM", 3) == 0)
        DataComm->commPort  =  (args[argv-1][3] & 0x0F)1;

    return  TRUE:
}
////////////////////////////////////////////////////////////////

main(int argv,  char far * far * args)
{
```

(continued)

transfer then occurs unattended-but not in the background.

The XMODEM protocol has been around for over a decade and was invented in order to transfer files between computer systems over phone lines. It was developed to overcome some of the problems inherent with data transfers that used modems such as dropped bits, random characters injected in the data stream through noise, bursts of errors, and so forth.

Under the XMODEM protocol, a file is transferred in blocks of 128 bytes preceded by a header and followed by a checksum byte. Once the block is sent, the program waits for a return acknowledgment that the block was received. If a Negative Acknowledgment is returned, the sender will retransmit the block.

The XMODEM protocol isn't very fast, nor is it absolutely error free. The 8-bit checksum is sufficiently small so that, statistically speaking, errors can occur in the data stream that will "fool" the checksum algorithm. The protocol is slow because the latency time waiting for an acknowledgment is a large percentage of the 128-byte block being transmitted.

Listing **I-continued**

```
    int showHelp = 0;
    DataCommControl DataComm;
    printf("Xmodem Send Object-oriented Example.\n\n");
    if (!ParseCommandLine(argv, args, &DataComm))
    {
        printf("Sending: Sx filename [ATDT telephone#][com
port]\n");
        printf("Receiving: Rx filename [comport]\n\n");
    }
    else
    {
         printf("Sending file %s to %s on Com%c.\n",
            args[1],
            (DataComm.telephone) && *(DataComm.telephone)
            ? DataComm.telephone :"<no phone>",
            (DataComm.commPort+1) | '0');
        XmodemObject(XMODEM, &DataComm);
        XmodemObject( DISCONNECT, &DataComm);
        close(DataComm.file);
    }
}
```

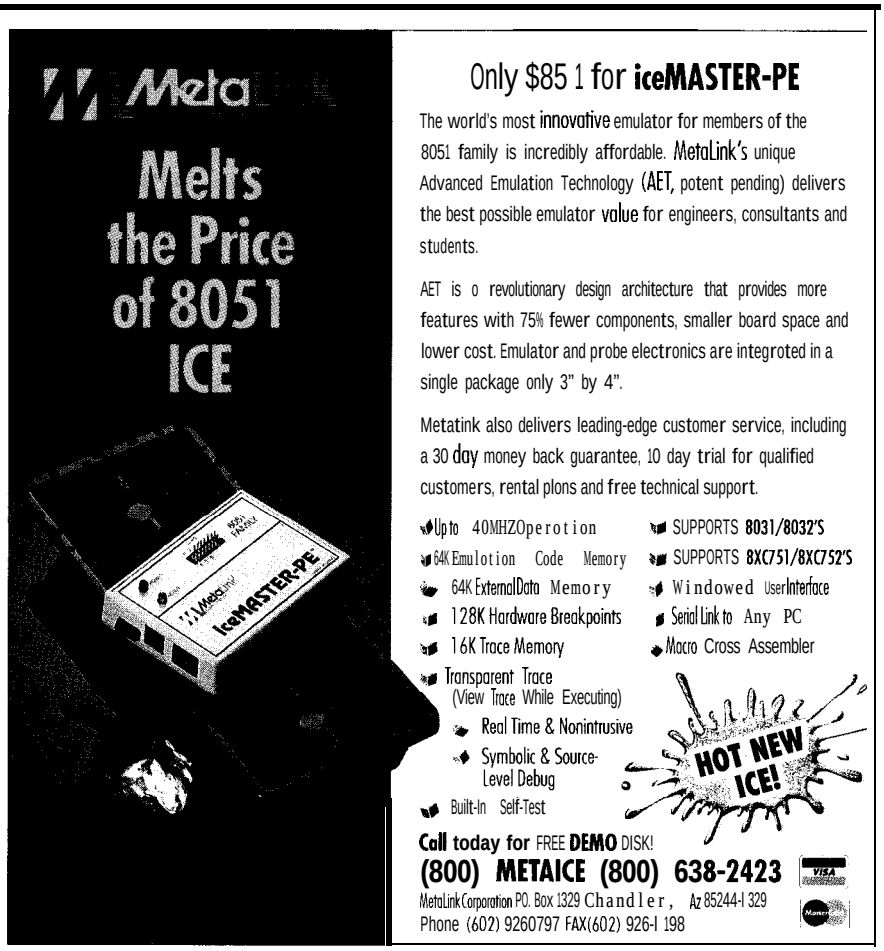
WHAT ARE THE OBJECTS IN DATA COMMUNICATIONS?

If this were rocket science it would have been easy to visualize and model real-world components. We have a harder time, however, when we cannot touch and feel the objects. Recall that an object has to have and exhibit a specific behavior and it should be able to respond to and/or send messages.

For our purposes, I'll implement the XMODEM data transfer program (Listing 1) as two objects. One object, the XMODEM object, will read and process the file. The other object, the DATA COMM object, will perform the actual serial data communications. I'll refer to these objects as the XmodemObject and the DataCommObject, respectively.

I'll have to define each message that will pass between these two objects as well as their expected data. The XmodemObject will receive a message that is a command to transfer a file through a specified comm port. The XmodemObject will pass a command message to the DataCommObject that will cause the DataCommObject to make a connection, send individual XMODEM packets, and finally to terminate the connection with a hangup message.

I further defined the interface to the DataCommObject as an int (integer value) followed by a far pointer to data, as shown below:



```
int DataCommObject (int Message,
    void far * Data);
```

This is a convenient way to pass both a command and data.

ERROR REPORTING

Finally, I need to cover the issue of error messages. If the `DataCommObject` cannot make the connection because the comm port address is wrong, or the data comm parameters don't match the port's capabilities, or the phone will not connect, then it should generate a message indicating an error. This message should be reported back to the object that invoked the `DataCommObject`.

I pass these errors back in the function return value directly. This is by far the easiest and most cost effective method and is used in the code provided with this article.

SUMMARY

I have been using OOP techniques whenever I can. I find I can better organize the functionality of my code by using the OOP paradigm. A careful study of the code in Listing 1 should give you a better idea of how OOP can be used in many embedded systems.

Finally, I'd like to leave you with this thought: My dog loves to chase cats and rabbits. I describe his behavior to computer scientists as, "Don't mind the dog's method, he's object oriented." ❑

Mike Podanoffsky has spent the last 20 years as a software developer building real-time systems, multiuser networked databases, and language compilers.

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

401 Very Useful
402 Moderately Useful
403 Not Useful

Looking for the kernel that makes application debugging both quicker and easier?



Look to KADAK for the AMX'™ real-time multitasking kernel featuring the Insight'™ Debug Tool.

AMX and Insight cooperate with such industry standard source level debuggers as CodeView,™ FreeForm,™ Turbo Debugger'™ and XRAY.™ But that's just the start.

With Insight, a single keystroke will give you a full screen view of all your tasks, timers, mailboxes, messages, semaphores and event flags. Plus, the Insight Profiler will expose those unexpected task activities and timing effects.

You'll find AMX with InSight speeds AMX™ you through the testing process letting you get your products to market quicker than ever – one good reason to count on KADAK.

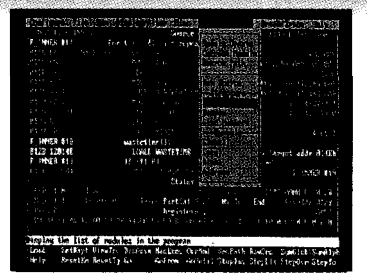
For a *free* Demo Disk – or to *order the* AMX and *InSight Manual for* only $85US – contact us t&y. Phone: (604)734-2796
Fax: (604) 734-8114

Count on KADAK.

KADAK KADAK Products Ltd. *Setting* real-time *standards* since 1978.
206-1847 West Broadway, Vancouver, BC, Canada, V6J1Y5

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

8051 68HC11 COP8 68HC05



iceMASTER™ Improved User Interface Features

Rental And 1 O-Day Trials Available

- **iceMASTER delivers productivity: easy** to learn, easy to use and fast!
- **Hyperlinked On-line help** guides you through the emulation process.
- **iceMASTER is FAST!** The 115.2K baud serial link keeps typical download times to under 3 seconds using a standard COMM port!
- **Brood support of derivative devices.**
- **Flexible user interface:** you can completely configure the windows for size, content, location and color.
- **iceMASTER is convenient!** It connects easily to your PC, requires no disassembly, nor does it take up any expansion slots. It works on any PC (DOS or OS/2), Micro Channel or EISA. Even Laptops!
- Supports **source level debug** (C and PL/M) and source level trace. 4K trace buffer with advanced searching and filtering capabilities.
- **Now virtual memory and mouse support.**

NEW 68HC05!

Call **today for FREE DEMO DISK!**
Call today to ask **about FREE8051** Macro Assembler!
(800) METAICE (800) 638-2423

MetaLink® Corporation

MetaLink Corporation P.O. Box 1329 Chandler, Az 85244-1329 Phone: (602) 9260797 FAX: (602) 926-1198 TELEX: 4998050MTLNK

#117

FEATURE ARTICLE

H. Bradford Thompson

PC Parent-Child Programming: A Path to Multitasking Under DOS

Who says multitasking under MS-DOS can't be done? While ordinary interrupt redirection can be used, it is fraught with perils. The parent-child paradigm, though, is a clean way to accomplish the task.

Suppose you want to use a PC as an instrument controller, and the system requirements involve keeping track of time, collecting data, and simultaneously performing calculations on the collected data. What's the best bet for an operating system? Is it OS/2, UNIX, Windows, Desqview, or another? Well, why not consider the most common of all-Microsoft's MS-DOS? You may have heard, "DOS isn't multitasking," but if an 8088 processor running under DOS could walk and chew gum, it would do both at once. More to the point, an 8088 under DOS may be able to run a program while it crunches your data and operates your instrument, and it could do each without unduly slowing any of the others. Of course, a more powerful 80x86 will do the job equally well, just faster.

However, the daunting task of setting up and debugging a specialized multitasking system can be frustrating. But not long ago, I discovered the power of the DOS parent-child program structure, which can remove a lot of the pain from the process. In addition, it allows a developer to divide the problem into separate (smaller) problems, because it allows the task to be divided between assembly coded routines for the low-level stuff, while the main program can still be developed using standard, high-level tools available for PCs. Curious? Hang on and I'll explain what it's all about, then provide a program framework and finally an example.

DOS INTERRUPTS

The way to get DOS to do several jobs at once is through interrupts. That's only natural since DOS communicates with both the system hardware and any running programs through interrupts. It even keeps track of the time that way, as Bruce Ackerman described in "High-Resolu-

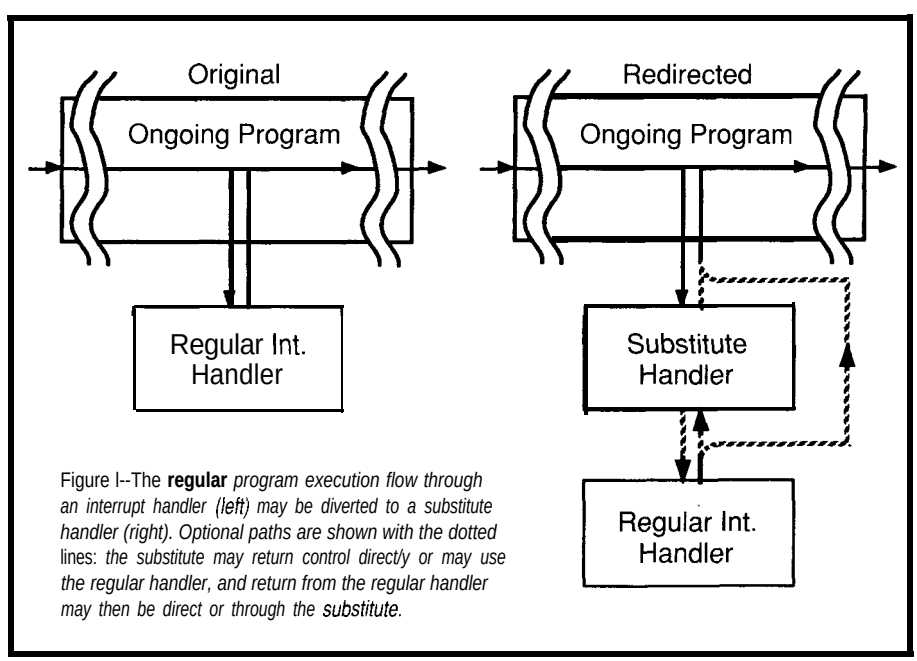


Figure I--The **regular** *program execution flow through an interrupt handler (left) may be diverted to a substitute handler (right). Optional paths are shown with the dotted lines: the substitute may return control directly or may use the regular handler, and return from the regular handler may then be direct or through the substitute.*

Listing 1—*Assembler* macros may be used *to* substitute *interrupt handlers and to restore the original paths when done.*

```
SET_INT.INC
; Macros to set and restore interrupt vectors.
; Set Interrupt Vector, saving old vector for later replacement.
; Assumes new vector is in present code segment.
;
_SetIntVector MACRO Int_No, New Int, Old_Vec
; Int_No is the number of the interrupt to be redirected.

; New_Int is the location of the new interrupt handler.
; It is assumed that the handler is in the code segment where
; the macro is used.

; Old_Vec is a four-byte space where the old interrupt vector
; is saved.

        mov     ax, 3500h + Int_No      ; Get old int. vector using
        int     21h                     ; DOS function 35h
        mov     word ptr Old_Vec[0],bx ; and save it
        mov     word ptr Old_Vec[2],es
        mov     dx,cs                   ; Put interrupt segment
        mov     ds,dx                   ;(= code segment) in ds.
        mov     dx,offset New_Int       ; put int offset in dx,
        mov     ax,2500h + Int_No       ; Install new vector using
        int     21h                     ; DOS function 35h
endm

; Reset Interrupt Vector -- companion to _SetIntVector

_ResetIntVector MACRO Int_No, Old_Vec
        lds     dx, dword ptr Old_Vec  ;Put back old int. vector
        mov     ax, 2500h + Int_No
        int     21h
endm
```

tion Timing on a PC" *(Circuit Cellar INK,* Dec. '91/Jan. '92, issue 24). A hardware interrupt, Int 08h, is generated on each "clock tick," which normally happen at a rate of 18 times a second, and the processor is diverted from the main program long enough to update the system tick count. I'll adapt Ackerman's fast-timer for use as an example for my framework program.

Keyboard input to a PC also occurs through interrupts. The keyboard produces an interrupt (Int 09h) to store each keypress (and release). A second kind of interrupt, a *software interrupt,* is then used to unload the keyboard buffer; see Chris Ciarcia's "Software at the Hardware Level" *(Circuit Cellar INK,* June/July 1991, issue 21).

The neat thing about all of this is we can divert any DOS interrupt for our own purposes, as long as we're willing to accept the consequences. DOS even provides (through an interrupt, of course) functions to help us do this. But like all powerful tools, interrupt diversion can really mess things up if they're not handled right. We need to be sure to provide for the job the interrupt normally does, as well as any additional tasks we want done, and then restore the state of the system when we're done.

The second requirement is crucial, and can really make things go wrong if it is not properly done. Imagine what happens if you've told DOS that on each keystroke or clock tick, it should jump to a location (that you think is) inside your program; if your program is not present at that location, or your program bombs and goes back to DOS without removing the jump, every thing may appear normal (since your jumped-to code may still be in memory), but as soon as you try to load anything new, the diverted interrupt will jump to some unpredictable place in a new program and try to execute whatever "instructions" it finds there! A runaway printer or a frozen machine are common results, but this kind of scenario can make your worst "PC crash fears" possible.

CALL: INT 21 h, with registers:

Register	Contents
AX	4800 (Function number)
ES:BX	Address of parameter block
DS:DX	Address of program name string

Parameter block:

Offset	Contents
o-1	Segment of environment block [0]
2-5	Address of command tail
6-l 3	Addresses of file control blocks [FFh, FFh, FFh, FFh]

Command tail:

Offset	Contents
0	Length (n) of tail text in bytes
l-n	Tail text
n+l	13h (CR) as terminator

[ ] indicates values to insert for default results.

Figure 2-A child *process* is invoked by *Int 21h function 4Bh and* is *named EXEC. The call a/so includes a parameter block and a command tail.*

HOW INTERRUPT DIVERSION WORKS

Let's see what happens during an interrupt. As I noted, interrupts can be triggered by hardware or software. The keyboard and the clock-tick interrupts, Int 09h and Int 08h described above, are examples of hardware interrupts. A program gets keyboard-buffer contents with a software interrupt, specifically Int 16h. For either type of interrupt, the same series of events occurs:

- The current program location is saved, in a subroutine fashion, to allow a safe return from the interrupt.
- *The four-byte address of the appropriate *interrupt handler* is obtained from a jump table stored in low memory. This address is called the *interrupt vector.*
- The interrupt handler (the code at the address specified by the vector) is executed.
- The handler issues an I RET, causing a return via the address that was stacked in the first step.

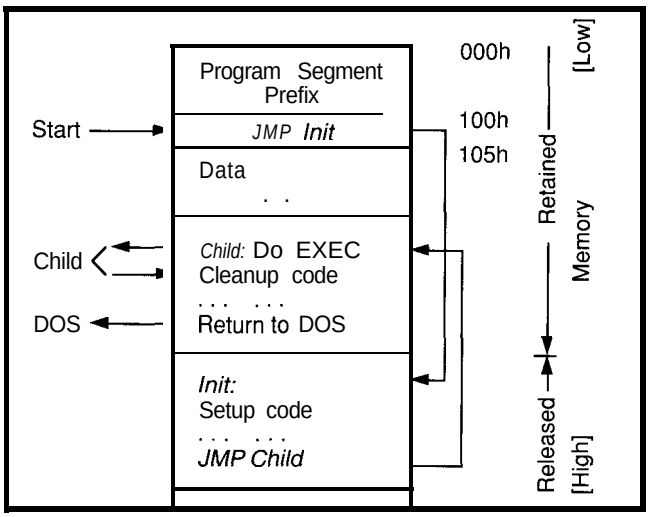


Figure 3—*The arrows show program flow through the structure of the parent program framework. Memory from location Init: on is released before the child program is called. Data is placed at fixed location 105h where child program routines can find it.*

To substitute our own task, we simply substitute our own interrupt vector in the jump table. We could do this directly, but that's risky-suppose we're in the middle of changing an interrupt vector when that interrupt occurs? DOS makes life simpler and safer by providing two functions to do the job for us. How these functions are used is shown in the first assembler macro in Listing 1.

This macro also saves the old interrupt vector. We will need this when we reset things when we're done; the second macro in Listing 1 is the one responsible for that task. Besides, we may still want to use the old interrupt handler, but just add our own code to it. After doing our thing, we simply jump to the old handler, where its I RET instruction returns it neatly back to the main program. Or, the return can be made through the substitute handler rather than direct. Program flow for the modified vector interrupt handler is shown in Figure 1, with dashed lines showing optional routes.

THREE WAYS TO SET 'EM UP

A straightforward approach to interrupt diversion is to do it within your main program. The program must also provide foolproof restoration when done. This means taking into account irregular exits as well as normal exits, including exits via break keys (Ctrl-Break or Ctrl-C). Any error-induced exits must also be provided for, particularly during debugging, unless the user of your program loves rebooting their machine.

An alternative is to set up the diversions with a TSR (Terminate-and-Stay-Resident) program triggered by a hot key. The advantage here is that the TSR code, once installed, remains in memory and can be invoked through program calls and DOS prompts. The disadvantage is the TSR remains in memory, taking up space and doing its thing, until something is done to remove or disable it.

A third choice is the parent-child program launching structure. A program, called the parent, installs the interrupt diversions and then calls up the main program as a *child process.* A child process can be passed command-line arguments just like any other program. The advantage to this approach is when the child is done, control is returned to the parent rather than to DOS. The parent can then do whatever cleanup is required (sound familiar?). Breaks during a child program need no special treatment since control just goes back to the parent. Many "bombs" in the child processes will also return neatly to the parent, which is very helpful during debugging. Of course, the parent program stays in memory while the child runs. This is not as big a drawback as it might seem. Parent programs don't have to be very big. The prototype parent described below takes less than one kilobyte of memory.

The parent-child scheme is a neat one for the interface developer. All the hardware stuff can be built into the parent, written for efficiency in assembler code, and tested with a simple child program. After the bugs are out of the parent, child programs to handle the interface-related issues can be written in any convenient high-level language. If you work out hardware systems for others who write their own analysis programs, you can let them write and fiddle with the child programs.

HOW TO CALL YOUR CHILD

A child process is invoked by (surprise!) an interrupt call, specifically function 4Bh of interrupt 21h. This is called the E X EC function. The rules for

Listing *2-The parent program framework. Compare this listing with Figure 2. Code to implement any specific application is placed at [A] through [D].*

```
;#[ A 1              Template for Parent Program
;# for construction of programs that install interrupts, etc,#
;# for use in child programs. Stuff in ### borders should be #
;# replaced by appropriate code for the specific application.11
;# In this place write the program title, etc.                #
;# Template designed for the Microsoft MASM assembler;        #
;# tested w/MASM 6.0                                          #
.NoList
Include Set_Int.Inc
.List
     .model tiny
     .8086

; Start:
          Go to one-time code that does the nitty-gritty, then
          return here for clean-up and sign-off.
;
_parent segment
     assume cs:_parent, ds:_parent ;Tell MASM about
; segment registers
     org 100h
```

(continued)

Listing *P-continued*

```
start:
            jmp     init            ;go do setup in over-write area
; [Then return to child-execution segment]

; Fix location to allow this info to be found by a child program,
; via parent address in the child PSP.  Allows up to 4-bit jmp
; address so child process can work with any future version of
; this routine.

            org     105h
;# [ B ] Parent Data
;# First goes data, etc., that child routine needs to find. #
;# Here place interrupt handlers, vectors, etc.             #
stackptr DW 0       ;Space for stack ptr during DOS EXECUTE
; Data for EXEC DOS call:
;file path buffer for EXEC DOS call. Also holds command tail
file-path db 82h dup (0)

; EXECUTE parameter block-for EXECUTE DOS call
par_blk     dw      0               ;environment segment-use 0
tailoff     dw      0               ;command tail
tailseg     dw      ?               ;pointer
            dd      -1              ;FCB pointers
            dd      1
; Error message
cr          EQU     0dh
lf          EQU     0ah
XFailedMsg  db      cr,lf
            db      'Failed to find and execute child program'
            db      cr,lf,'$'
; Local stack
stackk      dw      64 dup (0)
stackend    dw      0
;====================== Run the child process====================
;Returns here from init procedure
child:      int     21h             ;start the child process
;Clean things up:
            mov     ax,cs
            mov     ss,ax           ;Restore stack seg and pointer
            mov     sp,cs:stackptr;lost in Child routine call.
            mov     cl, 0           ;tentatively, no error
            jnc     pexit           ;Did it go OK?  If yes, exit
            mov     cl, 2           ;No. Set exit DOS errorlevel 2
            mov     dx,offset XFailedMsg
            mov     ah,9            ;DOS string write,
            int     21H             ;error message
pexit:
;# [ c 1
;# Restore vectors and other things that init routine changed#
;# Note: Stack ptr need not be restored, DOS does it on exit #
eexit:
            mov     ah, 4Ch         ;Exit to DOS
            mov     al,cl           ; errorlevel:
            int     21h             ; 0 if Child executed
                                    ; 1 if no command line arg
                                    ; 2 if execution failed
;================================================================
; Initialization code:  Used once, then released for use by child

init:
            assume  cs:_parent,ds:_parent
            mov     sp,offset cs:stackend  ;initialize stack ptr.
; Get program from command string
            mov     si, 80h         ;String source is PSP loc 80h
            mov     di, offset file-path
                                    ;Destination is file-path buffer
            lodsb                   ;1st byte of src is byte-count
```

(continued)

how the EXEC function works are given in Figure 2. The mechanics for how this function can be put to work to suit our purposes will be better understood after reviewing the listing for the prototype. A place to start is to write out the DOS command-line that would be used to call up the child program, such as:

```
C:>PATH\CHILDPGM.EXT  with
    some args
```

Now parse this command line to divide it into a program-name part (PATH\CHILDPGM.EXT) and a command tail (with some args). Each of these parts is stored separately, and has its own rules. Register pair DS:DX contains, in DOS (segment:offset) form, the location of the program name. The name may include a DOS path. The child program name *must* end in .EXE or .COM followed by a zero byte. *Don't omit the extension.* While a DOS command-line call will find a program if you omit the .EXE or .COM, the EXEC function will not.

The command tail is stored differently. The register pair ES:BX contains the location of a "parameter block," and this in turn contains the location of the command tail. Ahead of the tail is a single byte containing the length of the tail string. Following the tail is a carriage return character (ASCII 13h). The carriage return is not included in the length count. Thus our example tail would be, in assembler notation:

```
db 14,  'with some args', 13h
```

The parameter block contains three other pointers to areas we can define for the child program: the environment block and two file control blocks. We will give these default values, shown in the brackets. In particular, putting a zero in the environment pointer tells DOS to pass the child the same environment the parent got.

Our sample parent program will read these child command parts from its own command line. If the command line to invoke the child program from a DOS command line were:

```
d:>path\CHILDPGM with
      some args
```

then the command line to start the same program as a child should be:

```
d:>PARENT path\CHILDPGM.EXE
      with some args
```

Note that the command-line entry is made including the .EXE extension, and is followed by the command tail.

A PARENT PROGRAM FRAMEWORK

Now I'm ready to describe how to make a prototype parent program. The minimum tasks such a program must perform are laid out below:

(1) Fill child process program name and command tail from parent command line.
(2) Divert interrupts and do any other application-specific setup.
(3) Assign memory for use by child process.
(4) Call child via EXEC Function.
(5) Put things back as they were (i.e., undo step 2).

I've described all of these, except for number 3-assign memory for the child. When the parent (or any program) is called, DOS generously assigns it all the memory available. A parent must release what it doesn't need. In order to give the child every byte we can, I'll release not only the memory not occupied by the parent, but also the space the parent uses only once, before the child is called. This takes a little extra planning, but it means I don't need to squeeze and skimp on setup code. The memory layout of the parent is shown in Figure 3. At the DOS entry location, relative address 100h, there's an immediate jump to location Init: to perform the set up, after which we jump to Child:, which is the location of the call to the child program. Memory from Init: can be released after the code has been processed.

The data area has been placed immediately after the JMP Init statement. This fixes its location so the child process can find it. I skipped

Listing *2-continued*

```
        cmp  al. 0              ;Check for a zero count
        jz    abort             ;Quit if no argument
        mov cl, al              ;Put count in cl
        xor  ch,ch              ;clear high byte
firstin:
        lodsb                   ;Get char. from command line
        cmp   al,20h            ;discard any leading space
        jne   arg1_c
        loop  firstin
arg1_in:
        lodsb                   ;Get char. from command line
        cal, 20h                ;Check against space
        je    arg1_done
arg1_c:
        stosb                   ;Store it in file-path name
        loop  arg1_in
        jmp   asc_z             ;Skip if loop ended due to count
arg1_done:                      ;Loop ended due to space
        dec  cl                 ;so count it but don't copy it
asc_z:
        xor   al,al             ;Store a zero to terminate
        stosb                   ; the ASCIIZ string
;=== Define the new command-tail buffer & put the rest there ===
        mov   tailoff, di       ;Fill in tail-pointer from di
        mov   al,cl             ;First the count
        stosb
        inc  cl                 ;Add one so we'll copy the cr
        rep   movsb             ;Now copy string with term. cr
;# [ D 1
;# Write code to redirect interrupt vectors & do any other        ;#
system setup. Changes made here should be unmade by code at       ;#
;# [ c 1.   "Old" interrupt vectors should be stored at [ B ].    ;#
;======== Release all possible memory for use by child ========
mem_kept EQU (offset cs:init-offset cs:start+10fh)/10h

        mov   bx, mem_kept      ;This is size to be kept/para
        mov   ax,cs
        mov   es,ax             ;Segment for release reference
        mov   ax,4a00h          ;DOS release-memory function
        int   21h
; ================ Set up to run the child process ==============
                                ;Preserve stack pointer!
        mov   cs:stackptr,sp    ;DOS EXEC Interrupt loses it!
        mov   tailseg,cs        ;Command-tail segment for ptr.
                                ;get pointers for DOS function:
        mov   dx,offset file-path
;child process name/path
        mov    bx,offset par_blk
;child process par. block
        mov   ax,4b00h          ;Set for DOS "EXECUTE" function
        jmp   child             ;Leave once-only code, go to
                                ;child-process call location.
:====Error routine for missing command-line Child program name====
abort:
        mov   dx, offset NoArgMsg
        mov   ah,9              ;Quitting - no valid argument
        int   21h               ; on DOS command line
        mov   cl.1              ;Errorlevel 1 on pgm exit
        jmp   eexit
; Error message text:
NoArgMsgdb   cr,lf
        db    'No valid argument on DOS command line.'
        db    cr,lf,cr,lf,'$'

_parent ends
end     start
```

ADDING SUBPROGRAM OBJECT MODULES TO PROGRAMS

For FORTRAN, the module FTFUNC.OBJ is combined with the parent by simply listing it on the compiler command line:

```
FL MAINPGM.FOR FTFUNC.OBJ
```

To include an assembly coded routine in QuickBASIC is a little more involved, and described pretty briefly in the manuals. I'll run through the steps for our test program. A routine to be used within QB must be part of a "Quick Library," conventional extension .QLB. A Quick Library TEST.QLB can be made by:

```
LINK /Q FTFUNC.OBJ, TEST.QLB,, BQLB45.LIB
```

This is shown for version 4.5, thus the name BQLB45.LIB. Be sure to use the version of Microsoft LINK that comes with your QuickBASIC! The Quick library is then included on the command line, which now looks like:

```
FASTIMER QB.EXE FASTEST /LTEST
```

The line items in this example are, from left to right, the parent program, QuickBASIC, the test program, and the Quick Library name preceded by the switch /L. If you plan to make a stand-alone program, you should also make a regular library. A one-file library is very quickly made with Microsoft's library manager LIB, which also comes with QuickBASIC:

```
LIB TEST.LIB+FTFUNC;
```

All this works just fine when it works. However, QuickBASIC has five paths to set, and they all matter. There's a "Set Paths" submenu on the Options pull-down menu within QB. The time to get all the paths right is well spent. This will create or update a file QB.INI in your default directory; check that this file did get written. Note that QuickBASIC looks for each file in the default directory before using one of these paths.

forward five bytes, although the JMP only needs three. If I ever design a parent that uses S-byte jumps, I won't have to rewrite any of the child program subroutines.

Listing 2 shows my parent framework. Code for specific applications gets inserted at four locations, marked [A] to [D]. I've described some of the finicky details in comments. The framework can be compiled and tested as-is; it doesn't do much, but can be tested and used as a proof of concept.

Below is a simple QuickBASIC program called COPYCAT.BAS, which just prints out its own command-line arguments.

```
C$ = COMMAND$
PRINT "Copycat says: "; C$
END
```

When COPYCAT is compiled and run, its output is:

```
C:>COPYCAT Meow
Copycat says: MEOW
```

The following shows the same compiled program being run as a child, verifying that PARENT passes along the rest of the command line when it calls COPYCAT.

```
C:>PARENT COPYCAT Meow
Copycat Says: MEOW
```

An intriguing wrinkle to parent-child method is shown below:

```
C:>PARENT \QB45\QB.EXE
          COPYCAT /CMD meow
Copycat Says: MEOW
```

where the parent can use the QuickBASIC "programming environment" (QB.EXE) as its child! After QB loads itself and the COPYCAT text, pressing the *Run* key produces the output shown. I haven't tried this experiment with other programming environments, but I suspect most of them will work just as well. As a result, a child program being prepared to run in compiled form can be written, tested, and debugged within a modern, convenient programming frame.

THE FAST TIMER PARENT

Listing 3 contains the patches needed to build a parent program for the fast-timer technique that was first described by Ackerman. At the start of the patch in section [B] are two words (four bytes) reserved at location 105h for a counter. The first, tick_lo, will be incremented approximately 18 times per second. Each time tick_lo overflows (almost exactly once an hour), tick_hi is incremented. An overflow from tick_hi will occur some time in the eighth year.

The counter routine in the Patch section [B] is our substitute handler for DOS interrupt 1Ch, which is a secondary DOS timer interrupt. The instructions that divert calls on 1Ch are shown in the patch section [D], which is executed when the parent starts. We also need to change the mode of the PC's timer-that's the job of the remaining code in section [D]. Finally, patch section [C] is executed on return from the child, and it contains the steps needed to undo what's done in section [D] and turn over a "clean" PC to the next program.

COMMUNICATING WITH YOUR CHILD

The final part of our fast-timer package is a pair of subroutines that can be called in a child program to read the DOS clock and add to the data in tick_lo and tick_hi if necessary. How does such a routine find these values? When writing or compiling a program, we don't know just where in memory it will be at run time. If we want to pass data between parent and child, the child needs a way to find its parent. The DOS program loader

Listing 3—*Patches for fast-timer example. [A] through [D] should be inserted at the marked locations in Listing 2.*

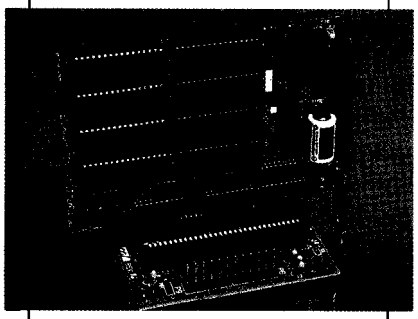
```
; [ A ]              Fast-Timer Parent Program
; Sets up fast clock for child process, allowing timing with µsec.
; res. using PC clock. Built on HBT's standard Parent Framework.
; [ B]                      Tick Data
            org     105h
tick-lo     dw      0             ;count buffer
tick-hi     dw      0
; New tick-count handler:
            assume ds:nothing   ;forbid use of ds (contents unknown)
new_int_lc:
            inc     cs:tick_lo
            jnz     New-11        ;skip next instruction
            inc     cs:tick_hi    ;unless count overflowed
New_il:
            jmp     dword ptr cs:old_vec_lc
old_vec_lc DD       [?]
; [ C ] ==========Put back the old time-tick vector =============
            ResetIntVector lch, old_vec_lc
  ---------- Restore 8253 chip mode and tick-rate ----------
            mov     al,036h       ;Reset 8253 chip
            out     43h,al        ; control register -- mode 3
            xor     al,al
            out     40h,al        ; count to 10000h
            out     40h
: [ D ] ================Set 8253 mode and rate ==================
            mov     al,034h
            out     43h,al        ; control register: mode 2
            xor     al,al
            out     40h,al        ; count to 10000H
            out     40h,al
;---------- Install new old time-tick vector, save old one ---------
            _SetIntVector lch, new_int_lc, old_vec_lc
```

conveniently places the parent location in the initial 100h-byte block of the child program, called its Program ***Segment*** *Prefix* or *PSP*. There's a DOS function to find a program's own PSP. Finally, once we've found the parent, we know the location of its data area, since we carefully placed this at parent address 105h.

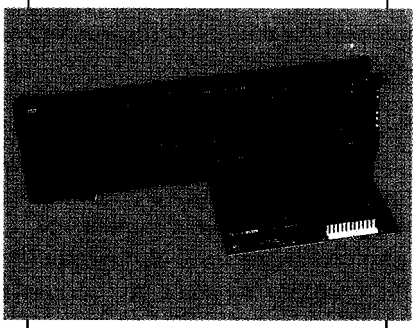
My routines for doing this are shown in Listing 4. They can be called from Microsoft assembler, FORTRAN, or QuickBASIC (other languages should work as well, but these are the ones I've tested). Near the start of FTimer, between labels JO and J1, is a section that finds the parent tick-count locations. A call to Int 21h, Function 62h returns the child's PSP location, which in turn serves as a bast to fetch the parent location. The parent's base address is then stored for future use, so this code section only needs to be executed once. The instruction JMP WORD PTR JPtr provides a switch: the pointer JPtr originally points at J0, but the first two instructions after J0 reset it to point at J1, shortening all subsequent FTimercalls.Therestof FTimer uses Ackerman's technique to fetch the DOS clock count as the least-significant two bytes of a four-byte integer, and gets lo_tick from the parent as the most-significant bytes. It also fetches hi_tick and squirrels it away.

For many applications, the one-hour time range of FTimer will be enough. If not, a call to FTimer may be followed by one to FTimerHi, which fetches the last saved hi_tick value. In both FORTRAN and QuickBASIC, these routines are used as functions. They have no arguments, and so avoid one source of pitfalls in interlanguage programming. A simple QuickBASIC demonstration program is shown in Listing 5. I've included a function that translates the time into seconds. Combining a subprogram object module with a main program is straightforward in FORTRAN, but a little tricky in QuickBASIC. Instructions for doing both are in the sidebar.

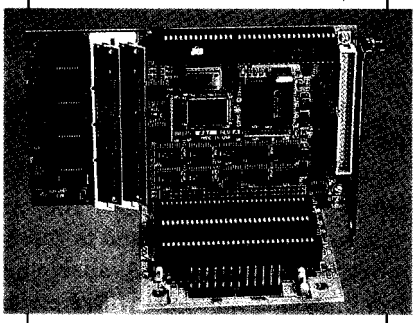
VMAX
QUALITY PRODUCTS
RESPONSIVE SERVICE
RELIABLE DELIVERY



SOLID STATE DISK — $135"
½ 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 2 M
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 U.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

Fax for fast response!
295 Airport Road
Naples, FL 33942

#118

PROSPECTS

Now that I've shown you how the parent-child structure works, let's take a look at some things that can be done with it. First of all, I've barely scratched what can be done with the timer interrupts, Int 08h and 1Ch. Parent routines could be modified to acquire data from an instrument on every DOS clock tick. The frequency of DOS "ticks" could be changed to provide more frequent sampling. The method for doing this was also described by Ackerman, and earlier by C. Claff in BYTE (1986 IBM Extra Issue, p. 254). The handler can place the data in buffers for later use by an ongoing program. That program can, if you like, set flags to direct and control the data-gathering process on the fly.

I would like to share one of my experiences as an example of how speeding up the "tick" can be put to good use. I once built a program that logged signals from a bank of 144 phototransistors and also watched an ADC output, by using Int 08h sped up to 146 ticks per second on an 8-MHz PC/AT-level machine. The main program, written in FORTRAN, could retrieve data as needed and wring out the results in real time. This approach to the data acquisition process slowed the main program down by less than 10%.

The interrupts most commonly diverted are those for the keyboard, namely Int 09h and 16h. All "hot-key" add-on programs grab these. You can roll your own here too, and in some cases you may need to. Available hot-key programs can clash with a given application since almost every application program handles keyboard input differently. Diverting these interrupts within a parent program can be used to add a hot key to the QuickBASIC programming environment. Doing so let me lay down templates for ***WHILE*** and FOR loops, ***IF-ELSE*** structures, and the like. By the way, if you do adapt Ciarcia's code, be sure to add traps for enhanced-keyboard functions-Int 16h functions 10h and 1 lh. Apparently these can be handled just the same as functions 00h and Olh, respectively.

Listing 4—*Fast-timer* functions. These *should be linked to child programs to implement the fast timer that is set up by the parent.*

```
; Fast Timer Function -- These functions, used with program
; FASTIME, return the time to a BASIC or FORTRAN program,in
; internal timer pulses, frequency 1.1931817 MHz.
.8086
_FTFunc_TEXT        SEGMENT BYTE PUBLIC 'CODE'
_FTFunc_TEXT        ENDS
; Function FTimer
; No arguments.  Returns the current time in units of 1/1.19318
; microseconds,  in DX (high 16 bits) and AX (low 16 bits).
; QuickBASIC:
; DECLARE FUNCTION FTimer&.
;   The four-byte result is an unsigned integer, and overflows at
;   1 hr. If read as signed 4-byte integer, the count goes from
;   0 to 2,147,483,647 and then from -2,147,483,648 to -1. then
; overflows and repeats.  The overflow causes a second counter
;   to be incremented, providing an additional two bytes of total
;   count. The overflow count is read by FTimerHi& if needed.
_FTFunc_TEXT        SEGMENT
            PUBLIC FTimer
FTimer      PROC    FAR
            push    bp              ;Microsoft Standard Entry
            mov     bp,sp
            push    es
            JMP     WORD PTR JPtr ;To JO the first time, then J1

;   The code between JO and J1 is executed on the first entry,
; bypassed on later calls.  It finds and saves the segment of
; the parent program,  so that the tick recorded there can be
;   used as the most significant part of the value.
JO:
            mov     ax, jptr+2      ;reset to skip this next time
            mov     jptr. ax
            mov     ax,6200h        ;"get PSP" DOS call
            int     21h
            mov     es,bx           ;put psp seg in es
            mov     ax,es:16h       ;get parent PSP seg
            mov     parent_seg,ax   ;and save it
jl:
            mov     es,parent_seg
            mov     bx,es:105h      ;get initial tick reading
            mov     cx,es:107h          and tick overflow
            mov     al,06h
            out     43h,al          ;latch the count value
            in      al,40h          ; read LS byte
            mov     ah,al
            in      al,40h          ;read MS byte
            xchg    al,ah           ;swap
            not     ax              ;complement/increment, to
            inc     ax              ; change count-down to count-up
            mov     dx,es:105h      ;get present tick reading
            cmp     bx,dx           ;check whether lo-tick changed
            je      ft_ret          ;if it didn't change, we're ok;
            cmp     ax,8000h        ;if it did, decide which to use.
            jb      chk_hi          ; count small: use second
            mov     dx.bx           ; count large: use first
chk_hi
            cmp     bx,0ffffh       ;hi_tick ok unless
            jne     ft_ret              lo-tick overflowed.
            mov     cx,es:107h      ;if so get present value
            add     cx.dx               and perhaps subtract 1
                                        [dx is either 0 or -1]
ft_ret
            mov     hi_count,cx     ;save hi-tick
            pop     es
            mov     sp,bp           ;Standard exit.
            pop     bp
```

(continued)

Listing 4—*continued*

```
            ret
FTimer      ENDP
;
; Function FTimerHi
;  No  arguments.  Returns the overflow (high count) is AX.  This
;    function is meaningful only after a call to FTimer -- the
;    value returned is the high tick count at the time of
;  FTimer  call.
; QuickBASIC:  DECLARE  FUNCTION  FTimerHi%
;    The count returned is an unsigned integer.  Each high count
; is worth one hour.
            PUBLIC  FTimerHi
FTimerHi    PROC    FAR
            mov     ax,hi_count     ;retrieve the high count.
            ret
FTimerHi    ENDP
JPtr        DW      J0              ;"switch", starts at JO
            DW      J1              ; inserted in JPtr on first call
parent_seg  DW      0               ;segment of start of parent pgm.
hi_count    DW      0
_FTFunc_TEXT        ENDS

END
```

Listing 5—*Fast-timer QuickBASIC demo program. The function FTime# returns the time in seconds as a double-precision floating point number.*

```
' Simple Demo Program for the routines FTIMER and FTIMCRHI

DECLARE FUNCTION FTime# 0
DECLARE SUB WaitForKey 0
DECLARE FUNCTION FTIMER&          '} These functions must be in
DECLARE FUNCTION FTIMERHI%        ' }     a loaded Quick Librarv
CONST ClockRate = 1193180#        ' IBM PC Timer tick rate, Hz
CONST FullCount = 4294967296#     ' 2^32 -- 4-byte full count
CLS
DO
    PRINT : PRINT "Hit a key to start timer": WaitForKey
    T1# = FTime#
    PRINT USING "  Started at ######.###### sec."; T1#
    PRINT "Hit a key to stop timer": WaitForKey
    T2# = FTime#
    PRINT USING "  Stopped at ######.###### sec. "; T2#;
    PRINT USING "   E 1 a psed:######.######sec "; T2# T1#
  LOOP
END
FUNCTION FTime#
' Function to convert time from Fast Timer subroutine FTimer& into
' a double-precision real value in seconds. Negative values are
' converted to the appropriate unsigned-integer positive values,
' and overflow counts obtained from the function FTimerHi%.
    T1# = FTIMER&                              'Read the timer
    IF T1# < 0 THEN T1# = T1# + FullCount      'Convert val. to +
    T1# = T1# + T2# * FullCount                'Add any overflow
    FTime# = T1# / ClockRate                   'Convert to seconds
END FUNCTION
SUB WaitForKey
' This routine flushes the keyboard buffer, thrn waits for a
' keystroke. An ESC stops the program. Any other normal keystroke
' continues.
    DO:  X$ = INKEY$: LOOP  WHILE  X$ <> ""    'Clean out buffer
    DO:  X$ = INKEY$: LOOP  WHILE X$ = ""      'Wait for first key
    IF X$ = CHR$(27) THEN SYSTEM               'Use ESC key to stop
END  SUB
```

Parent programs can perform many useful purposes. One that I built for a college instructional network runs a sequence of "children" with each child able to control what other child gets to run next, provides a buffer for inter-program data exchanges, and maintains a legally required security record in a campus network environment.

ARE PARENT PROGRAMS PROUD OF THEIR CHILDREN?

I hope you too will find the parent-child program structure a powerful and versatile tool. Perhaps a more developed parent-child environment can provide other useful functions in the realm of virus detection, additional run-time controls such as passwords to individual programs, and so forth. Let me know if you discover any interesting uses, possibilities, or quirks to this method of launching executables.

For those of you with a yen to go beyond the scope of what I presented here, the best source I know for practical information on DOS is "The DOS Programmer's Reference," second edition, by Terry Dettmann and Jim Kyle, published by Que Corporation in 1989. This book contains lots of data on DOS interrupts and the E X EC function.

Brad Thompson holds a Ph.D. from Michigan State University. He works on computer instrument interfaces and instructional program systems at Gustavus Adolphus College, Saint Peter, *Minn., where he is* Scholar *in Residence in chemistry and physics.*

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

High-speed Modem Basics: The Working Hardware

FEATURE ARTICLE

Michael Swartzendruber

Following up on last month's introduction to modem theory and some of the more popular standards, Michael shows how to build your own modem using a chip set from Exar and a DAA from Cermetek.

Last month I explained some of the common standards that relate to modems and also showed what gains in throughput and performance state-of-the-art modems are trying to achieve. The purpose of that discussion was to take some of the mystery away from this very deep pile of jargon. This month I am going to step away from the abstract and get down to some hardware by building the Gemini project. I will also take a close look at the way the Gemini works.

There are two primary vendor technologies that come together to create the core of the Gemini modem: Exar and Cermetek. Exar is an established provider of modem technologies and offers a wide variety of modem chip sets. Cermetek specializes in hybrid assemblies. Among their key offerings are fully integrated "single-socket modems" and Data Access Arrangements (DAA).

A CLOSE LOOK AT THE DAA

The DAA is the main interface between the modem and the telephone company (telco). The DAA's primary purpose is to provide necessary isolation (up to 1500 volts, so your equipment won't damage the telco's equipment and a lightning hit on the telco line won't blow up your end) without distorting or otherwise affecting the analog data stream. The DAA also performs all telco-related functions such as taking the line off hook, detecting ring signals, and doing 2-wire to 4-wire conversion.

Many times DAAs are implemented using discrete components, but I chose the Cermetek hybrid DAA for a few key reasons:

*Simplified circuitry design (replacing several discretes with one component) made the prototype easier to build and the circuit board easier to lay out.

- To simplify any future dealings with regulatory agencies, I chose to include the Cermetek DAA in my design since it meets or exceeds all regulations for a DAA (see last month's issue for a list of these requirements). This saves me the step of having to prove the compliance of the DAA in the Gemini.
- The simplification of the analog circuit (especially in the realm of isolation between the Gemini and the telco) took away my worries of connecting my prototype to the telco (the last thing I needed was to upset those guys).
- To make it easier for others building the Gemini who might have concerns very similar to my own.

THE CERMETEK DAA

Working with the Cermetek 18 17 is a snap. The device runs on a simple 5-volt supply and has two control lines and two signal lines. The control lines consist of Off Hook (high true) and Ring Indicator (low true pulses). The Off Hook pin is an input to the DAA that the modem controls. Taking this line high causes the DAA to indicate to the telco that the phone is "off the hook." The Ring Indicator is an output that will pulse low for the duration of the ring signal from the telco (typically two seconds) and will return to the high state between rings (typically four seconds). The resistor (R15) and the capacitor (C47) that are connected to the Ring Indicator pin of the DAA are used to produce a ring envelope from the Ring Indicator signal.

The DAA also has XMIT and RCV pins for use with analog signals from and to the modem. The XMIT pin is an input to the DAA and accepts the analog data stream from the modem. This pin must he AC coupled to the modem's transmitter. The RCV pin is the converse of the XMIT and is used to forward analog data from the telco to the modem. This pin should also be

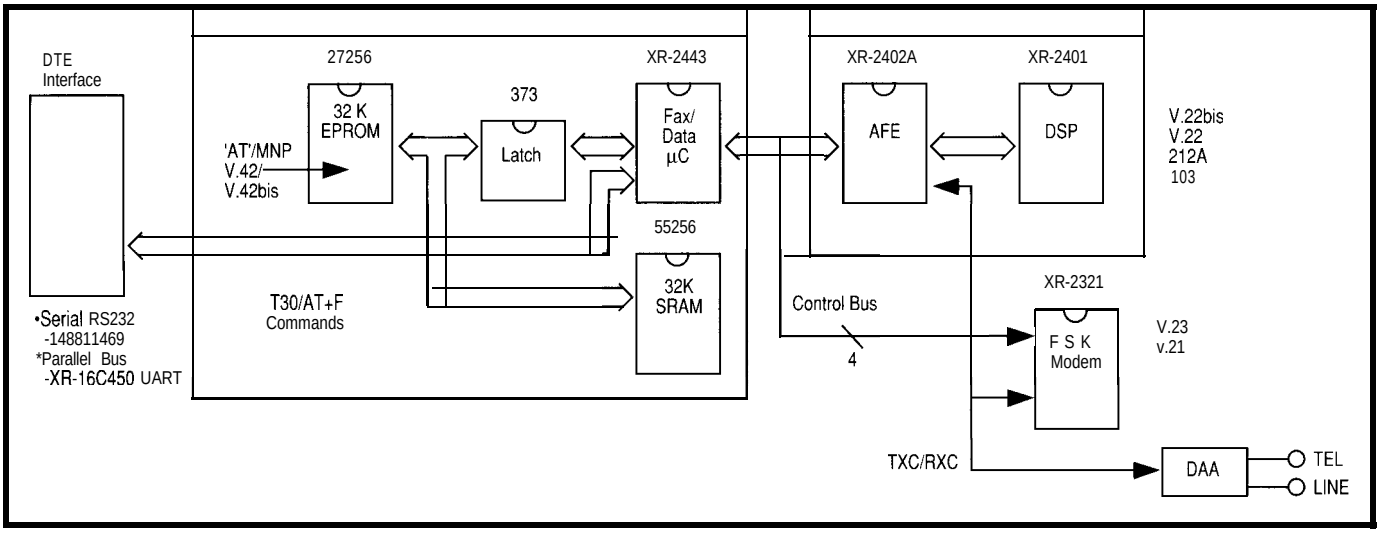


Figure I--The front end *of the XR-2400 chip set is responsible for taking in the analog data stream and extracting digital data, where it gets passed along via the control bus to the* XR-2443 for *V.42bis compression.*

AC coupled to the modem's receiver circuit.

The application note for the DAA makes some very specific recommendations regarding the layout of circuit traces connecting to, or passing close to, the DAA. Most of these recommendations are made to enhance the odds that your design will pass FCC (or other agency) testing. I was not able to comply with them on the prototype (although I did my best), but I did comply with them for the circuit board I designed for the Gemini.

Among the recommendations are some specific to the distance between traces (especially those on the telco side of the device). The tip and ring traces must be separated from each other by 0.1 inch and from all other traces by 0.2 inches. In addition, the tip and ring traces must have a nominal width of 20 mils. Furthermore, these traces should be kept as short as possible and should be laid out to prevent signal coupling to any nearby traces. Other recommendations for power supply bypassing are included, but many of these are not exceptionally different from normally applied design rules for any integrated device. The application note also includes information concerning ways to minimize EMI/RFI injected into the telco from the modem and surge protection. These components are not required, but can be a nice enhancement-especially if you are going to go through EMI/RFI testing.

I added the optional surge protection circuit that they recommend. The surge protection circuit consists of a 250-volt varistor (D1) and series resistors (R6, R26) between the telco network and the surge protector. This small (and completely optional) circuit enhancement can be seen on the Gemini schematic on the page that illustrates the telco connection.

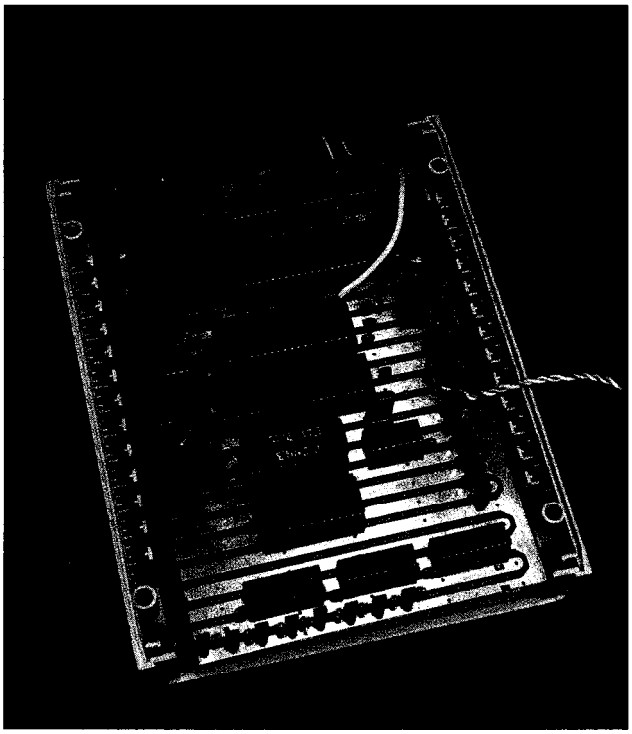


Photo I--Thanks *to a dedicated chip set, the bulk of the modem functions are done in just a few big chips. Gone are the days of lots of touchy analog electronics inside modems.*

THE EXAR CORE

The core of the Gemini modem consists of the XR-2400 chip set and a small amount of supporting glue logic. The chip set contains three primary devices: the XR-2400, XR-2402A, and XR-2443.

The XR-2400 is a DSP-based modem signal processor and provides V.22bis modulation and demodulation functions. The XR-2402A is a high-performance V.22bis AFE (Analog Front End) which provides a whole slew of functionality. Finally, the XR-2443 is a microcontroller with the V.42bis compression algorithm (among other things) built right in. A block diagram of the chip set as it relates to the Gemini is in Figure **1.**

The front end of the modem has the responsibility of accepting the analog data stream and extracting the digital data contained on the carrier wave. It must also accept digital data from the "rest of the modem" and modulate that data onto the carrier for telco network transmission. In short, the front end of the modem provides the modulation and the demodulation services.

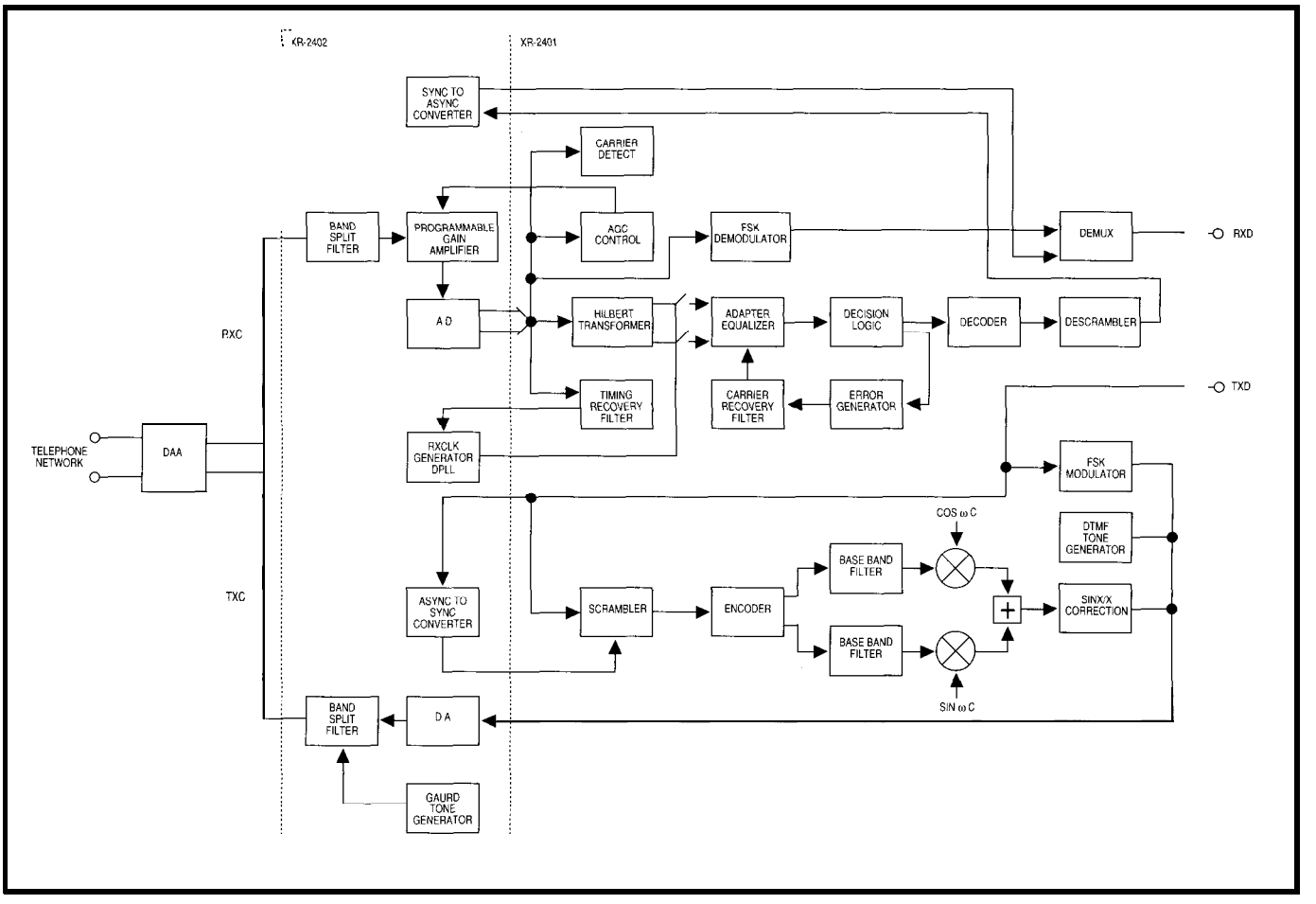
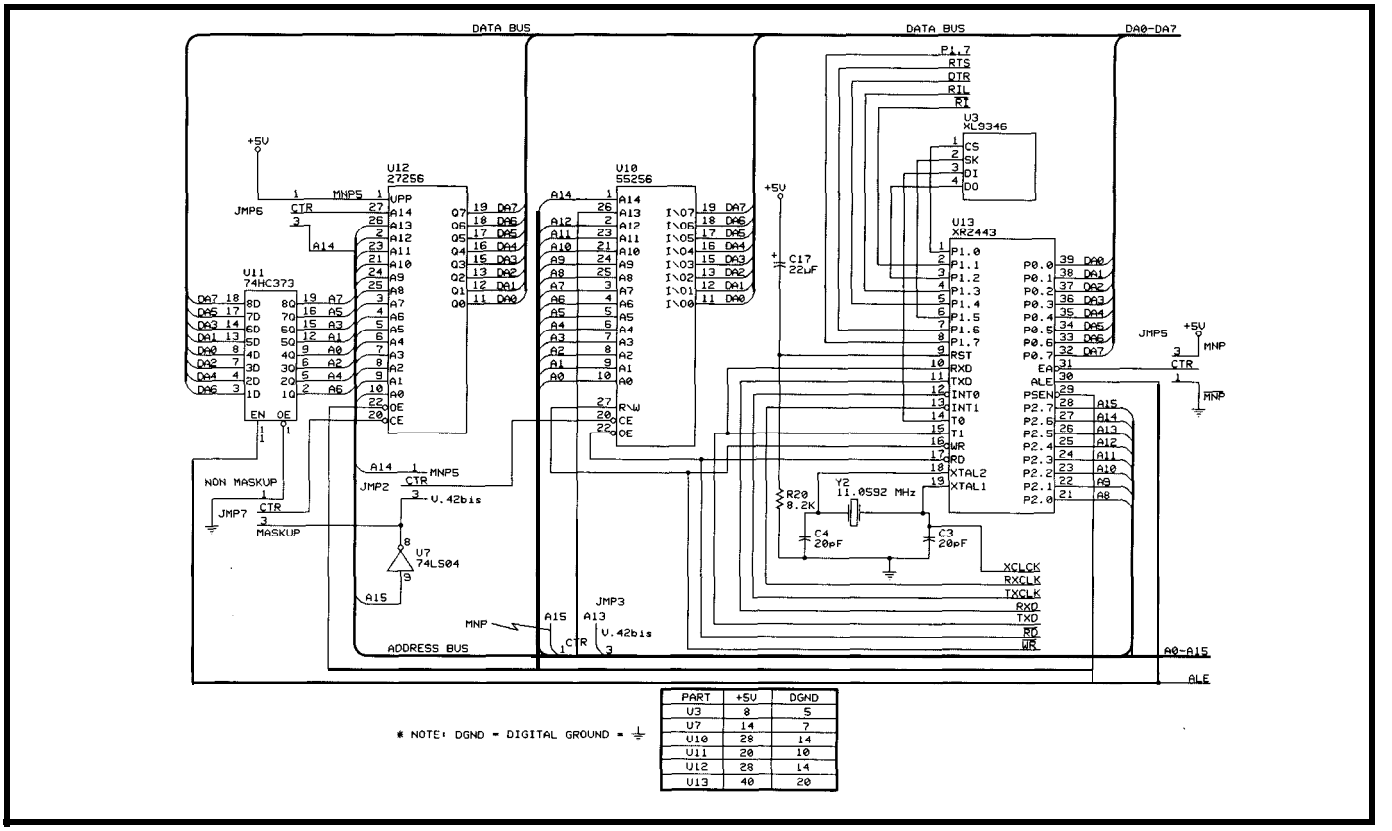


Figure **2**—*The* front *end is composed of the XR-2401 and XR-2402, both of which exchange and process data information before sending it to the XR-2443 and the* DAA.



PART	+5V	DGND
U3	8	5
U7	14	7
U10	28	14
U11	20	10
U12	28	14
U13	40	20

Figure **3a**—*At* the *core of the Gemini modem is the XR2443, a custom-programmed 8031 processor.*

Although the XR-2402A is the AFE used in the Gemini, it is also involved in almost everything that goes on in the modem.

The 2402 contains the A/D and D/A converters. As you might expect, the D/A converter is used to create analog signals for injection into the telco, while the A/D accepts analog signals from the telco for further processing by other devices in the modem. Before the signal from the telco is presented to the ADC on the AFE, it passes through a programmable gain amplifier (PGA). The gain of this amplifier is set by an AGC and control circuitry (used to control the PGA) that resides in the XR-2401.

The 2402 and the 2401 are a tightly knit pair. The output from the ADC on the 2402 is fed into several functional blocks on the 2401. Among these blocks is a timing recovery filter, the output of which feeds back to the 2402 and drives a phase-locked loop whose output is then fed back into the 2401 to control the input switches to the adaptive equalization circuit in the 240 1. To further complement the data reception functions, the output of the descrambler (this generates the CRC) in the 2401 is fed back to a synchronous-to-asynchronous converter in the 2402. The output of this block is fed back into the demultiplexer block of the 240 1. The data transmission (from the modem to the telco) goes through both devices in a similar manner. When data arrives from the modem to be transmitted to the telco, it is fed to the async-to-sync converter in the 2402. The output from this converter is fed into the scrambler (generates the CRC) along with the data that is bound for the telco. A block diagram showing how these two devices are connected is shown in Figure 2.

In addition to serving as the DAA signal gateway, the AFE controls the amplitude of the signal that is used to drive the speaker during call progress monitoring. The 2402 also has status bits that can be read by the host microcontroller to indicate carrier detect and normal energy reception (high and low band). Low-band detection is used to indicate dial, busy, and ring-back tones. The speaker control functions (including mode and volume) reside in a register in the 2402.

The 2402 also serves as the control point between the front end and the microcontroller. The chip is accessed from the microcontroller like a memory address; it has the intelligence to recognize commands that belong to the 2401 and will forward them to that device. The 2402 will also forward status and data information in both directions between the 2401 and the microcontroller. Whenever the 2402 receives data bound for the 2401, it generates an interrupt to the 2401. The ISR in the 2401 locates the information (data or command) and acts on it appropriately.

The 2401 contains a register that determines its mode of operation, which include idle mode, DTMF mode, or, if engaged in a data transfer, which modulation method to use (FSK or DPSK). The 2401 contains a DTMF generator that is controlled by the host microcontroller. The output of the generator is completely dependent on the data written to the 2401 while it is in DTMF mode.

An Unbiased Survey for DOS Developers.

☐ No!

I don't want to find out how I can save a lot of money using ROM-DOS 5 instead of MS-DOS@ in our 80x86 product line. I don't care if ROM-DOS 5 iscompatible with MS-DOS 5 but costs much less. I like spending much more than I have to. It makes me feel like a philanthropist and besides Microsoft@ probably needs the money more than I do anyway.

☐ Yes, I want to know the facts about ROM-DOS 5. Please send me information and a free bootable demo disk to try with my software.

In the U.S.A. Call Toll Free 1-800-221-6630

or fax this coupon to (206) 435-0253.

Name _____

Company _____

Address _____

City _____ State _____ zip _____

Phone _____ Fax _____

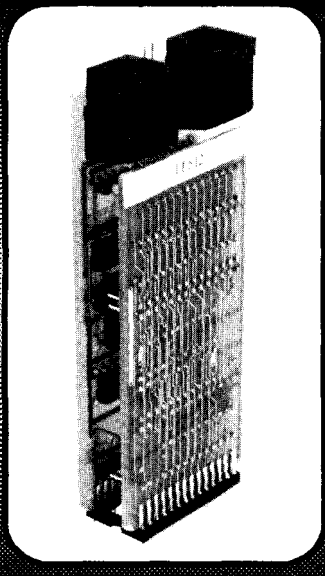
307 N. OLYMPIC, SUITE 201 • ARLINGTON., WA 98223 USA • (206) 435-8086 • FAX: (206) 435-0253

Datalight®

MS-DOS and Microsoft are registered trademarks of Microsoft Corporation

#121

EPROM EMULATORS



$129

AND UP

- Downloads through standard PC compatible printer port
- Includes RJ-11/12 adaptor and 7' modular cable
- Generates RESET and /RESET
- Accepts Binary, Intel HEX, Intel Extended HEX and 'S' files
- Unique Vertical design minimizes mechanical interference and eliminates the possibility of noise cross-talk or transmission-line effects from the use of a cable
- Multiple devices (except EE256) can be daisy-chained for 16/32 bit or multi-bank target systems

EE256 (2764-27256) $129
EE512 (2764-27512) $159
EE1M (2764-27010) $199

MORE ADVANCED 8 and 16bit MODELS AVAILABLE (TO 4Mb)!

10 day MONEY-BACK GUARANTEE
90 day repair/replace warranty

Technical *Solutions*
PO BOX 462101
Garland, TX 75046-2101
Call or FAX for ENGINEERING SPECS.
Voice or FAX: (214) 272-9392

#122

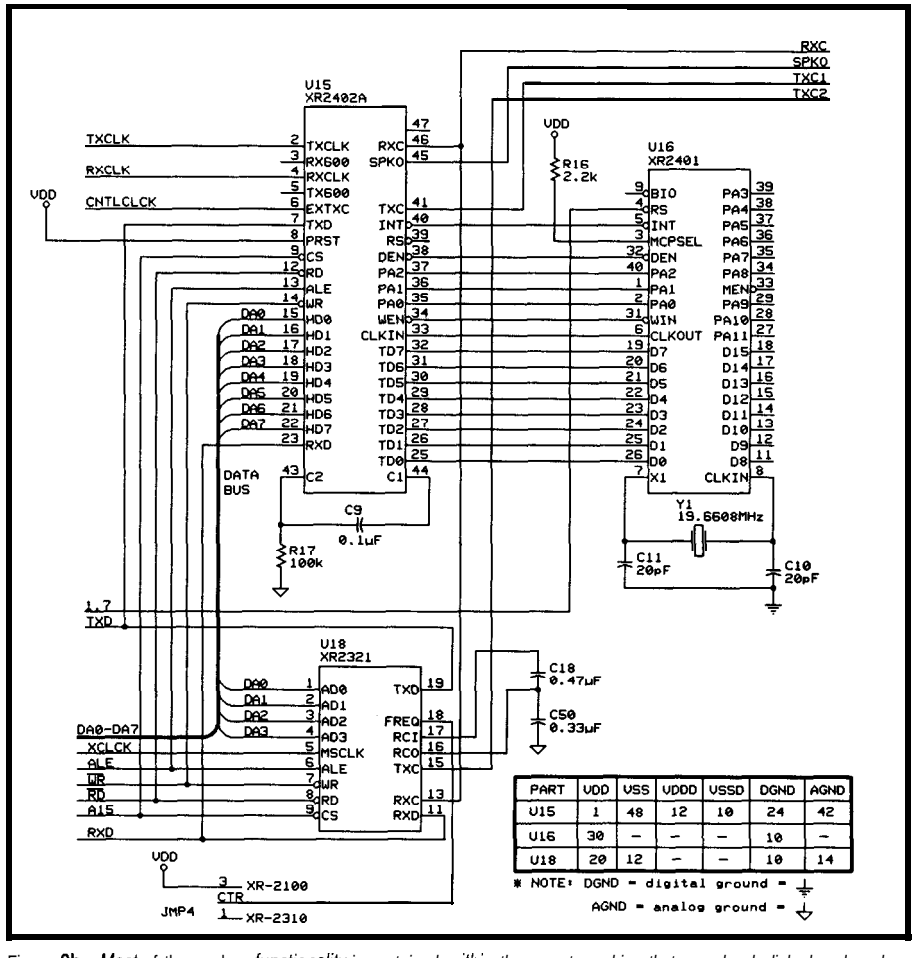


Figure **3b**—*Most* of the modern *functionality* is *contained within three custom chips that are closely linked and work* hand in hand

THE BRAINS

The XR-2443 processor is really just an 8051 that contains a custom-masked ROM designed by Exar. If you look closely at the circuit consisting of U13, U10, U12, U11, and U5, you can see "just another 8031" single-board computer. The 2443 ROM provides command control over the 240 1 and the 2402 by translating AT and MNP commands into commands specific to the operation of the chips. The actual commands (AT and MNP) reside in the EEPROM installed in the modem. This makes evolution of the AT and MNP command sets possible without requiring changes to the 2443 ROM. The firmware supplied by Exar provides full AT and MNP command set compatibility.

For those who want to go the extra mile, Exar also provides a fully decoded memory map of the EPROM and the RAM, the address ranges used by the Exar-supplied firmware, all the interrupts used in their firmware, and a full list of the entry points used in their firmware along their hexadecimal addresses. They provide this detailed level of documentation to simplify the effort required in creating value-added software for their core technology.

In summary, Exar provides everything you need to make an off-the-shelf modem through a straight application of their technology. They also give you all the information necessary to extend their firmware to create a superset of what they bring to the table. The biggest advantage to using the 2443, though, is the fact that it contains pretested code that implements MNP levels 2-5 as well as V.42 and V.42bis.

DESIGNING WITH THE EXAR CHIP SET

The data and applications notes for the Exar chip set make some very specific recommendations for layout of components and traces. One of the

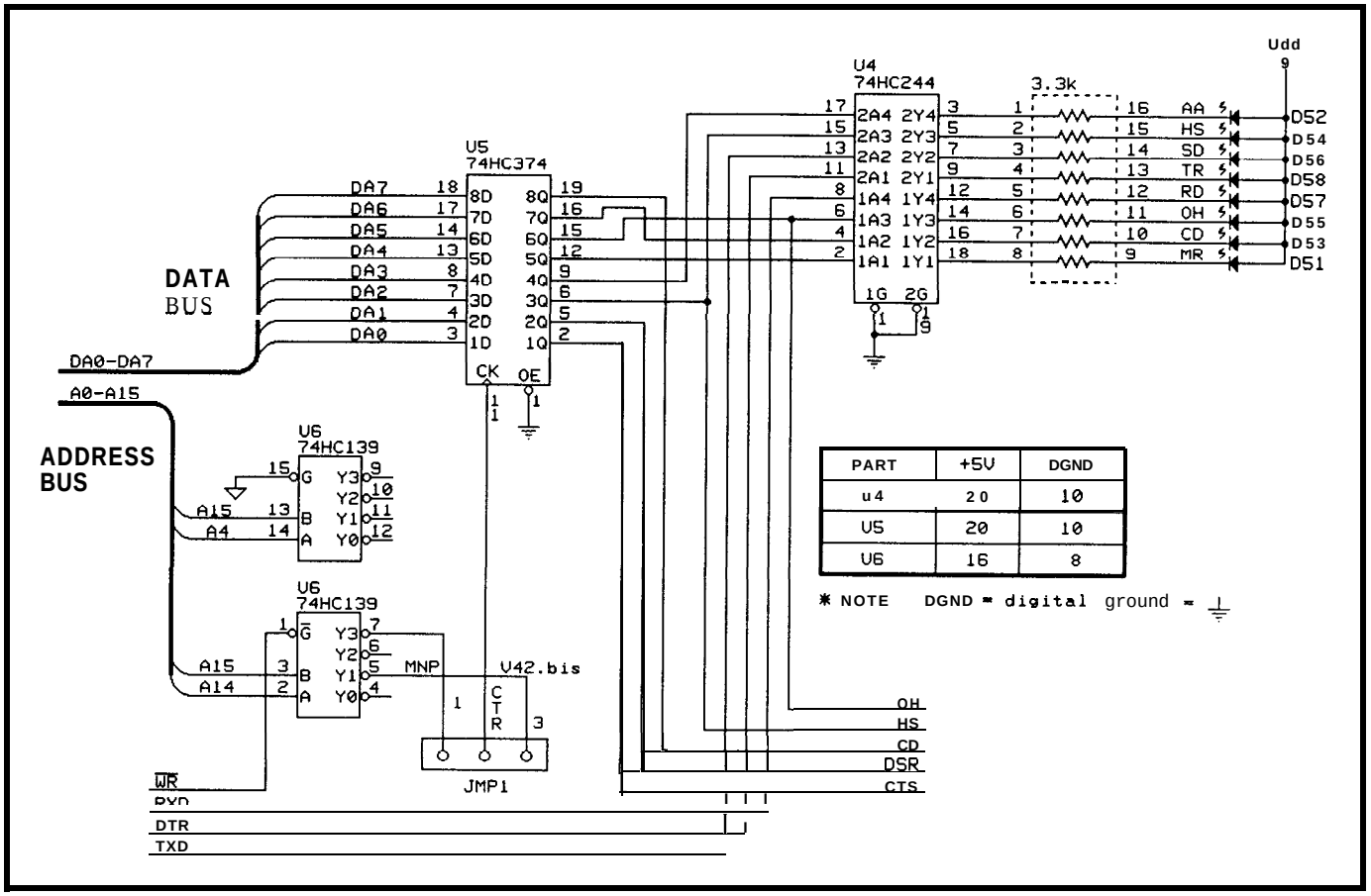


Figure **3c**—*The 74HC374 latch holds the status* bits for *both the serial port and the front-panel LEDs. The 74HC244 simply provides drive current for the LEDs*

points they emphasize is that digital ground and analog ground should be kept separate up to the point where they enter the power supply. Obviously, the better your ground plane, the better off this circuit will be. In addition, they stipulate that the 2402A should be as close to the DAA as possible to keep these traces short. The traces carrying the analog KX and TX signals should be kept as far away from digital signals as possible. They also recommend lots of bypass caps, which should be mounted as close to the power pins as is possible.

WHAT ABOUT THE REST?

In order to make the core chips happy, the Gemini also includes a handful of support logic. The final schematic is shown in Figure 3.

The RS-232 interface level shifting is accomplished with ordinary 1488s and 1489s. I could have used MAX232s, but since I needed negative supply voltages for U15 and U19 anyway, I was able to save some money and board space.

U3 (an XL-93C46) is a 1024-bit serial EEPROM that is used to store interesting things like user-defined power-up settings, two additional user-definable settings that can be recalled and made active, and up to three phone numbers.

U6 acts as little more than an address decoder. It has two outputs that are enabled on converse states of A14 and A15. Jumper JMP1 selects which of these strobe signals will be used to clock U5. When U5 is clocked, it forwards the data on the data bus to its output side. Therefore, U5 is an output port in relation to the XR-2443 microcontroller. The data on the output side of U5 is used as an input to U4, which drives status LEDs. The outputs of U5 also go to the D-type connector (the RS-232 port) to give status information to the PC.

Just for good measure, I also included the XR-2321 in the Gemini. This single-chip modem is not required at all for the Gemini to operate in most of its modes. I threw it in to add V.21 (300 bps) and V.23 (1200 bps, half duplex) support to the modem. While neither standard is used much in the U.S., European users might find them handy.

THE LONG AND THE SHORT OF IT

So there you have it. As you can see, there is not much magic going on inside a modem these days. With the increasing amount of integration—especially analog integration such as that found in the 2401, the 2402, and the Cermetek DAA-products of increasing complexity can be created from readily available components. Implementing precooked microcontrollers (those with masked ROMs such as the 2443) is another way to get more products to market in short order because they allow you to leverage off the work already done and packaged in the component.

But, after exploring some of the inner workings of modems by building the Gemini, I wonder if maybe the term "modem" has outlived its usefulness. Sure, these devices still perform basic modulation and de-

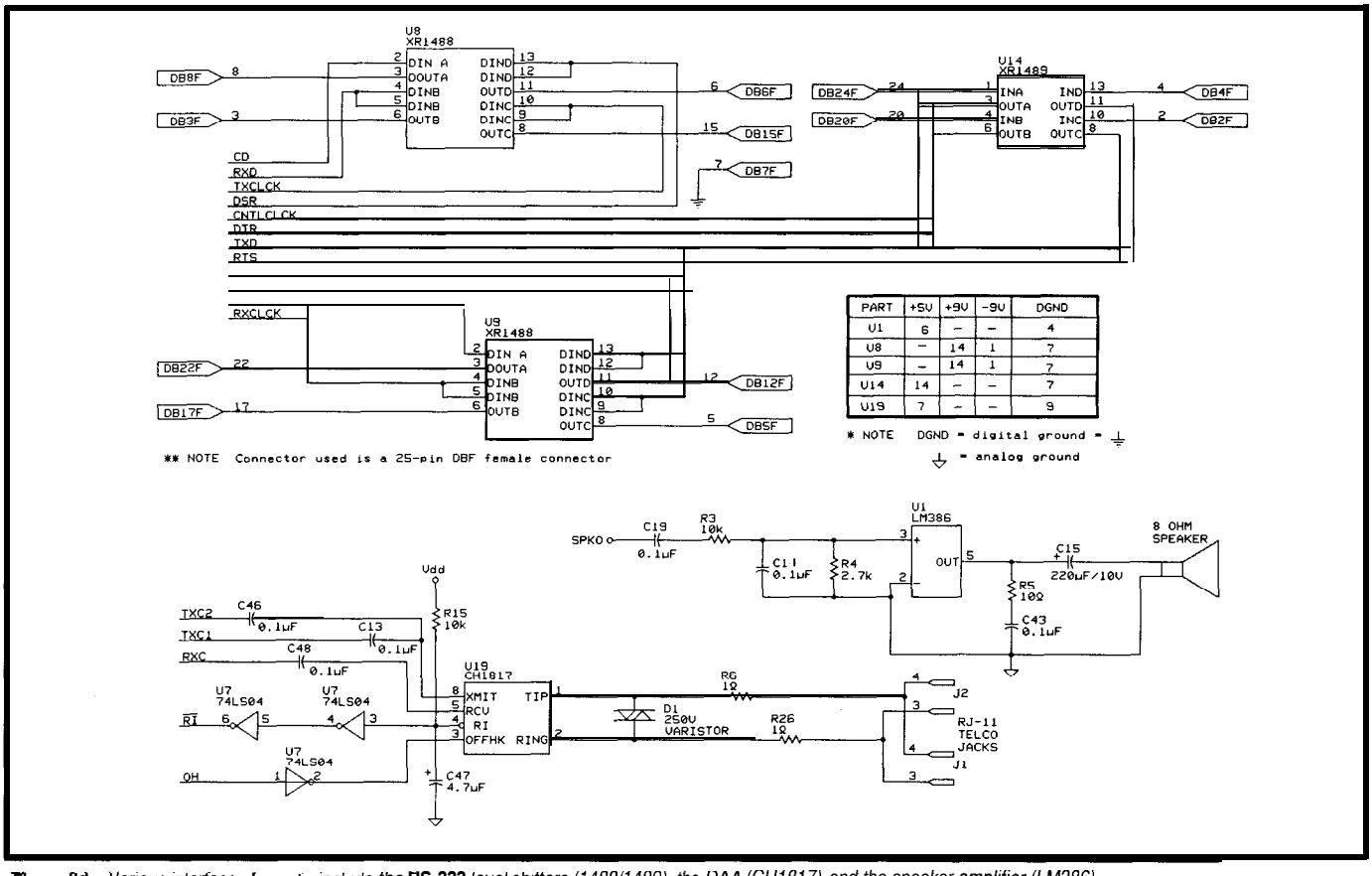
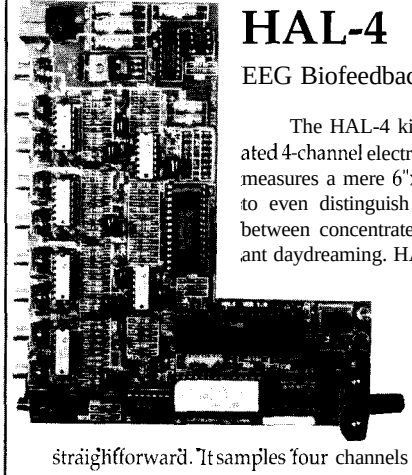


Figure 3d—*Various interface* **elements** *include the* **RS-232** *level shifters (1488/1489), the DAA (CH1817), and the speaker amplifier (LM386)*

CIRCUIT CELLAR KITS



HAL-4

EEG Biofeedback Brainwave Analyzer

The HAL-4 kit is a complete battery-operated 4-channel electroencephalograph (EEG) which measures a mere 6"x7". HAL is sensitive enough to even distinguish different conscious states—between concentrated mental activity and pleasant daydreaming. HAL gathers all relevent 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 straighforward. 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.....**$179.00** plus shipping

• 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 clams are made for this device, and it should not be used for medical diagnostic purposes Furthermore, safe use requires that HAL be battery operated only!

Sonar Ranging Experimenter's Kit

Targeting ◆ Ranging ◆ Machine Vision

The Circuit Cellar TI01 Ultrasonic Sonar Ranger is based on the sonar ranging circuitry from the Polaroid SX-70 camera system. The TI01 and the original SX-70 have similar performance but the TI01 Sonar Ranger requires far less support circuitry and interface hardware.

The TI01 ranging kit consists of a Polaroid 50-kHz, 300-V electrostatic transducer and ultrasonic ranging electronics board made by Texas Instruments. Sonar Ranger measures ranges of 1.2 inches to 35 feet, has a TTL output when operated on 5V, and easily connects to a parallel printer port.

TI01 Sonar Ranger kit. ..... **.$79.00** plus shipping

CHECK OUT THE NEW CIRCUIT CELLAR

HOME CONTROL SYSTEM

◆ Expandable Network
◆ Trainable IR Interface
◆ Digital and Analog I/O
◆ Remote Displays
◆ X-10 Interface

Call and ask about the HCS *II*

To order the products shown or to receive a catalog,

call: (203) 875-2751 or fax: (203) 872-2204

Circuit Cellar Kits • 4 Park Street • Suite 12 • Vernon, CT 06066

#123

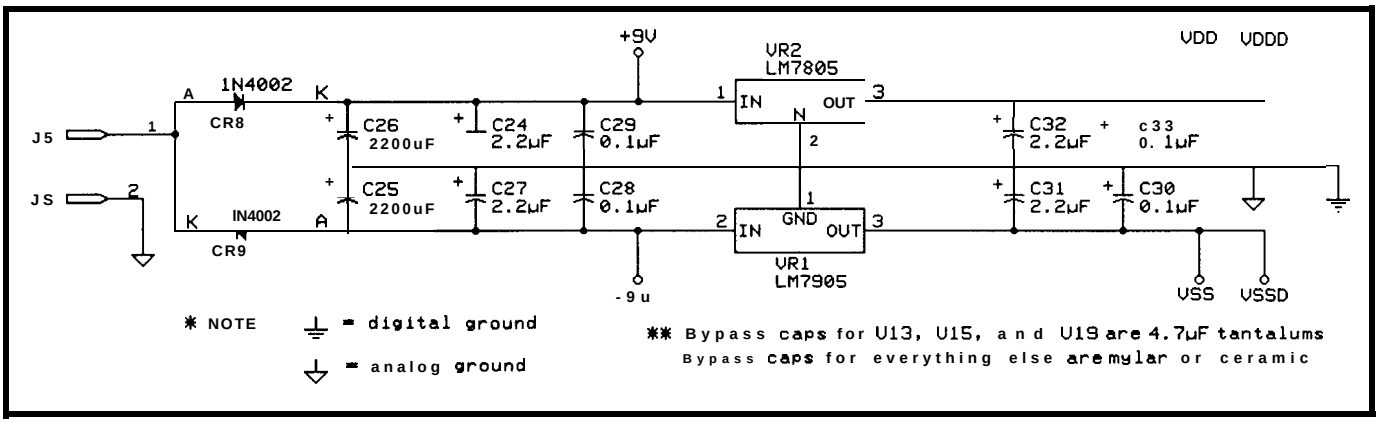


Figure 3e—*The power supply section* ***produces*** *±9 V and ±5 V. Analog* ***and*** *digital power and analog and digital ground are tied fogefher at single points at the power supply.*

modulation, but when you stop to consider everything else they do, I think elevating their title to a "communications coprocessor" wouldn't be out of line. From now on, I think I'll have a little more respect for the work that goes into making those LEDs blink on my modem as I talk to my friends in Cyberspace. ❑

Special thanks to the people at Exar for supporting me during this project. Their help was top notch.

CONTACT

Exar Corp.
P.O. Box 49007
2222 Qume Dr.
San Jose, CA 95161-9007
(408) 434-6400

Cermetek Microelectronics, Inc.
1308 Borregas Ave.
Sunnyvale, CA 94089
(408) 752-5000
Fax: (408) 752-5004

Michael Swartzendruber is an engineer with experience in network and communications design and Windows and Macintosh programming. He is also a Technical Editor for the Computer Applications *Journal.*

IRS

407 Very Useful
408 Moderately Useful
409 Not Useful

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: (919) 667-2100
Fax: (919) 667-0487

DEPARTMENTS

46 Firmware Furnace

56 From the Bench

62 Silicon Update

70 Embedded Techniques

78 Patent Talk

85 ConnecTime

FIRMWARE FURNACE

Ed Nisley

Memories Are Made of This: The '386SX Project Goes Nonvolatile

Need to remember something? Ed adds some EPROM and EEPROM to the Firmware Development Board and shows how to add your own routines to the existing BIOS.

Back in February I pointed out that "doing firmware" for the '386SX project didn't require an EPROM because the BIOS has all the code you need to boot a program from disk. That approach has served us well, but some applications just cry out for a smidge of nonrotating storage.

Compared to the confines of an 8031 system, the megabyte of address space in a PC (we'll ignore protected mode's 16 MB or more for now) seems limitless. As it turns out, though, there isn't that much space left for our EPROM, and a whole 64K bytes of free space may be hard to come by.

I'll start by reviewing the PC's memory layout, explore ISA bus memory timing, then describe the circuitry needed to add an EPROM or EEPROM to the Firmware Development Board. With the hardware in place, a little firmware will let us load a program into the system so it becomes a part of the BIOS and runs whenever the PC starts up.

WHERE DOES MEMORY COME FROM?

PC's memory organization pays homage to The Original IBM PC and

its 8088 CPU. The best we can do at this late date is quibble about the details because the Big Picture is thickly encrusted with compatibility barnacles and impervious to change. Figure 1 shows the major divisions in the first megabyte of storage.

The first and largest chunk is the (in)famous 640K-byte block devoted to user programs and data. If your real-mode application requires contiguous RAM, this is as good as it gets. While there are ways to extend this block, none are particularly attractive.

The video-RAM buffers occupy 128K bytes starting at A0000. The old CGA card freed the space below B8000 to get 736K bytes of contiguous user RAM, but the VGA's 128K bytes of buffer renders that trick essentially useless. Although a VGA in CGA mode can release the space below B8000, simply changing back to VGA mode will lock up the system.

Of course, if your application doesn't use video, you can yank the card and devote the address space to whatever you'd like. The Bad News is you can never, ever, install a video card. That seems a shame given the utility of built-in, BIOS-supported, standard video, but it's your call..

Figure I--The *first* megabyte of PC memory serves many *different functions* defined both *by the BIOS* and by convention. You should think *long* and hard *about* compatibility problems *before you devote a chunk* of address *space to a* nonstandard use!

Address Ranae	Size	Type	Function
00000–9FFFF	640K	RAM	Programs & data
A0000-BFFFF	128K	Video RAM	Video buffers
C0000–C7FFF	32K	Video ROM	BIOS Extension
C8000–CFFFF	32K	ROM or RAM	BIOS Extension
D0000-DFFFF	64K	ROM or RAM	BIOS Extension
E0000-EFFFF	64K	ROM or RAM	BIOS Extension
F0000-FFFFF	64K	ROM	System BIOS

The system board BIOS may occupy either 64K bytes starting at F0000 or 128K bytes starting at E0000. Early PCs sported an empty EPROM socket or two on the system board, but relentless cost reduction eliminated that nicety in short order. If your PC doesn't have BIOS code at E0000 you can use that address space for your own purposes.

Some I/O cards-notably video adapters, exotic hard disk controllers, and network adapters-include EPROMs that modify, extend, or replace some system board BIOS functions. The BIOS scans the 128K bytes of address space between C0000 and E0000 to find these EPROMs and execute their startup code during power-on initialization. Fortunately for us this entire process is well defined and essentially magic free.

I/O cards may include ordinary RAM in addition to the EPROM, but the cramped address space restricts RAM to small buffers and scratchpads. The history of LIMS EMS cards shows what can be done when you're desperate for more RAM: bank-switching 32 mega-bytes of storage through a 64-kilobyte slot isn't pretty, but it was the only way to do it when it needed doing.

Embedded applications have more freedom to chop up the lower 640K bytes, so, when we need huge buffers, they need not fit between the video buffers and the system BIOS. Some applications can make use of the vast extent of RAM beyond the lower megabyte, even when running in real mode, and we'll look at that later.

The Firmware Development Card will include either 8K bytes or 32K

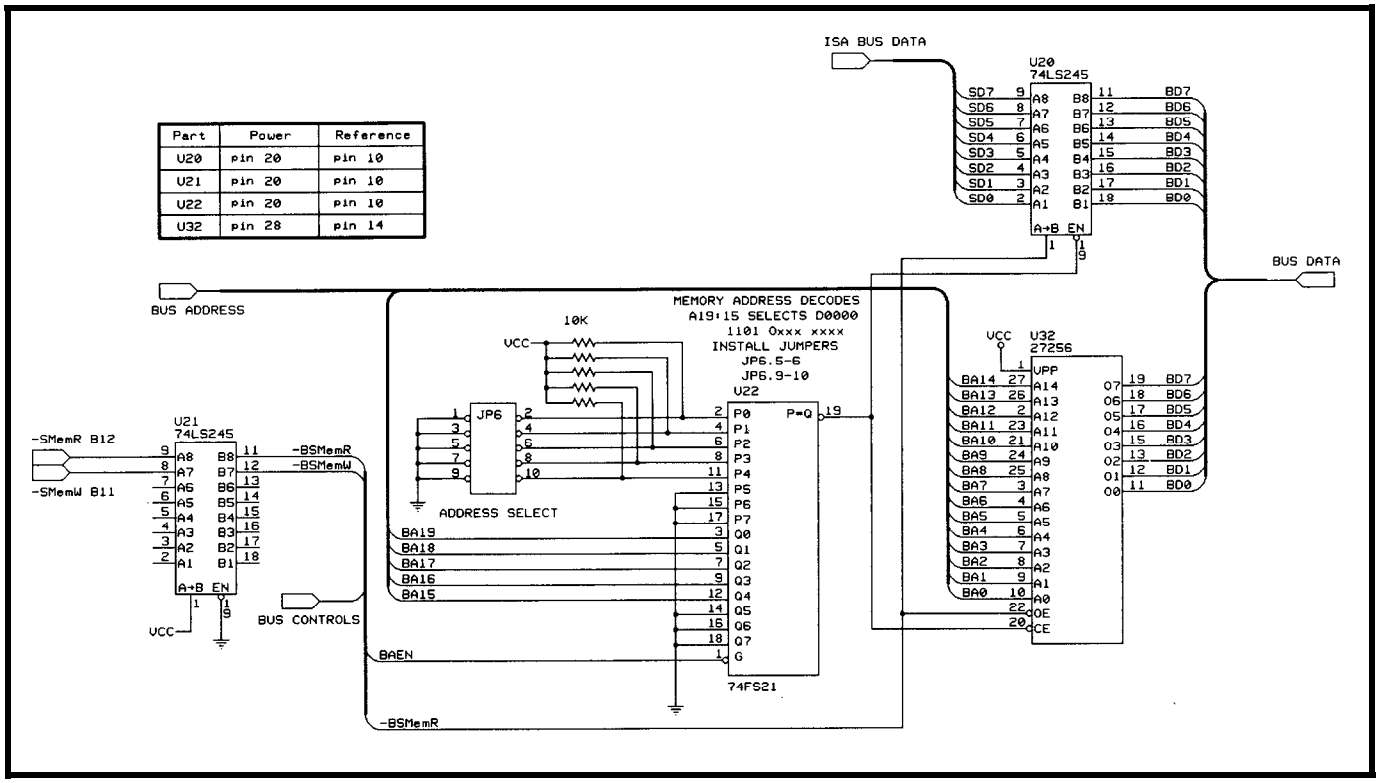


Figure *2-Adding an EPROM to the Firmware Development Board involves just some buffers and a decoder.*

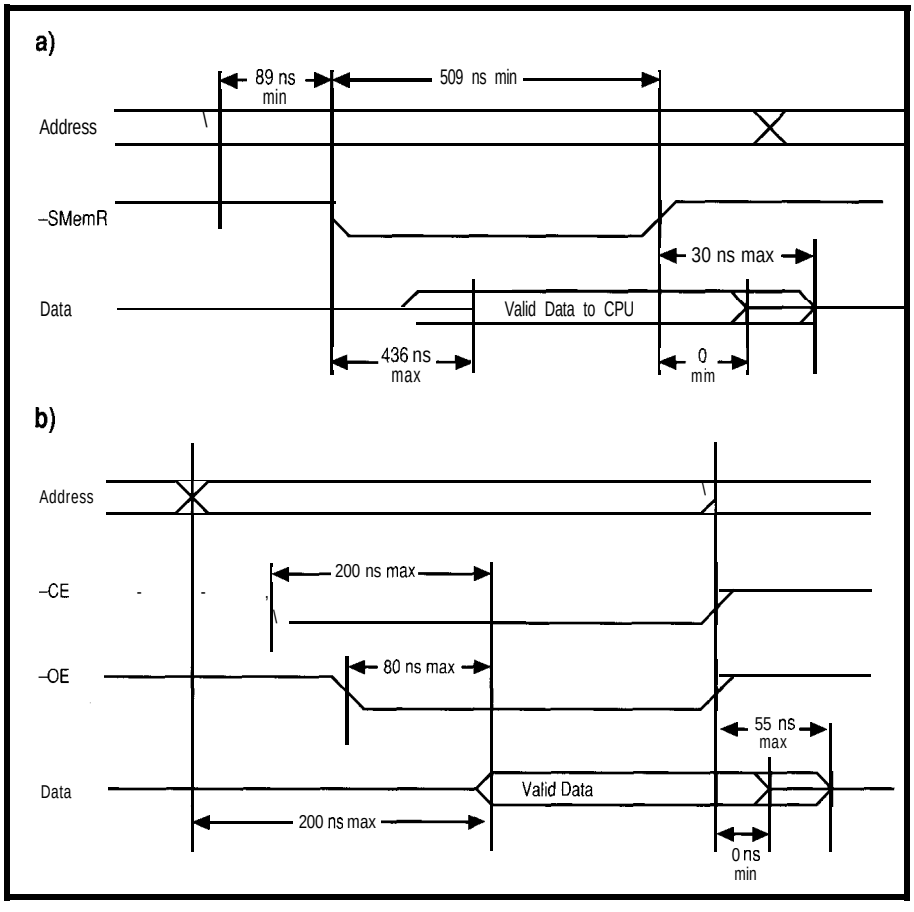


Figure 3-a) Signals *and timing involved in an 8-bit ISA bus memory read access are compatible with cards designed for the original IBM PC and are painfully slow by contemporary standards. b) Timings show the read cycle for a 200-ns 28C64A* EEPROM, but they *are typical of EPROMs as well. The bus buffers shown in the schematic truncate the rather long maximum data hold* time.

bytes of storage mapped into the PC's address space between C8000 and CFFFF. While the decoding circuitry allows you to plunk it at any other address in the first megabyte, I think you can see why the actual choices are rather restricted.

Before we link up with the BIOS, though, we must get the memory running.. .

CONFRONTING THE ISA SLOWS

Not only do ISA bus barnacles determine the memory layout, they also set the memory access time. You've probably noticed that high-end PCs now sport "local bus" connectors for memory and high-bandwidth I/O.. after this column, you'll know why.

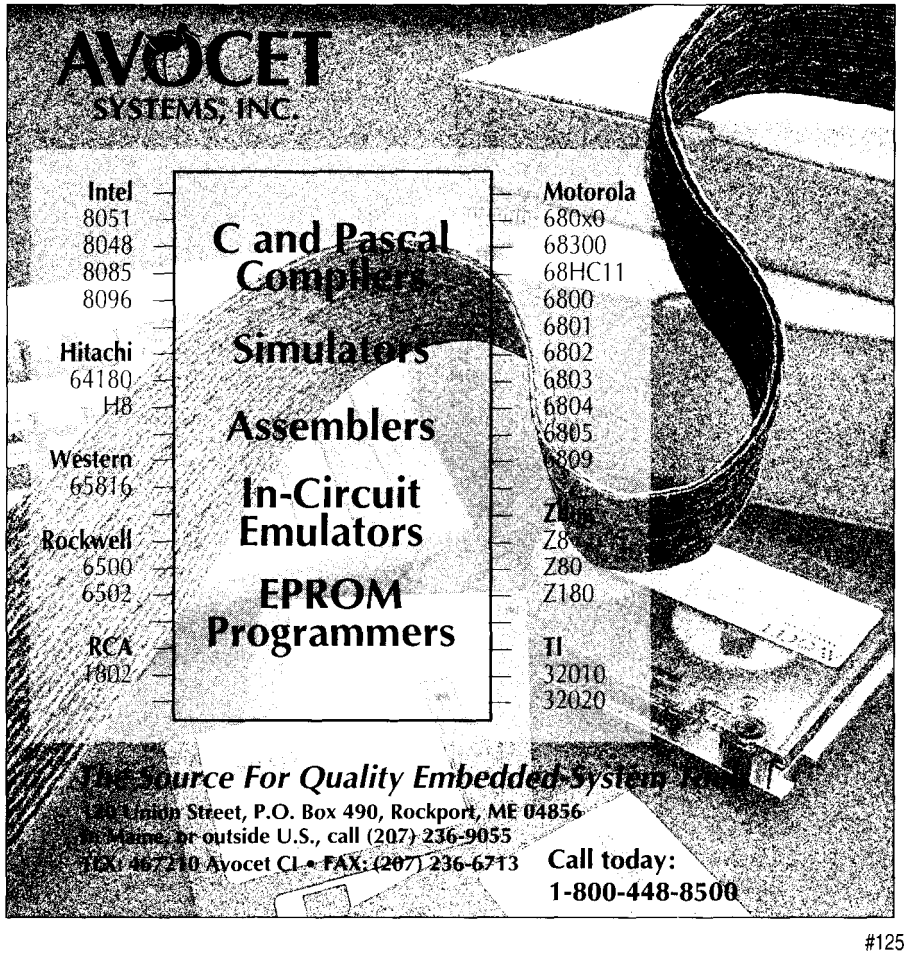
Figure 2 shows the falling-off-a-log simple circuitry needed to put an EPROM on the Firmware Development Board. The 'LS245 buffers isolate the bus data and control lines, the F521 activates the EPROM's -CE input when the CPU reads or writes a byte in the desired address range, and the EPROM holds the data.

The tradeoff for this simplicity is, as usual, performance. Because the EPROM is only eight bits wide, the CPU must fetch one byte at a time. The ISA bus defaults to the same achingly slow six-cycle 720 ns access for memory as it does for I/O, with the results shown in Photo 1.

Figure 3a shows ISA bus timings for an 8-bit memory access and Figure 3b show typical access times for a 200-ns EPROM or EEPROM. It's easy to see that the data will be ready long before the end of the bus cycle, so I won't go through the same analysis as I did with the I/O ports.

It seems obvious that we need to reduce the bus cycle time, but appearances can be deceiving.

There are two ways to speed up ISA bus memory accesses. You can use a pair of EPROMs (or a pricey 16-bit



#125

Listing 1--This test loop uses a single REP LODSB instruction to read the entire 32K EPROM address space. The elapsed loop time, measured either by stopwatch or oscilloscope, gives a good indication of how fast the ISA bus can handle memory accesses.

```
RAMSize = 0x8000;                /* repeat for 32K block */
RAMSeg = NV-SEGMENT;
Counter = 0;
while (!chkch()){
     outp(SYNC_ADDR,0x01);       /* scope sync */
     asm {
         MOV   CX,RAMSize        set up count
         PUSH  DS
         MOV   AX,RAMSeg         set up address
         MOV   DS,AX
         XOR   SI,SI
         REP
         LODSB
         POP   DS
     }
     outp(SYNC_ADDR,0x00);
     /* show count on FDB LEDs */
     outpw(LED_ADDR,~ByteToSegs(Counter));
     ++Counter;
}
```

part) to provide 16-bit accesses and add circuitry to activate –MEMCS16 to get three-cycle memory accesses. If that isn't good enough, you can attempt to activate SRDY fast enough to get a two-cycle access, but Solari's book is replete with cautions and compatibility hazards.

It turns out, though, that there is a better way that not only makes the bus access time irrelevant, but doesn't involve any extra hardware. The "ROM shadowing" feature of most current system boards copies the EPROM contents into RAM, disables the EPROM, maps the RAM to the EPROM's address range, and write-protects it. Poof: fast EPROM made possible by cheap RAM!

Contrary to popular opinion, this shadowing has nothing to do with the magic made possible by the 386 CPU's protected-mode memory management hardware. It's entirely a function of the system board LSI hardware, so your programs continue to run in real mode. The relocation hardware may chop up the memory above the first megabyte enough that protected mode operating systems have trouble using it, but that's a completely separate design issue.

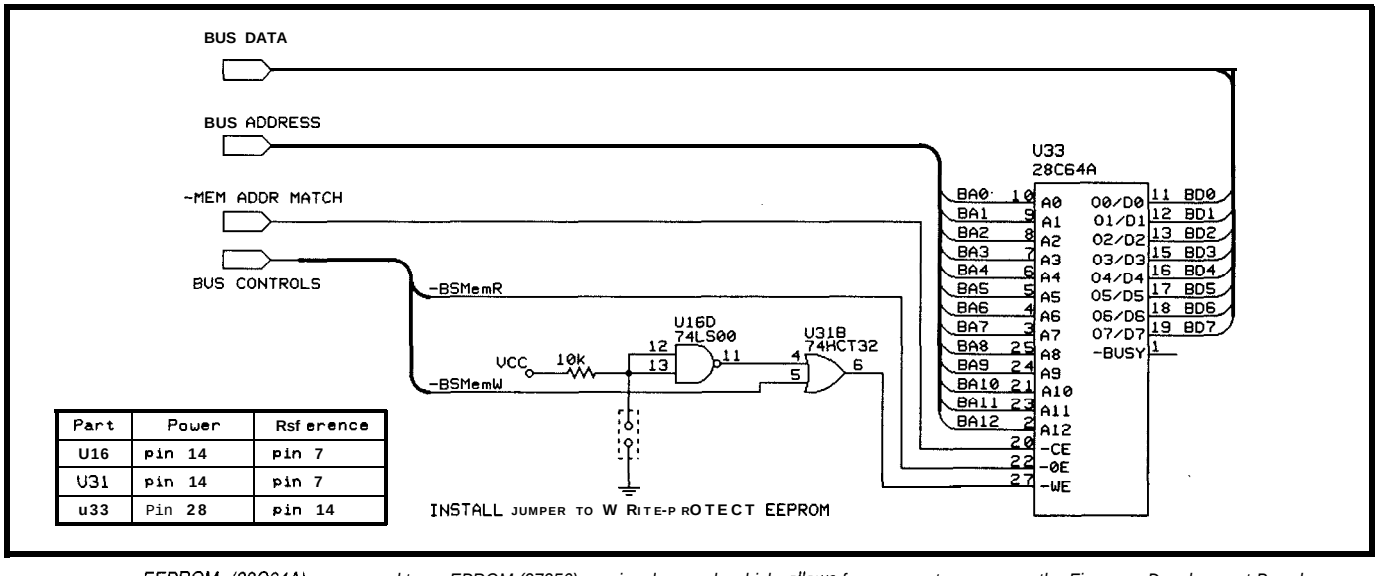
The BIOS does all the copying and remapping during the power-on reset sequence, so by the time your code gets control, the EPROM is out of the picture. The system board circuitry runs *much* faster than the ISA bus, so operations that depend on EPROM data get a corresponding boost.

Listing 1 shows a section of MEMTEST.C that reads the 32K block of storage with REP LODSB in a tight loop. It takes 33 ms per loop with shadowing disabled, or about 960 ns per LODSB. Enabling ROM shadowing cuts the loop to 7.3 ms, or about 210 ns per LODSB. Simple division: ROM shadowing reduces the elapsed loop time by about 80%.

Even if you could get a "no-wait-state" ISA bus interface running, it would still take 240 ns just for the bus access. To judge from my logic analyzer traces, each REP LODSB adds two bus cycles to the minimum required to fetch the data, so even an optimized interface would take 480 ns per byte.

There you have it: a four-chip, warp speed, no hassle EPROM storage system for your embedded system. Ain't science grand?

The MEMTEST.C program has the test routines I used to get the memory working, including a hex file with 32K bytes of pseudorandom numbers from Micro-C's rand() function. Burn PSR32K.HEX into a 27256 EPROM and run MEMTEST to read and verify it... that should give you confidence that your circuit works!



Part	Power	Reference
U16	pin 14	pin 7
U31	pin 14	pin 7
u33	Pin 28	pin 14

Figure 4-Using an EEPROM (28C64A) as opposed to an EPROM (27256) requires less park, which allows for some extra space on the Firmware Development Board.

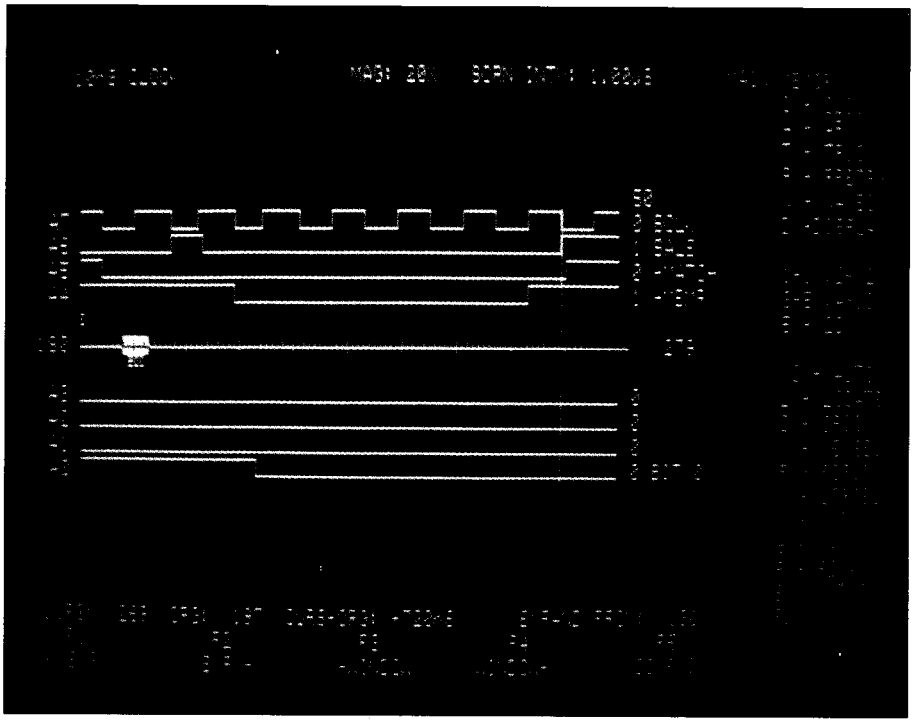


Photo 1--The logic *analyzer* **trace** *shows what happens when the CPU reads a single EPROM byte. The access begins with the rising edge of BALE and ends six BCLK cycles later when –SMemR goes high.*

WRITING TO EEPROM

There are times, however, when an EPROM is not the right hammer for the job. Whether you have frequent code changes, need nonvolatile data logging storage, or just don't want to hassle with an EPROM programmer, an EEPROM may solve your problem.

EEPROMs come in several different flavors, but for our purposes they're all pretty much alike. I'll use the Microchip 28C64A 8K-byte EEPROM as an example because it's readily available from the usual mail-order sources. Feel free to use something else, but remember the address space limits before you spring for a megabyte part!

Figure 4 shows the changes needed to put a 28C64A in place of the 27256 EPROM. Because the EEPROM has only 8K bytes, the CPU will see four identical copies in the 32K-byte address range decoded by the F52 1 comparator. You can add SA14 and SA13 to the comparator or just ignore the ghosts, which is what I did in the code for this column.

As shown in Figure 3b, reading an EEPROM is just like reading an EPROM: the data is ready in plenty of time. There are no special tricks needed to get data from the part.

Writing, on the other hand, is more complex because the EEPROM requires up to 1 ms (that's 1000 microseconds, one million nanoseconds, or about 63 *miles* on Admiral Hopper's scale) to erase and reprogram each byte. The 28C64A, as with all useful EEPROMs, has internal latch and timing circuitry to relieve the CPU of the details, but it cannot accept a new byte until the previous write cycle is completed.

If you have other things to do, you can ignore the EEPROM for at least a millisecond after each write; the timers on the Firmware Development Board are ideal for measuring this kind of delay. However, the 28C64A includes a polling mode that signals

Listing 2—*Writing a byte info the 28C64A is easy, buf you must then poll the chip until the write cycle is complete, which can fake up to one millisecond. The code must also include error handling for timeouts and defective parts. Note that this will work perfectly wifh a standard static RAM in place of the 28C64A, which is a good way to debug the code without using up the EEPROM's limited number of write cycles.*

```
for (RAMAddr = 0x0000: RAMAddr != 0x2000; ++RAMAddr){
    PRSData = peek(PRSSegment,RAMAddr);
    outp(SYNC_ADDR,0x01);                    /* mark the write */
    poke(NV_SEGMENT,RAMAddr,PRSData);        /* start it */
    if (SetAlarm(0x0000,0x2710,&Alarm)){/* error after 10 ms */
        putstr("Cannot set BIOS timeout!\n");
        break:
    }
    do {
        TestData = peek(NV_SEGMENT,RAMAddr);
    } while ((0x0080 & (PRSData ^ TestData)) && !Alarm);
    if (Alarm) {
        printf("Timout programming %04x to %02x, is %02x\n",
            RAMAddr,PRSData,TestData);
        if (MAX-ERRORS < ++ErrorCounter){
            putstr("Too many errors, giving up\n");
            break;
        }
    }
    TestData = peek(NV_SEGMENT,RAMAddr);
    if (TestData != PRSData){
        printf("Cannot program %04x to %02x, is %02x\n",
            RAMAddr,PRSData,TestData);
        if (MAX-ERRORS < ++ErrorCounter){
            putstr("Too many errors, giving up\n");
            break:
        }
    }
    if (CancelAlarm(0x0000,0x2710,&Alarm)){
        putstr("Cannot cancel BIOS timeout!\n");
        break;
    }
    outp(SYNC_ADDR,0x00);
    outpw(LED_ADDR,~ByteToSegs(HI(RAMAddr)));
}
```

Offset	Contents	Definition
0000	55	First signature byte
0001	AA	Second signature byte
0002	xx	Overall length in 512-byte units
0003-4	EB 01	JMP SHORT $+3 (to 0006)
0005	ss	Checksum
0006..	code	BIOS extension code

Figure 5— *The BIOS examines the start of each 2K block between C0000 and E0000 for BIOS extension code. If the first two bytes contain a valid signature and the block checksums to zero, the BIOS executes a FAR CALL to offset 0003 so the extension can initialize itself. Only the signature, length byte, and valid code at offset 0003 are required; the structure of the code block is not specified. The diskette EEPROM loader described in this* column *assumes that the checksum is located at offset 0005.*

when the write cycle is finished: if you read the address you just wrote, the chip inverts bit 7 while it's busy. Bits O-6 are undefined, so you must mask them off before comparing the bytes.

Typical writes are much faster than 1 ms, so you can save a considerable amount of time by monitoring the status. M EMT EST's EEPKOM programming loop takes about four seconds to write all SK bytes, so the average time is well under 500 µs.

Listing 2 shows how this works. After writing the byte using Micro-C's po ke () function, the code sets up a IO-ms timeout and enters a loop waiting for bit 7 to match the poke ()'d bit. After the exclusive OR returns zero (or after the timeout), the code checks for errors before continuing with the next byte.

Note that you need one additional read after the 28C64A reports that it's not busy, as the data may not be valid immediately after bit 7 changes.

The 28C64A EEPKOM has much the same pinout as an 8K-byte static RAM and I *strongly* recommend that you use a RAM instead of an EEPKOM until you are entirely sure your hardware and code are up to par. The 28C64A specs say it is good for 10^4 write cycles, but that works out to perhaps ten seconds at full throttle.

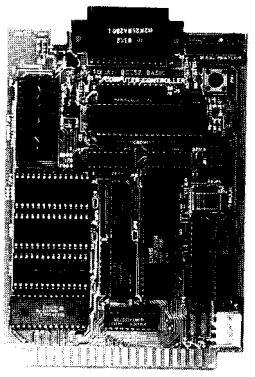
For example, one of M EMT EST's routines is a tight write loop so you can look at the control signals on a scope. This will slaughter an innocent 28C64A in short order, so be careful what's in the socket.

Pin 26 is +CE2 on 8K RAMs, but is not connected on 28C64As so you can simply wire it high. Pin 1 is the 28C64A's -Busy output, but it's not connected in 8K-byte RAMs so you don't have to worry about it, either. The process is similar for 32K-byte EEPROMs like the 28C256, but do match up the data sheets before doing anything silly.

Being able to program an EEPROM in the system is an advantage, but it's distressingly easy to clobber your precious code or data with an errant write. If you plan to use an EEPROM as a programmerless EPROM, the jumper and gates shown in Figure 4 provide simple, manual write protection.

Even if (you think) your code is completely under control, other routines may write into "your"

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 8-bit parallel ports, an auto-baud rate detect serial console port, a serial printer port, and much more.

PROCESSOR
- 80C52 8-bit CMOS processor w/BASIC-52
- Three 16-bit counter/timers
- Six interrupts
- Much more!

INPUT/OUTPUT
- Console RS232 autobaud detect
- Line printer RS 232
- Three 8-bit parallel ports
- EXPANDABLE1
- Compatible with 12 BCC expansion boards

MEMORY
- 48KRAM/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 SK RAM	$189.00 Single Qty
BCC52C	Low-power CMOS version of the BCC52	$199.00
BCC51 I	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 (02) 888-6401
Distributor Inquiries Welcome!

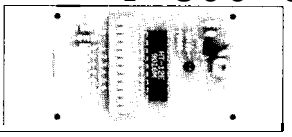
#119

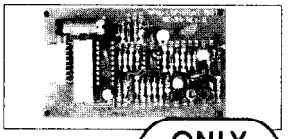
ELECTRONICS 123
A DIVISION OF MING Engineering & Products, Inc.

17921 Rowland Street
City of Industry, CA 91748

Technical Support: (818) 912-9864
VISA MASTERCARD *and DISCOVER*

1-800-669-4406





RC-99 *RF Transmitter & Receiver Set*
(300MHz, 12 bit address or 6 address & 4 data.)
ONLY $34.95

FREE
'93 SPRING CATALOG

Board-Level Items:
- Digital Voice Modules.
- RF Transmitters /Receivers.

Security:
- Auto Alarms and Accessories
- X-IO Home Security Accessories
- Infrareds.

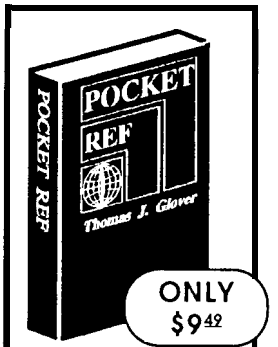
Tools:
- Soldering Irons.
- EPROM Erasers.
- Fluke DMM.
- IC Testers.
- Industrial PC.
- EPROM Programmers.
- Ultrasonic Measuring Device

Prototyping PCB's:
- Full line of Syntax PCB s.

Batteries:
- Alkaline
- Ni-Cad.
- Sealed Lead Acid.
- Button Cell

IC's:
- DRAM memory.
- Holtek Encoder / Decoder IC's.

Radio Communications:
- Motorola Spirit Radios.



ONLY $9.49

Pocket PC Ref for $14.49

#126

Listing 3—*The ROMSCAN program mimics the BIOS signature search through the address space between C0000 and E0000. It displays the header for valid extensions, which will he/p you identify a vacant spot for the Firmware Development Board's nonvolatile memory.*

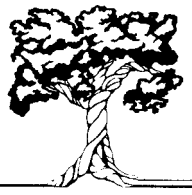
```
BaseSeg = 0xC000;
BaseOffset = 0x0000;

do {
    BlockFlag = peekw(BaseSeg,BaseOffset);
    BlockSize = peek(BaseSeg,BaseOffset+2);
    if ((0xAA55 == BlockFlag) || SHOWALL){
        printf("%04x %04x %04x   %02x ",
                BaseSeg,BaseOffset,BlockFlag,BlockSize);
        if (0xAA55 == BlockFlag){
            TestAddr = BaseOffset;
            CheckSum = 0;
            for (BlockSize *= 0x0200; BlockSize; --BlockSize){
                Checksum += peek(BaseSeg,TestAddr++);
            }
            printf("%02x OK!\n",CheckSum);
        }
        else {
            putstr("-\n");
        }
    }
    BaseOffset += 0x0200;
    if (!BaseOffset){
        BaseSeg += 0x1000;
    }
} while ((BaseSeg < 0xE000) ||
            ((BaseSeg == 0xE000) && !BaseOffset));
```


WHY are our cross compilers so inexpensive???

Because we **give** them away **free!**

We base our cross compilers on the GNU C/C++ compiler from the Free Software Foundation. We provide you with one year of support*, and give you a ready-to-run cross compiler with *complete source* for DOS, Windows, OS/2 2.0 or UNIX for $495 per year. Or, get the extended support package for $895, which includes GNU *Emacs* and *make*, the CVS and RCS source code control utilities, and the TEX typesetting system. Targets include i386, i860,i960, Motorola 680x0, 683xx and 88000, MIPS and Sparc.



Hundred Acre Consulting
5301 Longley Lane Suite D-144, Reno NV 89511
(800) 245-2885 +1-702-829-9700

*Compilers are available without support for a media distribution fee.

#127

address space. For some reason the BIOS setup code in my '386SX writes AA every 2K bytes or so throughout the address range: an unprotected EEPROM won't survive a reconfiguration.

Installing the write protect jumper forces the EEPROM's -WE line high, so any writes will simply fail. M EM TEST's loop reports a timeout error after 10 ms, but any other code will simply conclude that the EEPROM is an unchanging ROM.. .which is exactly what we want.

In next month's column, I'll add a software-controlled write-protect bit for the battery backed RAM, but a manual jumper is enough for now. That circuitry will need an HCT32, which is why I used it here.. .it won't go to waste!

THE BIOS CONNECTION

Now that the Firmware Development Board has a smidge of nonvolatile memory, we can bolt code onto the System Board BIOS to run after each hardware reset. This opens the door to "diskless systems" that boot with no mechanical motion.

Actually, given the limited space available (what can you do in 32K bytes these days?], it's more likely that the (E)EPROM will hold key hardware interface routines rather than the whole embedded application. The rest of the code can be on diskette or, with a little ingenuity that I'll get into in a few issues, be downloaded through the serial port as needed.

In any event, we'll start small: once again the end result will be a few blinking LEDs...but the weight of knowledge behind them should make you feel good!

As I mentioned earlier, the BIOS scans through the memory between COO00 and EOOOO on power up to find BIOS extensions. It's looking for the distinctive signature shown in Figure 5: 55 and AA bytes on a 2K-byte boundary with a valid checksum over the block of storage specified by the length byte.

Listing 3 shows how the search works. I wrote ROMSCAN. C to examine my system's address space so I knew where to put the Firmware Develop-

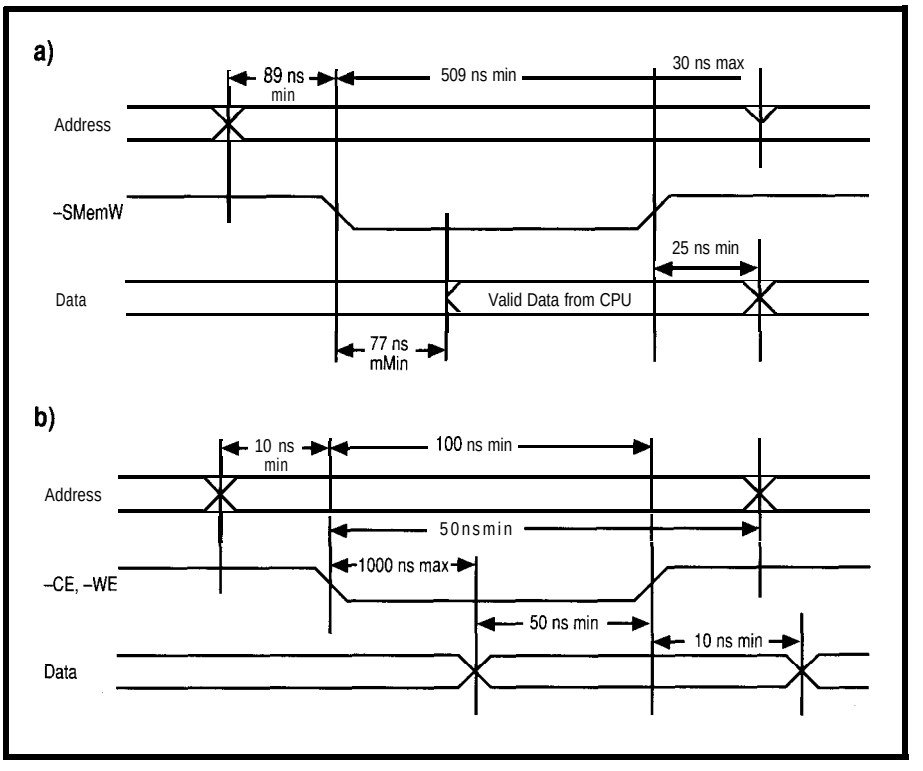


Figure 6—*a) ISA bus memory write accesses are similar to reads. Note that the data may not be valid for quite a while after the leading edge of –SMemW, so the* destination must *rely* on the *trailing* edge for precise liming. *b) The write cycle for a 28C64A EEPROM* is essentially identical *to* a standard *RAM, but the EEPROM cannot accept more data for* about a millisecond. This *timing diagram shows the signals for the first part of the process; the article* text describes *how the* firmware *detects the end of the cycle.*

ment Board's nonvolatile memory. It turned out that the only BIOS extension was a 32K EPROM on the VGA card at COOO:OOOO, but your system may have other ROMs on other cards.

ROMSCAN cannot identify anything that isn't a ROM, so if your system includes EMS cards, network adapters, or other oddities [you aren't doing this on your real PC, are you?) it won't show their RAM buffers or other ROMs. On the other hand, neither can the BIOS, so we're even.

When the BIOS finds a valid extension, it does a FAR CALL to the instruction starting at offset 0003. It is your responsibility to put the first byte of your routine at that address! When your code is done initializing itself, it should return control to the BIOS with a RETF (far return) instruction. The BIOS then seeks out other extensions and, after calling all of them, it continues with the normal disk boot process.

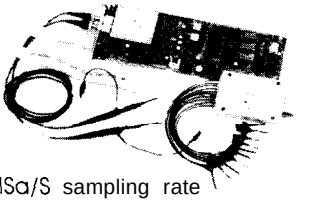
The length byte counts in units of 512 bytes starting from offset 0000, so a length of 02 indicates a 1024-byte block. If your routine is only 800 bytes long you must still use a length code of 02. An 8K-byte EEPROM with a single extension will use a length code of 10.

The checksum covers the entire block, but the BIOS does not care where you put the checksum byte itself. As long as the (length x 512) bytes starting at offset 0000 add up to zero, the BIOS is happy. The code for this column puts the checksum in offset 0005 just after a short jump, but that is entirely my convention.

Listing 4 shows a simple BIOS extension with two useless functions. If the low-order DIP switch on the Firmware Development Board is ON, it blinks the LEDs forever to indicate that it's in control. If the switch is OFF, it turns on both decimal points and returns to the BIOS to continue the normal boot sequence.

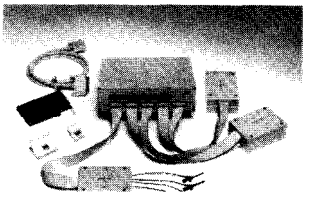
Before loading this into the (E)EPROM, we must compute the checksum. I modified the diskette boot loader from issue 3 1 to perform both functions on a system with the Firmware Development Board installed. After reading the file from

200MSa/S Digital Oscilloscope



· 200 MSa/S sampling rate
· PC-BASED INSTRUMENT
· 2 Analog channels
· 8 Digital channels (8ch. logic analyzer)
· 125MHz Single shot Bandwidth
· 4K samples/channel (analog & digital:)
$1599 - DSO-28 100
$1999 - DSO-28200
Price is Complete Pods and Software included

400 MHz Logic Analyzer



· up to 128 channels
· up to 400 MHz
· 16K samples/channel
· Variable threshold
· 8 External clocks
· 16 level triggering
$799 - IA12100 (100 MHz,24 Ch) Price is Complete
$1299 - LA32200 (200 MHz,32 Ch) Pods and Software
$1899 - LA32400 (400 MHz,32 Ch) included

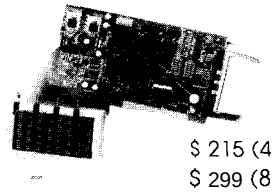
Universal Programmer

'AL
GAL
E PROM
EEPROM
'LASH
MICRO



$475
5ns PALs
4 MEG EPROM (8 & 16 bit)
22V 10 & 26CV 12 GALs
Free software updates on BBS

Gang Programmer



$ 215 (4 Socket)
$ 299 (8 Socket:

Up to 1 MEG EPROMS

Call (201) 808-8990
Link Computer Graphics, Inc.
369 Passaic Ave. Suite 100, Fairfield, NJ 07004 fax: 808-8786

#128

diskette as usual, LOADEXT writes it into the EEPROM, computes the checksum, and stuffs that into offset 0005.

Once LOADEXT is finished, install the write-protect jumper, pop the diskette, and hit the reset button to start your new BIOS extension.. .it's that easy!

One gotcha that is painfully obvious in retrospect: if your code doesn't fill a multiple of 512 bytes, the checksum must include whatever junk is in the last block. You cannot compute the sum on just your code, because the length byte includes more than that. Listing 5 shows the code needed to figure the EEPROM checksum.

For EPROMs, of course, the unprogrammed bytes are FF, so you can compute the checksum correctly without actually having to handle those bytes one by one. In fact, because the hex file doesn't include the unprogrammed bytes, you'll have to use the length to figure out how many FFs the BIOS will include in its calculations.

BIOS EXTENSION HINTS & TIPS

Much of the Firmware Development Board's hardware will be supported by BIOS-like routines in the nonvolatile memory we just got working, but that code will appear in separate columns after we get the hardware thrashed out. Many of you are champing at the bit, though, so I'll hit the high spots here.

First of all, I don't think it's practical to write BIOS extensions in C. After all, we have a pile of stuff to fit into a very small bag.. this is a job for assembler code! While I'm sure it's possible to modify the Micro-C startup code to run from the EEPROM, it requires Small model and some additional code. Dave's comments in the startup code files should be helpful if you want to try it.

The code in COM files produced by PC linkers starts at offset 0100, but the BIOS extensions must begin at offset 0. Probably the easiest way to relocate the code is to subtract 0010 from the CS register and use an indirect branch to increment IP by 0100. For example, the byte at C800:0003 is also located at C7F0:0103. It's the same trick I used in the boot sector loaders, so check there to see how it works.

Any RAM needed by the EEPROM code must be in the lower 640K and you must ensure that the routines don't step on each other's storage. Remember that the DOS memory allocation routines don't exist! The BIOS keeps track of the memory size, so you can allocate space by reducing that number. In effect, your data will lie beyond the end of memory, but only because you've moved the "Dead End" sign up a few feet.

The BIOS calls all of the extensions after it's set up all of the interrupt vectors but before it attempts to boot from diskette. Your initialization code can hook any interrupts that it will need to regain control later on: timer ticks, serial ports, whatever.

In particular, Int 19h is called after all the extensions are initialized to handle the disk boot load. If you capture Int 19h, your code regains control just before the BIOS expects to boot from the diskette.. .so you can boot from, say, a program loaded over the serial port.

Int 18h, on the other hand, is supposed to be called after Int 19h concludes that there are no bootable diskettes or hard disks. By capturing this interrupt, you have the option of booting from diskette to update the firmware (for example) while running without a diskette the majority of the time.

According to my references Int 18h is not supported on all clones, as it was originally used to start good old IBM Cassette BASIC. I'll give it a try, but some BIOS spelunking on your own system is in order to be sure it works the way you want.

Listing 4—*BIOS* extensions normally *do something useful, but this is just a demonstration. If the low-order DIP switch is ON if will "lock up" and blink the LEDs forever. If the switch is OFF if will turn on the LED decimal points and return to the BIOS to continue the normal boof sequence.*

```
                CODESEG
                STARTUPCODE

                DB      055h            ;  signature
                DB      0AAh
                DB                      ; length in units of 512 bytes
MainEntry:
                JMP     SHORT Booter    ; force two-byte jump
                DB      000h            ; zero checksum until loaded
Booter:
                MOV     DX,SW_ADDR      ; should we run?
                IN      AX,DX
                MOV     DX,LED_ADDR     ; set up for display

                TEST    AL,01h          ; low switch ON?
                JZ      Onward          ; zero is yes, so stay here

                MOV     AX,08080h       ; show we were here
                NOT     AX
                OUT     DX.AX           ; just decimal po nts!

                RETF                    ; and return to BI OS!
Onward:
ReShow:
                MOV     AX,0FFFFh       ; all LEDs go off
                OUT     DX.AX

                MOV     CX,0
Waitl:          LOOP    Waitl

                MOV     AX,00000h       ; all LEDs go on
                OUT     DX,AX
                MOV     cx,o
Wait2:          LOOP    WAait2
                JMP     ReShow          ; continue forever
```

Listing *5-The B/OS extension checksum includes everything in the defined block. This routine, taken from the LOADEXT boot sector EPROM loader, computes the checksum based on the extension's code plus whatever is in fhe last block beyond the end of fhe code. It writes fhe result at offset 0005 in the EEPROM, which must be a zero in the disk file.*

```
                MOV     AX,EEP_SEG          aim at EEPROM
                MOV     DS,AX
                XOR     SI,SI             , beginning of block
                MOV     AL,[DS:0002h]     ; pick up length code
                MOV     AH,0                set high byte
                MOV     BX,512            convert to bytes
                MUL     BX                … in AX
                MOV     CX,AX             set up for loop
                MOV     BH,0              set up checksum
MakeCSum:
                LODSB                     pick up data byte
                SUB     BH,AL             tick checksum
                LOOP    MakeCSum          over entire block

                MOV     AL,BH             set up checksum
                MOV     DI,00005h         … address
                CALL    WriteEEP            do it
```

RELEASE NOTES

The code on the BBS this month includes the source and hex files for everything you've seen here, as well as modifications to the FIRMDEV.H and .ASM files mentioned in the past. The EEPROM boot loader and EXTDEMO BIOS extension are written with Borland's TASM, but everything else is in Micro-C.

Be careful with diskettes containing the LOADEXT boot loader, because, unlike BOOTSECT from issue 31, they require a Firmware Development Board. If you boot them in an ordinary system, it will seem to hang without any error indication. It won't damage anything, but it can be pretty scary.

By the time you read this, we should be settled in North Carolina. I hope all my machinery makes it through the move!

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 on CompuServe at** 74065,1363 **or through the Circuit Cellar BBS.

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

CIRCUIT CELLAR PROJECT **FILE**
SPECIAL SUBSCRIBER OFFER

GET BOTH VOLUMES I & II FOR ONLY $29.95*

*For **a limited time only!***

TOP projects from the Circuit Cellar Design Contest
NEW projects & tutorials
Something for every interest

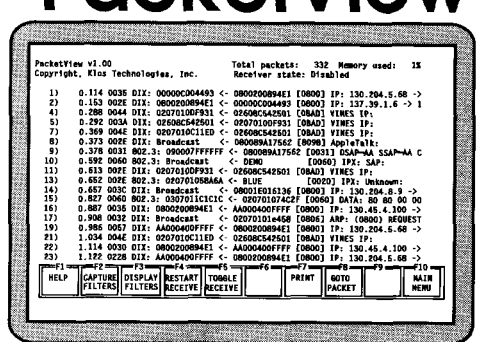
***Order both** volumes and save!* (regularly $17.95* each)
VISA, MasterCard, or International Postal Money Order (U.S. funds drawn on U.S. bank only)

Circuit Cellar Project File
4 Park Street, Vernon, CT 06066
Tel: (203) 875-2199
Fax: (203) 872-2204

"includes domestic delivery. Please add $6 per copy for delivery to Canada/Mexico via U.S. Mail; add $8 per copy for other non-U.S. addresses via U.S. Mail.

Introducing: **PacketView™**

PacketView is the first truly low-cost network analysis and debugging tool available for the PC! Supporting Ethernet, Token-Ring, and ARCNET, PacketView can go places no other software-only analyzer can! PacketView is an essential tool when developing network drivers, protocol stacks, routers, and many other network based software products! And at only $249, how can you go wrong? Download the demo from our BBS at (603) 429-0032 and see for yourself just what PacketView can do for you!



Only $249



Features:

- Protocols recognized: TCP/UDP/IP, SNMP, Novell IPX/SPX/NCP, AppleTalk, XNS and Banyan VINES
- Use your own PC and Ethernet network adapter (in most cases)
- Programmable Filters for Capture and Display
- Custom Protocol Decoders can be developed with C or Assembler
- 24-Hour BBS and 6 Months Free Software Updates
- SerialView Coming Soon! For SLIP and PPP Protocol Analysis

Klos Technologies, Inc.
604 Daniel Webster Highway, Merrimack, New Hampshire 03054
Voice: (603) 424-8300 FAX: (603) 424-9300 BBS: (603) 429-0032

#1

Breathing New Life Into an Old Friend: Revisiting the Z8

Is there a surgeon in the house? Jeff performs a heart transplant on an off-the-shelf 8031 processor board to install the venerable Zilog Z8 and gets a speed demon out of the deal. Who said newer is better?

FROM THE BENCH

Jeff Bachiochi

Tonight the space shuttle Discovery is once again keeping its silent vigil over the Earth, circling once every 93 minutes. A hydrogen valve sensor on one of the liquid-fuel engines refused to signal a closure, scrubbing the launch set for last night. These 1:30 A.M. launches are unaccommodating for us first shifters.

I view the "NASA Select" broadcasts on Satcom F2R, Transponder 13. It is offered to all cable systems without charge, though few of them actually carry it. Twenty-four-hour pre-to-post flight coverage preempts the normal four-hour blocks of educational science-oriented programming. This flight, STS-56, is investigating both ozone depletion and solar flare activity.

No, this is not an infomercial. However, I do strongly believe that the U.S. would not have today's level of technology if we hadn't invested in the space program 30 years ago. And it would be a serious mistake to stop investing now (soap box mode off).

The "red crew" is preparing to exercise the robot manipulator arm, as my telephone starts ringing. I decide to let it ring. "One of the kids will pick it up," I think to myself. The ringing eventually stops. As I refocus my attention on the live shot of the Earth taken through the shuttle bay window, I'm handed the portable phone.

"Hello," I mumble while staring at the view of the Earth from above. "This is Johnson Space Center calling. Is this Jeff Bachiochi?" Surprised by this unlikely statement, I stammer "Huh. What? Yes! Wait, who is this really?" The caller assured me he was from Houston and explained how he had called directory assistance starting with Circuit Cellar's office exchange and then continued to call neighboring exchanges in a spiral pattern until he located me. Well, he had my attention.

"We've got a serious problem here and if we can't solve it this weekend, we want to fly you down to Houston on Monday morning." I felt the strain in his voice. "I want you to talk to the project engineer."

For the next 30 minutes we discussed the system they were developing for use on Space Station Freedom. It had been about ten years

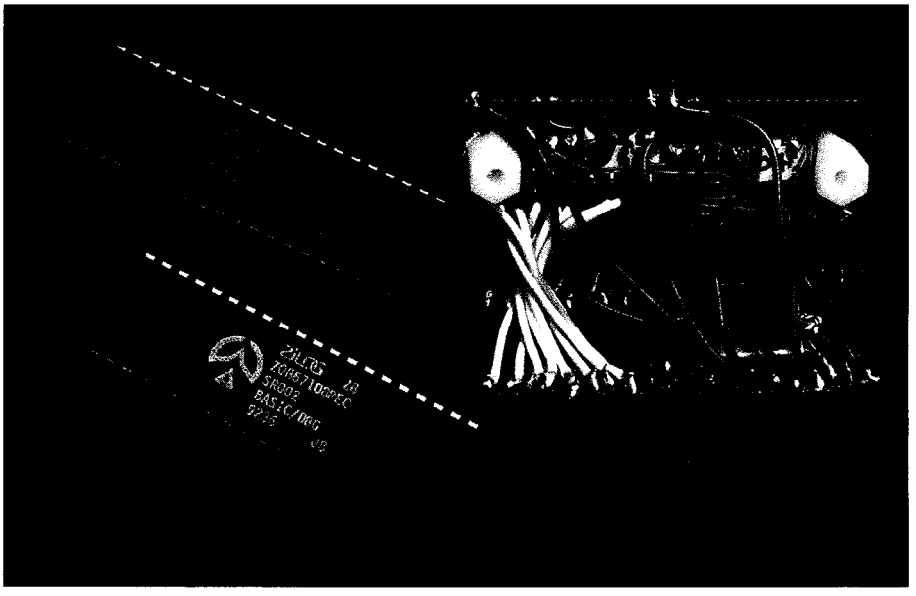


Photo I--The *processor adapter consists of a 40-pin component* carrier *wired to a 40-pin socket Stranded wire was used in an attempt to avoid breakage when the pieces are flexed.*

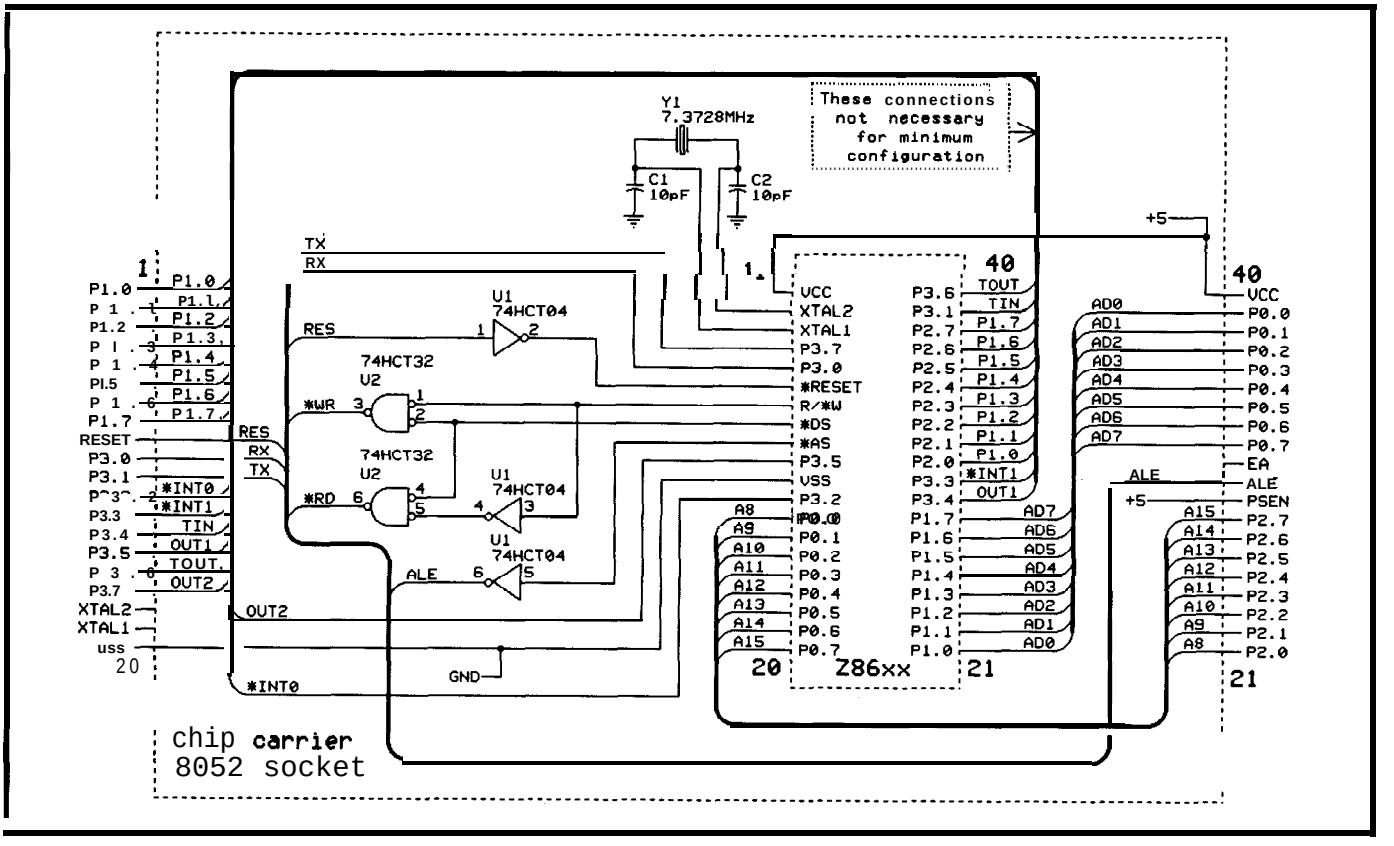


Figure 1—*The interface circuit adapter requires some extra glue logic and a 40-pin chip carrier plugged into a 40-pin DIP IC socket.*

since I had worked at any length with that system, which was based on the Zilog Z8671. Fortified with a barrage of diagnostic tests to perform and signal points to investigate, we parted phone lines. I assured them I would be around most of the weekend if they needed to confer again.

STILL CRAZY AFTER ALL THESE YEARS

The Zilog 28671 is part of the Z8 family of processors. Masked into the part is a 2K tiny BASIC interpreter. As a subset of Dartmouth BASIC, Z8 BASIC/Debug allows the user to easily examine and modify any memory location or I/O port as well as do bit manipulation and logical operations. For faster execution, machine language routines can be called from BASIC as subroutines or functions.

Unlike some of the newer micros, the 28671 can operate without external RAM. Programs using few variables can share the internal stack space for variable storage, which saves real estate and parts cost. If external RAM is available, it is automatically used for variable storage.

The Z8 family of microcontrollers dates back to the late '70s and boasts both 2K and 4K, maskable ROM and protopack parts. The protopack parts actually accept an EPROM piggyback style as opposed to a windowed programmable part, making it much easier to use. The preprogrammed part, the 28671, incorporates "Tiny BASIC," which in those days opened a whole new world for hardware junkies. Imagine being able to test out your I/O without having to sit down with an assembler.

Along with the introduction of an expandable microcontroller system based on the 28671 (one of Steve's early BYTE projects), another programming language was gaining momentum: Forth. Back in those days, we were crazy enough to try just about anything, including developing and masking a Z8 with a Forth compiler (based on the Forth-79 standard); 4K Forth anyone!

I've always liked the Z8 because of its register-oriented architecture. The internal register file consists of 124 general-purpose registers, 16 CPU and peripheral control registers, and 4 I/O registers, each eight bits wide. Any of the general-purpose registers can be used as an accumulator, address pointer, index, data, or stack register (a very flexible arrangement). Using register pointers and working register groups allows for fast context switching and shorter instruction formats.

USED 2K RAMS

That first Z8 system made used two 2K RAM chips, and, like my first 16K TRS-80 Model 1, at the time I thought I would never use all that RAM. But now, having rekindled my interests in the Z8, I was having second thoughts about pulling out that system again with its memory limitations. What I really needed was to slip that processor into a newer chassis.

THE HOUSE THAT JACK BUILT

I like recycling. Why can't I reuse an existing design but with different components? I mean, not new parts in a new design, but different parts in an unmodified, preexisting design? Confused? Well, hang on a second, and I'll try to clear it up for you. Remember the RTC52 and I/O boards I presented

+	! (store)	>=	CR	H.	R> (R-FROM)
–	# (sharp)	>IN	CREATE	HERE	REPEAT
*	#>	>R	D+	HOLD	ROT
/	' (tick)	@ (fetch)	D.	I	S->D
AND	+	AGAIN	DABS	IF	SIGN
GO@ (mach.lang. call)	+!	ALLOT	DIGIT	IMMEDIATE	SMUDGE
GOSUB	+LOOP	AND	DLITERAL	J	SPACE
GOT0	, (comma)	BASE	DNEGATE	KEY	SWAP
IF	–	BEGIN	DO	LEAVE	THEN
INPUT	." (dot-quote-space)	BLK	DOES>	LITERAL	TYPE
LET	0=	C,	DROP	LOOP	U*
LIST	1+	C!	DUP	MIN	U/
NEW	2	C@	ELSE	NEGATE	UNTIL
PRINT	DROP :	CASE	EMIT	NUMBER	WHILE
REM		C-MOVE	ENDCASE	OF	WORD
RETURN	<#	COLD	EXECUTE	OR	XOR
RUN	<	COMPILE	EXPECT	OVER	[(left bracket)
STOP	BUILDS<null>	CONSTANT	FIND	PAD	] (right bracket)
THEN	=	COUNT	FORGET	QUIT	
USR (x)					
USR (x,y)					
USR (x,y,z)					

Figure 2—*The 2K Tiny BASIC implements all of the most useful BASIC commands (left). Incredibly, the 4K Forth implements four times as many commands (right) in only twice the space. Program execution is also much faster.*

in "From the Bench" in the April/May '89 issue of ***Circuit Cellar INK!*** I've used these boards over and over again since then. Well, let's just pull out that old 80C52 processor and pop in the Z...er, wait. That won't work-at least not yet anyway.

Refer to Figure 1 to see how you can make an adapter to retrofit your RTC52 (or any other 803 I-based system) for use with a Z8 processor. You can make this interface circuit fit in the same space occupied by one 40-pin chip by using stacked sockets to hold the "translation circuit." To do this, start with a 40-pin chip carrier plugged into a 40-pin DIP IC socket (this way they will remain aligned with one another). Drill a 1/8-inch hole centered between the rows of pins at each end of the socket. Now separate the socket and carrier from each other. Two short #4 nylon standoffs are used to hold the socket (which the Z8 will plug into) above the carrier (which plugs into the 803 1 socket on the circuit board). All the interfacing circuitry is sandwiched between the carrier and the socket. The finished interface circuit adapter appears in Photo 1.

Three inverters and two AND gates are needed to transform the Z8's output signals into those which closely correspond to those of the 8031. The two glue-logic chips are super glued upside down to the top surface of the chip carrier. I stuck small labels on to their exposed bellies to identify which was which and where pin 1 was located. Simple tricks like this one tend to greatly improve your chances for continued sanity later on. I used wire-wrap wire to connect the appropriate pins of the glue-logic chips to the pin-leads on the chip carrier, but when wiring the "straight through" connections between the carrier and the DIP socket, I selected the smallest size stranded wire I could find. This method improved the flexibility between the two parts which would be opened and closed a number of times like a book while I was testing this rig. I did not want brittle connections to cause me grief.

To make swapping even simpler, the Z8671's DIP socket has a 7.3728-MHz crystal tied directly to it. The crystal on the RTC board is not used to drive the Z8 and those pins are left unconnected.

USE THE FORCE

The ROM (BASIC/Debug and Forth) versions of these parts read address %FFFD on reset to determine the console data rate. This value can be read from the EPROM or an I/O port depending on which is mapped

Listing 1--The *Z8671 BASIC/Debug chip contains an internal 2K tiny BASIC interpreter that speeds program development.*

```
1 REM $NAME/Z8/ $SOURCE
10 PRINT "Print prime numbers between 1 and ???"
20 INPUT N : REM Get Maximum Number
30 GOSUB 100
40 PRINT "End of run"
50 STOP
100 X = 0 : REM Initialize Number
110 X = X+1: REM Next Number
120 Z = 0 : REM Reset PRIME Flag
130 Y = 1 : REM Initialize Check
140 Y = Y+1: REM Next Check
150 IF Y >= X THEN GOT0 180
160 IF X = (X/Y)*Y THEN Z = 1
170 GOT0 140
180 IF Z = 0 THEN PRINT X: " is a prime number"
190 IF X < N THEN GOT0 110
200 RETURN
```

Listing 2-The *BASTO4TH program* automatically converts a *Z8 BASIC/Debug program into Z8Forth, making the BASIC-to-Forth* transition *a bit smoother.*

```
*****************************************************************
1 REM $NAME/Z8/ $SOURCE
*****************************************************************

     (assorted initialization code omitted for clarity)

*****************************************************************
10 PRINT "Print prime numbers between 1 and ???"
*****************************************************************
  : aab
." Print prime numbers between 1 and ???"
 CR ;
*****************************************************************
20 INPUT N : REM Get Maximum Number
*****************************************************************
  Variable vN
 : aac vN #IN
 ( Get Maximum Number)

*****************************************************************
30 GOSUB 100
*****************************************************************
  Variable aac0
 : aad bptr @ aac0 @ bptr !;
*****************************************************************
40 PRINT "End of run"
*****************************************************************
  : aae
." End of run"
 CR ;
*****************************************************************
50 STOP
*****************************************************************
  : aaf 2 endr ! ;
*****************t***********************************************

100 X = 0 : REM Initialize Number
*************************************t**********************
 10 aac0!
 Variable vX
 : aag 0 vX!
 ( Initialize Number)

*****************************************************************
110 X = X+1: REM Next Number
*****************************************************************
  : aah vX @ 1 + vX!
 ( Next Number)

*****************************************************************
120 Z = 0 : REM Reset PRIME Flag
*****************************************************************
  Variable vZ
 : aai 0 vZ !
 ( Reset PRIME Flag)

*****************************************************************
130 Y 1 : REM Initialize Check
*****************************************************************
  Variable vY
 : aaj 1 vY!
 ( Initialize Check)

*****************************************************************
140 Y = Y+1 : REM Next Check
*****************************************************************
  : aak vY @ 1 + vY!
```

(continued)

8051 SBC AT A NEW LOW PRICE

We are proud to offer our standard **8031SBC-10** Single Board Computer at a new, low price - just $79 per unit or as low as $49 each for quantity purchases. An 8031 with two JEDEC sockets, one RS232, 5V regulator, expansion connector. Optional second serial port, 80C31 or 32.

Controller 80C552

At $149, our 552SBC-10 has the price and features you need right now! It's an 8051 core processor with an eight channel, 10-bit A/D, two PWM outputs, capture/compare registers, one RS232, four JEDEC memory sockets, and more digital I/O. And we didn't stop there! You can add options like two more RS232/422/485 ports, 24 more digital I/O ports, Real-Time Clock, EEPROM, and battery-backup for clock and RAM right on board. Start with the Development board; it has all the peripherals plus a debug monitor for only $349. Download and debug your code right on the SBC, then move to the OEM board above for your production needs. We also do custom design work - call for our reasonable prices.

New 8051 Family Emulator Support

Our **DryICE** Plus product has been expanded to include support for the Siemens **80C537**. The base emulation unit is still only $299, with the 80C537 pod priced at $199. Other 8051 family processors supported are 8031/32,80C31/32, 8751/52, 87C51/52, 80C154, 80C451, 80C535, 80C552/562, 80C652, and 80C51FA,B,C. Each of these pods is priced at $149. Where else can you get an emulator with this much power and flexibility for only $448 complete?

Our original stand-alone 8031 ICE is still priced at $199. Though not as flexible as the DryICE Plus, it offers excellent price/performance for learning or the occasional job need.

Call for your custom product needs. Free Quote - Quick Response

VISA MasterCard AMERICAN EXPRESS Card

HTE HiTech Equipment Corp
9400 Activity Road
San Diego, CA 92126
[FAX: (619) 530-1458]

(619) 566-1892

#130

into the system at that address. Since I was only ever going to use the processor at 9600 bps, I cheated! If no device is enabled at address %FFFD, nothing drives the bus and it floats. This is read back as an indeterminate value, unless a little force is used. By tying D1 high and DO and D2 low with 10k pull-up and pull-down resistors, the undriven bus will look like xxxxx010 (9600 bps] at any undriven address.

28 FORTH DEVELOPMENT ON YOUR PC

Working with Forth on the Z8 is as easy as working with BASIC/Debug: just connect a terminal and type (or download) your program. For those who like to do their developing on the PC, the Z8FORTH program is designed to run as a Forth processor under MS-DOS. The program is fine for some code development, but loses its usefulness when you need to handle hardware I/O (which isn't supported by the simulator).

A list of the keywords for these languages is shown in Figure 2. If you are a bit shy about learning a new (or old) language, don't fret. The BAST04TH translator, also an MS-DOS program, shows you how it's done. Taking either a BASIC file or keyboard input, the BAST04TH translator will display each line of BASIC and its translated counterpart. Let's look at a simple program and its conversion and compare execution speeds of the two programs run on the same system. To get an idea of what you feed into the BAST04TH translator, see the BASIC program shown in Listing 1. To see what the translator creates from that program, take a look at the code in Listing 2.

I played no tricks, did no smart coding; it's just straightforward code using as many BASIC/Debug statements as necessary. The translated code, minus the commented BASIC lines, was about 2K, whereas the BASIC/Debug code was around 500 bytes. The BASIC routine took 2 17 seconds to check for primes between 1 and 100. The Forth code did the same job in 170 seconds with the same RTC system, same crystal, and at the same baud rate as the BASIC processor.

Listing 2-continued

```
 ( Next Check)

*****************************************************************
150 IF Y >= X THEN GOT0 180
*****************************************************************
  Variable aakl
 : aal vY @ vX @ >=
 IF aakl @ bptr !
 THEN ;
*****************************************************************
160 IF X = (X/Y)*Y THEN Z = 1
*****************************************************************
  : aam vX @ vX @ vY @ / vY @ * =
 IF 1 vz !
 THEN ;
*****************************************************************
170 GOT0 140
*****************************************************************
  : aan 18 bptr !;
*****************************************************************
180 IF Z = 0 THEN PRINT X: " is a prime number"
*****************************************************************
 26 aakl !
 : aao vZ @ 0 =
 IF vX @.
 "  is a prime number"
'CR
 THEN ;
*****************************************************************
190 IF X < N THEN GOT0 110
*************************************xx**********************
  : aap vX @ vN @ >= 0=
 IF 12 bptr !
 THEN ;
*****************************************************************
200 RETURN
*****************************************************************
  : aaq bptr !;
```

HAPPY LANDINGS

While we can't necessarily teach old processors new tricks, we can provide them a larger field in which to play. Zilog, on the other hand, continues enlarging its family of Z8 processors. They begin with 18-pin DIPs and extend up through 84-pin PLCC parts (complete with internal DSP), but I'll leave some of these parts for another day.

Meanwhile, Discovery lands safely at Cape Kennedy, bringing back with it huge amounts of data from the Atlas 2 and Spartan Freeflying Payload Satellite. I'm going to need some sleep real soon now; the next flight is scheduled to launch next weekend. I can only hope I'm not needed in Houston.

Special thanks to Steve (Forth) Chalmer (wherever you are).

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.*

SOURCE

Micromint, Inc.
4 Park St.
Vernon, CT 06066
(203) 871-6170

2867 1 BASIC/Debug Processor $25.00
Zilog Z8 Technical Manual $20.00
Z8 Forth Processor $19.00

I R S

413 Very Useful
414 Moderately Useful
415 Not Useful

Talking Chips

SILICON UPDATE

Tom Cantrell

Still seeking that single-chip solution to add voice to your next project? Forget CODECs, ADCs, and DACs. The latest from Information Storage Devices records your voice using analog techniques.

Remember when? It must've been the early '70s. The world was chip-giddy, and everything seemed possible. Then some diabolical dashboard designers decided that drivers would appreciate nagging robotic mothers-in-law.

"The door is ajar.. .The door is ajar.. .The door is ajar.. "

The door is not a @#$% jar! If it were, I wouldn't be making these giant car payments! And furthermore, it would keep its mouth shut!

Thankfully, this "feature" has long since disappeared.

However, it really isn't fair to criticize the whole concept of voice output just because of one inappropriate use. That would be the high-tech equivalent of killing the messenger because you don't like the message.

Indeed, if we observe the prime directive that all sound-emitting gadgets should have a volume control (lest wires be cut as they were on many of those gabby cars], then I can think of a number of applications that might benefit from vocalization. Generally, they arc the applications in which it is inconvenient, or even dangerous, to rely on the operator to read a visual display.

For instance, consider the long-awaited air-collision avoidance system. A visual indicator would simply add yet another flashing light and set of dials to the already over-crowded instrument panel. While, in contrast, a voice solution that directly issued the command to "Pull **up!** You're about to crash," would certainly be recognized by the pilot. Which plane would you rather fly on?

Another sensible application would he a speaking hazardous-gas detector. When crawling around underground it can be rather inconvenient to have to keep looking up from your task to check a visual display. Or, in other cases, it just might he too dark to see it. One definition of "sinking feeling" might be, having been distracted for a minute or two, glancing at the visual display on the detector only to see it says "RIP."

On the more mundane front, there are plenty of times when I'm trying to probe a particularly messy rat's nest of cabling that I would like an audio output on a VOM or oscilloscope.

If you promise to design responsibly (remember the prime directive), I'll fill you in on a unique technology that allows you to add voice input and output to your application with just a handful of components.

YEAH, DAST THE TICKET

Traditionally, a digital voice I/O system uses an ADC to convert the audio for storage in digital memory; the recorded audio is then played back via a DAC. Along the way, both input and output sections need amplification and filtering. The addition of a DSP, an MPU, or a dedicated IC allows compression of the digitized audio to a degree depending on cost and quality constraints. For instance, an ADPCM (Adaptive Delta Pulse Code Modulation) chip can compress telephone-grade speech by a factor of two with little loss of quality.

Now, thanks to DAST technology from Information Storage Devices, it's possible to shrink the entire voice I/O function into a single, easy-to-use, low-cost chip.

DAST stands for "Direct Analog Storage Technology" which, as the name implies, dispenses with the A/D and D/A converters by using a simple method of directly reading ("playing back") and writing ("recording") analog information. Cleverly, it does so using standard EEPROM technology. This makes the parts easy to manufacture, which increases yields and in turn leads to lower cost.

Conventional EEPROMs store binary data by blasting a charge

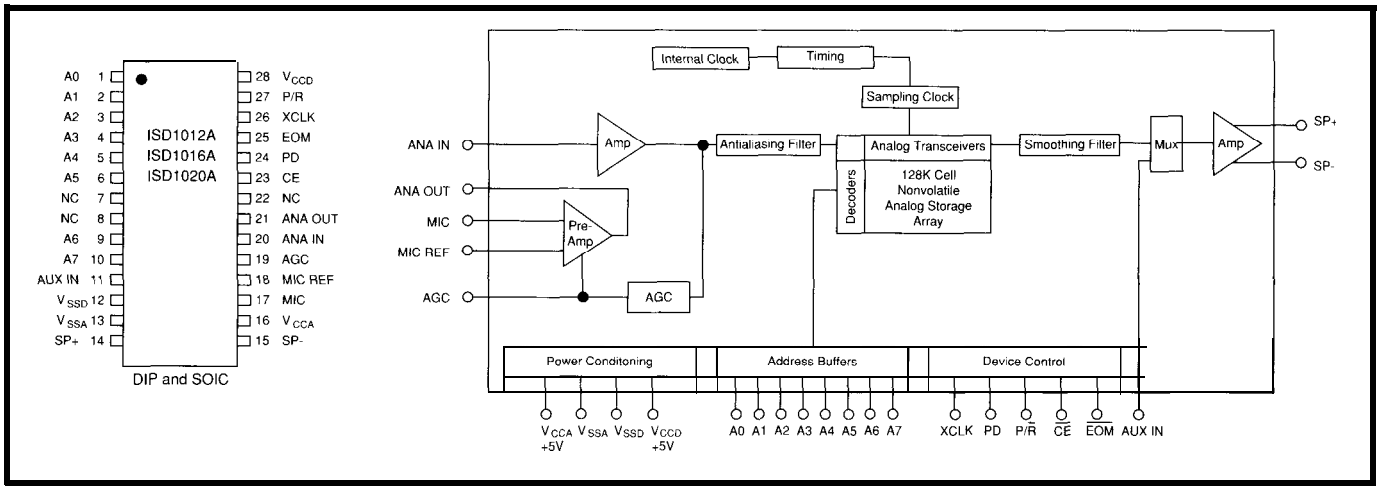


Figure 1—*Information is stored in the ISD1016A in its original* **analog** *form, eliminating* **the need** *for digital converters.*

through a thin oxide layer onto a floating gate. However, the DAST concept exploits the fact that the floating gate can store intermediate levels of charge, not just a 1 or 0.

The DAST write process works as follows: An analog voltage is sampled by the device. This analog voltage is stored in a sample-and-hold capacitor and is fed to one input of a voltage comparator. Then, in a closed-loop fashion, a series of small write pulses incrementally increases the voltage on the floating gate. The voltage level of the gate is fed to the other input of the voltage comparator. The incremental writes continue until the EEPROM cell voltage matches the sampled input as signaled by the voltage comparator. When the comparator voltage and the cell voltages match, the sampled data is considered written into that cell.

A benefit of the DAST scheme is that it is impervious to variations between cells. Writing the same voltage level to two unmatched cells may require different numbers of write pulses, but both will end up storing the same voltage level because of the comparator feedback loop.

Across the range of O-2.75 V, 12-mV resolution is possible. This means each DAST cell can store 230 distinct voltage levels. In other words, one cell of DAST storage is nearly equivalent to eight bits of digital storage.

Of course there are tradeoffs. If there weren't, with an 8: 1 cost advantage, ISD would surely be the leading-if not the only-chip company in the world. While such "multistate" techniques (which incidentally have long been used in such chips as 80x87 math coprocessors featuring a-level microcode) are predicted to overcome future density limits of VLSI, for now the extra write circuitry, variable write time, and speed / accuracy tradeoffs rule out DAST as a generic replacement for 1s and 0s. However, the technology is a good fit with the requirements of

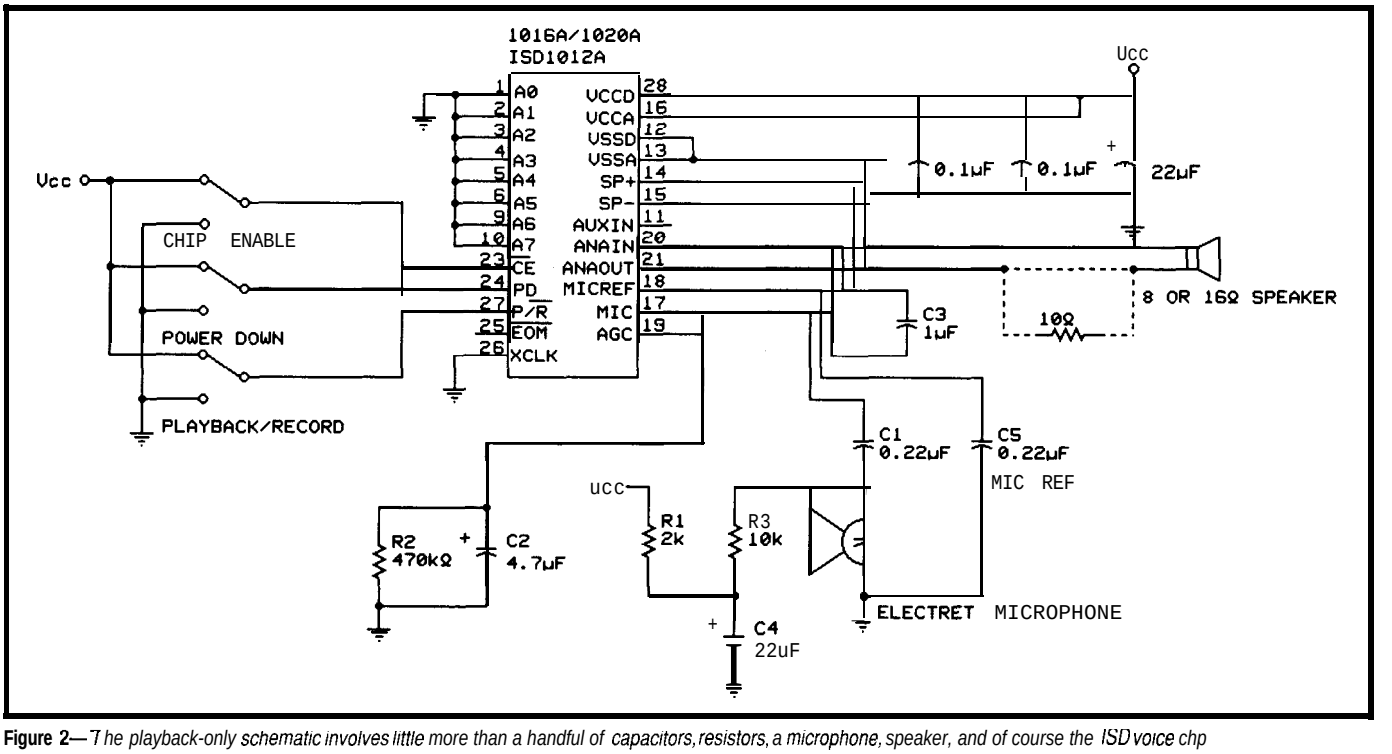
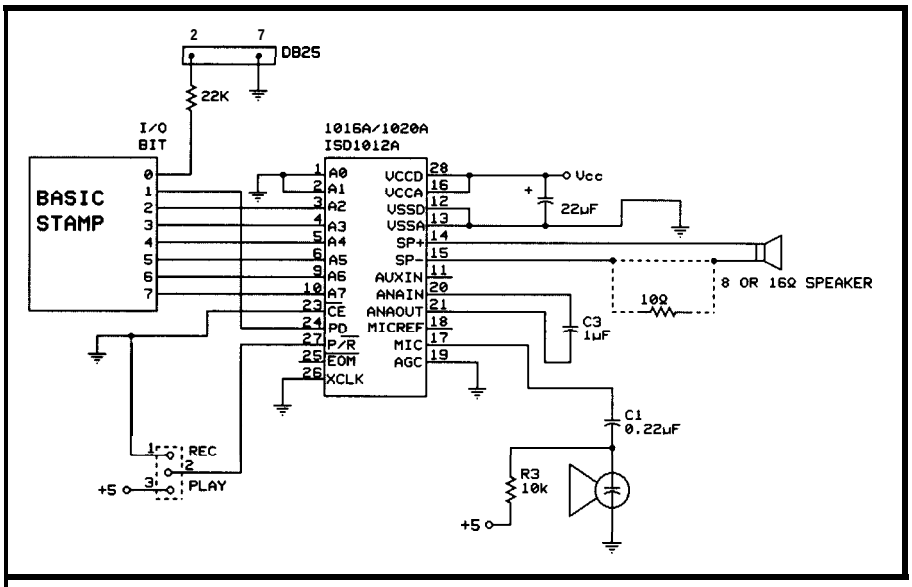


Figure 2—*The playback-only schematic involves little more than a handful of capacitors, resistors, a microphone, speaker, and of course the ISD voice chip*



gure 3-Adding some intelligence to *the setup (in the form of the Parallax BASIC Stamp) allows you to* **record new** *messages into the ISD10xxA* **chip.**

low-cost voice I/O, a market directly targeted by the ISD1016A.

ELOQUENT SILICON

As shown in Figure 1, the ISD1016A surrounds a 128K DAST EEPROM array with all the messy analog stuff-amps, filters, and so on—in a single package. For this discussion, I'll break the functions down into four basic areas-analog input, address, control, and analog output.

On the input side, the MIC and MIC REF signals are AC coupled to the chip from a typical electret microphone (such as Radio Shack P/N 270-090). Otherwise, only a biasing resistor is required, although an extra RC network to provide power supply decoupling may be useful to help reduce noise.

The output from the microphone preamp then takes a detour off-chip [ANA OUT). This pin is normally connected back to ANA IN through a capacitor and feeds the on-chip amplifier. The capacitor simultaneously acts as a filter and as a coupling capacitor. This admittedly strange looking setup is intentional-it allows you to substitute a direct analog input for the microphone and the extra capacitor serves as another pole for the filter.

The input also features automatic gain control (AGC) which handles varying sound levels (i.e., whispers vs. shouts) while maximizing dynamic range and minimizing distortion. An RC network connected to the AGC pin determines the attack and hold characteristics of the gain control.

The antialiasing filter is designed to limit the input frequency to less than the Nyquist limit. The passband depends on the sampling clock. For the '1016A, which packs 16 seconds of audio into the 128K DAST array, the sampling rate is 8 kHz, so the cutoff frequency for the filter is set at 3.4 kHz. ISD also offers a '10 12A which features 12 seconds of storage. The sampling clock for the '1012 is faster than the '1016 and has a higher passband. The third chip in the ISD line up is the '1020A. This chip offers a sampling clock slower than the '10 16, and it runs with a lower passband.

Speaking of the sampling clock, it's no easy feat to generate an accurate clock without a crystal or any external tuning discretes. However, it is important that the clock be rather precise since variations of as little as a few percent are easily detectable by listeners. The on-chip clock generator is trimmed at the factory for accuracy within 1.5% of spec. The sampling clock is also stable to within 0.5% of spec over the operating temperature and voltages for the device.

The "analog transceivers" block, in Figure 1, hides some complexity mandated by the slow (10-ms) write-time of the EEPROM. Since the EEPROM can only be written to at a 100-Hz rate, and the sampling clock is 8 kHz, two sets of 8-kbit input buffers (80x100 = 8 kbit) are used in a ping-pong fashion as write buffers. While samples are being written to EEPROM from one buffer, the next sample is accumulating in the other one.

For addressing purposes (via A0–A7), samples are grouped into 100-ms fragments-which is the smallest accessible unit of sound. Thus, the '1016A stores audio from addresses 0–159 in the 256-byte address space. Addresses 192-255 (A6 and A7 high) are reserved for programming various "operational modes" that are most appropriate for push-button (nondirect addressing) applications.

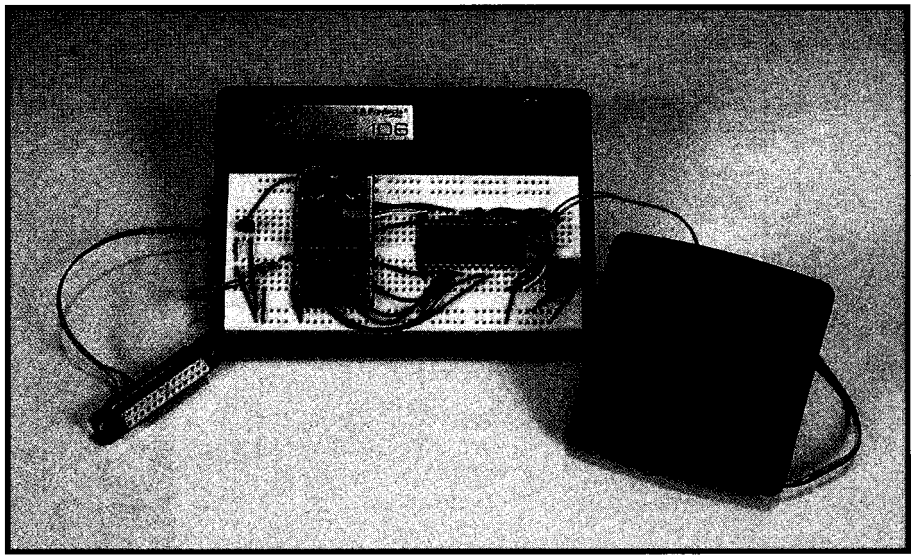


Photo 1-A *solderless* **breadboard and** *BASIC Stamp processor board make experimenting with the ISD voice chips easy.*

Listing 1—*The* Parallax *BASIC Stamp includes a very useful BASIC language on a very small board. A test setup to allow recording of messages on the ISD voice chips can be programmed very quickly.*

```
'A simple ISD1016A recorder
' using the Parallax BASIC Stamp
'Define variables, pins & constants.
'b0-b4 are byte registers

symbol  msg_num=b0       message number ($30-$6f) from host
symbol  msg_addr=b1      address of message in ISD 1016A
symbol  msg_len=b2       length of message in 0.1-sec. increments
symbol  reps=b3          repeat previous message reps times
symbol  count=b4         message length loop counter
symbol  pd=0             PIC pin 0 output to ISD1016A PD
symbol  rxd=1            PIC pin 1 serial input from host
symbol  duration=100     0.1-second message increment (ms)
symbol  gap=250          gap between repeated messages (ms)

dirs=$fd                 'make pins 0,2-7 output, 1 input
pins=1                   'PD  high power down the ISD1016A


loop:
     SERIN rxd,N2400,msg_num    'wait for msg_num from host
     IF msg_num < $30 THEN loop'valid range is "0" ($30) to
     IF msg_num > $6f THEN loop' "o" ($6f)
     msg-num = msg-num $30      'if valid then shift to 0-$3f
     IF msg-num > $f THEN play  'if not 0-$f, then play msg

     FOR reps = 0 TO msg-num    'if 0-$f
       GOSUB playrec            'then repeat previous ms
       PAUSE gap                '1 to 16 times with gap
     NEXT reps                  'in  between

GOT0 loop                       'wait for next msg-num

play:
     msg_num = msg-num $10      'shift msg-num to 0-$1f
     msg_len = 0                'default if msg_num not in table

'look up msg_addr and msg_len corresponding to msg_num
'msg_addr specified in 0.4-second increments
'msg_len specified in 0.1-second increments
'multiple LOOKUP statements are used so they fit on the screen

     LOOKUP msg_num,(0,1,2,3,4,5,6,7,8,9),msg_addr
     LOOKUP msg_num,(4,4,4,4,4,4,4,4,4,4),msg_len

     IF msg_len <> 0 THEN out
     msg-num = msg_num 10

     LOOKUP msg_num,(10,11,12,13,14,15,16,17,18,19),msg_addr
     LOOKUP msg_num,(4,4,4,4,4,4,4,4,4,4),msg_len

     IF msg_len <> 0 THEN out
     msg-num = msg-num 10

     LOOKUP msg_num,(20,21,22,23,24,25,26,27,28,29),msg_addr
     LOOKUP msg_num,(4,4,4,4,4,4,4,4,4,4),msg_len

     IF msg_len <> 0 THEN out
     msg-num = msg-num 10

     LOOKUP msg_num,(30,31,32,33,34,35,36,37,38,39,0),msg_addr
     LOOKUP msg_num,(4,4,4,4,4,4,4,4,4,4,160),msg_len

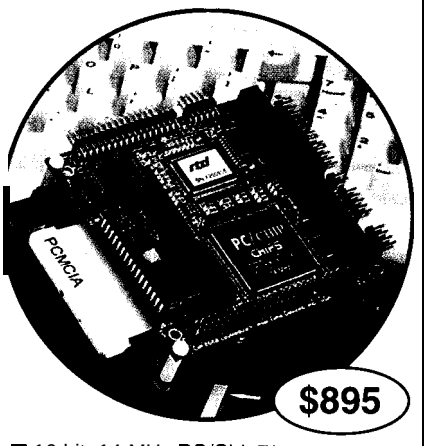
     IF msg_len = 0 THEN loop   'msg_num was not in table-ignore

out:
```

(continued)

The New Shape of Embedded PCs

The amazing CMF8680 cpuModule™ is the first complete 100% PC-compatible ***PC/104 single board computer*** *measuring only 3.6" by 3.8"!*



- 16-bit, 14 MHz PC/Chip™
- CGA/LCD controller
- 2M DRAM
- ROM-DOS kernel
- bootable 1 M solid-state disk
- configuration EEPROM
- 16-bit IDE controller & floppy interface
- PCMCIA interface
- two RS-232, one RS-485 & parallel port
- XT keyboard & speaker port
- watchdog timer
- +5 volts only operation

Designed for low power applications, the CMF8680 draws one watt of power, which drops to 350 milliwatts in sleep mode, 125 milliwatts in suspend mode. Free utility software lets your application boot from ROM!

RTD also offers a complete line of PC/l 04 peripherals for expansion:

- 1.8" hard drive & PCMCIA carriers
- 12- & 14-bit data acquisition modules
- opto-22 & digital I/O modules
- VGA CRT/LCD interface

For more information: call, write or fax us today!

Place your order now and receive a *CM102* ***PCMCIA*** *carrier module FREE!*

rtd®

Real Time Devices, Inc.
P.O. Box 906
State **College,** PA 16804
(814) 234-8087 ■ Fax: (814) 234-5218

#131

SAVE
53%
SUBSCRIBE TODAY TO
THE COMPUTER APPLICATIONS JOURNAL
12 ISSUES FOR ONLY
$21.95*
WRITTEN BYENGINEERS FOR ENGINEERS!
☞ HANDS-ON HARDWARE PROJECTS
☞ ADVANCED APPLICATIONS
☞ TECHNOLOGY TUTORIALS
NO VAPORWARE!
TO TAKE ADVANTAGE OF ALL THIS TECHNOLOGY, JUST FILL OUT THE SUBSCRIPTION CARD ON PAGE 16 OF THIS ISSUE AND
FAST FAX YOUR ORDER!
OR MAIL TO:
THE COMPUTER APPLICATIONS JOURNAL
P.O. Box 7694
RIVERTON, NJ 08077-8794
*Price good in U.S. only. Canada/Mexico $31.95, all other foreign $49.95. U.S. funds drawn on U.S. banks only.

Listing 1-continued

```
      msg_addr = msg_addr*4        'set bits A5-A0 to A7-A2
      msg_addr = msg_addr|1        'set bit PD high
      GOSUB playrec                'play or record
      GOTO loop                    'wait for next msg_num

'play or record message at msg_addr for msg_len*duration ms
playrec:
      PINS = msg_addr              'set A6-A2,PD high
      LOW pd                       'power up
      FOR count = 1 to msg_len
        PAUSE duration  'wait for msg_len*duration ms
      NEXT count
      HIGH pd                      'power down
      RETURN
```

Control over the operation of this chip is quite simple. When PD (Power Down) is asserted, the '1016A consumes a miserly 1 µA (typical)-not surprising since the EEPROM array needs no power to retain its contents. After deasserting PD, the chip starts up and is ready for action within 20 ms or so.

Once powered up, whenever CE* is asserted, the addresses are latched and a playback or record cycle (depending on the state of the P/R' pin) commences. A recording continues until CE* goes high at which point the '1016A inserts an EOM (end of message) marker in the data space. For playback, a simple pulse on CE' will play the entire content of the addressed message until the EOM marker is encountered. At this point the EOM' pin pulses low and playback stops. Alternatively, if CE* is held low, playback will continue through the EOM markers. In either the playback or record mode of the chip, once the device reaches the end of storage (i.e., address 160), the EOM' pin will go low and stay there until a PD cycle resets the chip. The latter feature is useful for cascading multiple '1016As using the "cascade operation mode" in which EOM* from the first chip drives CE* from the next and so on.

The only other control signal is XCLK, which as its name implies, offers the option of externally clocking the chip. Before you get all excited imagining strange uses, keep in mind that it is mainly intended for test purposes or perhaps to generate a more

ASCII Code	Voice
@	oh (zero)
A	d n e
B	two
C	three
D	four
E	five
F	six
G	seven
H	eight
I	nine
J	ten
K	eleven
L	twelve
M	thirteen
N	fourteen
O	fifteen
P	sixteen
Q	seventeen
R	eighteen
S	nineteen
T	twenty
U	thirty
V	forty
W	fifty
X	sixty
Y	seventy
Z	eighty
[	ninety
\	hundred
]	k
^	mega
_	giga
`	pico
a	nano
b	micro
c	milli
d	point
e	ohms
f	volts
g	amps
h	(play whole chip)

Figure 4—*The ISD1016A will hold up to 16 seconds of audio, plenty to sfore fhe 40 phrases necessary to build a talking DVM. Each phrase is accessed from within fhe BASIC Stamp program by an ASCII character.*

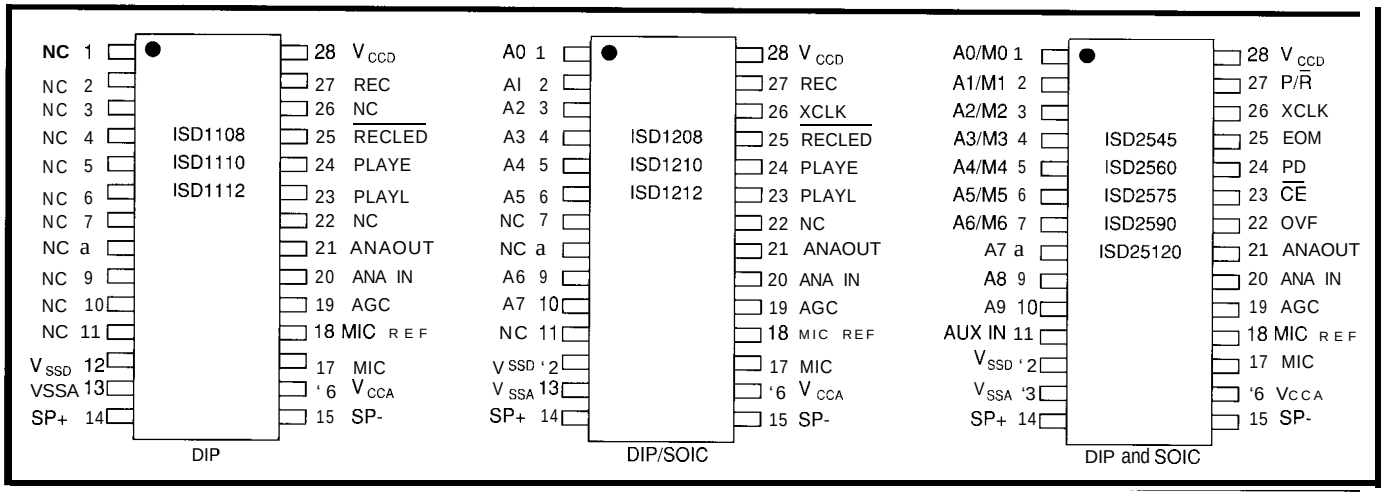


Figure 5—*Pinouts for the ISD voice chip family are similar as you go up the line. The higher-numbered chips sport larger DAST arrays.*

precise clock. The range of clock rates supported is not that wide since it is bounded on the low end by droop in the sampled analog inputs and on the high end by EEPROM write time. Since you probably won't be using it, remember to ground it lest you find, as I did, that leaving it open results in erratic operation (or better put, a bad case of chip laryngitis).

For playback, the analog output is fed through an internal smoothing filter (which is actually the anti-aliasing filter during record mode serving double duty) and amplifier for output at the SP+ and SP- pins. The amplifier output is about 50 mW and can directly drive a 16-ohm speaker. You can use an easier-to-find 8-ohm speaker simply by inserting an appropriate resistor (e.g., 8-10 ohms) in series with the SP- pin.

Experienced '1016A users report that it is important to use a boxed speaker, not just one hanging in the breeze, for best sound quality. Such a setup is loud enough to hear across the room, but further amplification will be required for high-ambient noise levels or for public address applications. As a convenience, an AUX IN pin is provided to take advantage of the on-chip amplifier when it isn't being used for playback. Whatever signal is seen at the AUX IN pin is passed through to the speaker outputs when the chip is not playing back a message.

Putting a system together involves little more than a handful of parts. Figure 2 shows the schematic of a minimal, nonaddressing (i.e., the whole chip is treated as one message) playback and record setup. Believe me, putting audio in an application doesn't get much simpler than this.

It also doesn't get any less expensive since the '1016A is only $6.64 (1000s). Keep an eye out for the 20-second variant ('1020A) at your local Radio Shack. Their version is the ISD 1000 priced at $17.99 each, where it is stocked as part number 276-1325.

TALK IS CHEAP

Feeling the urge to wire something up, I decided to design a simple subsystem that would accept a "message number" via an RS-232 link and then play or record that message. The design combines the '1016A with the new Parallax PIC-based BASIC Stamp computer (Figure 3 shows the schematic and Photo 1 shows the project). Other than these two key LSIs



#132

and an off-the-shelf 8-ohm speaker, the additional component count was just a PLAY/RECORD jumper, and a few resistors and capacitors.

However, I did have to make some compromises since the Stamp only offers eight I/O lines when a complete interface would call for thirteen (twelve for the '1016A and one for the RS-232) lines.

Given that the RS-232 line and the CE' line of the '1016 were required, and needing to trim five lines from the I/O budget, I quickly decided that the least-significant address bits could go since 0.1 -second message address resolution was overkill for me. By grounding A0 and Al, I settled on 0.4-second minimum address resolution. This decision trimmed two lines from the requirement. The next decision was to ignore EOM* by handling message length in software-three down, and two to go. Okay, I can gain one more line if I use a jumper, or a switch, for selecting the play/record mode. Four down, one to go. But, I did want to use PD to minimize power, allowing the whole gizmo to run off of the Stamp's battery and regulator, so I had to keep that one. Yes, as usual, I ended up one line short.

Mildly discouraged, I stared at the ISD data sheet, and by reading the application note again I found a solution. It turns out that CE* really isn't required. Instead, it can be grounded and PD alone used to initiate record and playback cycles. Great! Now I have enough lines on the Stamp to make this idea possible!

The Stamp program (Listing 1) accepts a byte from the host and interprets it as a message number which is played or recorded according to the state of the '1016A P/R* pin. Using the SERIN command, the Stamp waits for a message number from the host and decodes it as a new message {$40–$6F} or repeat count {$30–$3F = 1 to 16 repeats). Using the novel LOOKUP instruction, the message number is translated to a message address (in 0.4-s increments) and a length (0. l-s increments) and the command is issued to the '1016A for the appropriate length of time. Since EOM' isn't used in this design, the only difference between playback and recording is the state of the P/R* pin.

The number and length of messages is defined by the LOOKUP table. In this example, I went the talking DVM route and recorded forty 0.4-second messages listed in Figure 4. Sending, for example, the string "B\VHd@Gcg" will speak the phrase "two hundred forty-eight point oh seven milli amps." Notice how the last message code, the "h" command, plays the whole chip back, which makes it sound like that guy in commercials that talks really fast. You could relax the pace using the 20-second chip ('1020A) and stretch each message length to 0.5 seconds by changing the "duration" constant in the program from 100 to 125.

Notice the shortcuts I took with the microphone input (compare Figure 2 with Figure 3) such as grounding the AGC pin thus fixing the preamp output at maximum gain. I experimented with the AGC, but found it

ECAL Universal Assembly Language DevelopmentSystem

Product Information

ECAL is a complete assembly-language development system that provides all the tools needed to assemble, link, load. run. and debug your project for over 170 processors. By using user-editable control files. the ECAL macroassembler in its full configuration can handle 4-, 8-, 16-, or 32-bit microprocessors with unsurpassed speed and consistency.

Using the familiar DOS-based text windows. you can edit, assemble, set breakpoints. trace execution, watch registers and I/O, and communicate with your target's serial port in separate closable windows. If you prefer to use other tools, with a few keystrokes, ECAL will incorporate your previous work into its consistent andintuitiveenvironment.

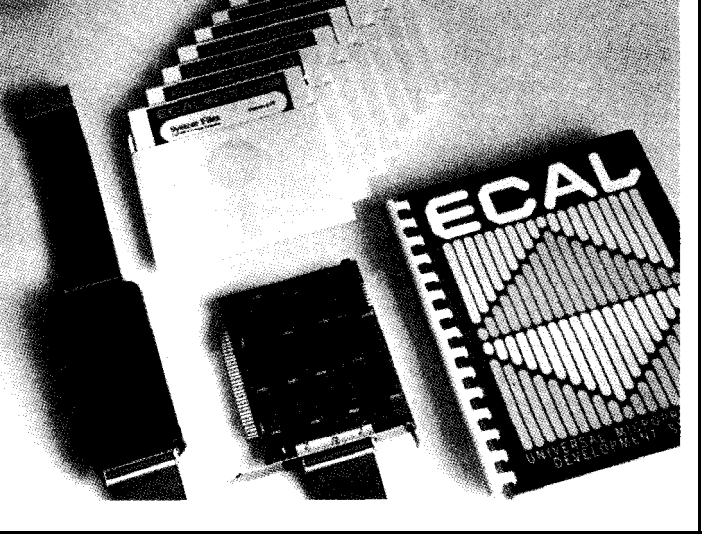
Ordering Information

- 05-0200-010 **ECAL OAS**
- 05-0200-020 **ECAL with EPROM Emulator**
- 05-0200-XXX **ECAL Single Processor**

The free ECAL evaluation program features all of the ECAL tools for all of the supported micros, giving you a true sampling of ECAL development cycle (source and object length limited).

VAIL Silicon Tools sells and supports ECAL and can bundle ECAL **withadditional hardware and software to satisfy your need for economical project development tools.**

- ▶ Alternative to Real-Time Emulator
- ▶ Support for 805 l, 8096, and 186
- ▶ Support for 170t additional processors
- ▶ User control of syntax and instructions
- ▶ Extremely fast assembly-2 Kbytes/sec
- ▶ Integrated split-screen editor or command-line assembly supported
- ▶ Integrated linker/loader
- ▶ Instruction trace and I/O windows
- ▶ Monitor and RS-232 corn. windows
- ▶ Single micro processor versions available
- ▶ Optional EPROM emulator and programmer
- ▶ Source-level debugger



Vail Silicon Tools

Contact
Vail Silicon Tools
692-A S. Military Trail
Deerfield Beach. FL 33442
Tel: (305) 570-5580
Fax: (305) 428-1811

#133

disconcerting for short messages (messages in the 0.4–0.8-second range) since the gain didn't have time to stabilize. The only component that seemed to have a dramatic effect on recording quality was the 22-µF low-frequency bypass capacitor.

Even with my simplified design, I'd say the results I achieved back up ISD claims of "telephone grade" audio. Messages were loud and clear but there was a slight hiss (both likely due to grounding AGC). Since the volume was surprisingly high, I implemented a home-brew "low-pass filter" by sticking the speaker in a cardboard box. The result was much mellower, but still quite clear.

Remember, mainly "playback" applications (i.e., factory vs. field programmed ones) can achieve higher quality by recording and editing with audio gear (including a sound-card-equipped PC) and using the ANA IN pin for programming. For best results, consider the official ISD evaluation unit/programmer-the ES001B, which sells for a reasonable $299.

THE LONG AND SHORT OF IT

ISD expects to fill out the lineup shortly with both shorter and longer duration parts. For pinout information on these parts, see Figure 5.

But for those of you who want to play with parts right now, you may want to play with what's already shipping. The 64K ISD1100/1200 series offers 8–12-second capacity (8–5.3-kHz sampling rate) at only $5.78 (1000s). The major difference between the 1100 and 1200 is the latter supports direct addressing while the former doesn't.

Expected shortly are members of the ISD2500 family. These devices will feature a 480K DAST array-nearly four times the capacity of the '1016A These chips will be priced to go ($14.94/1000s), and will store from 45 seconds to 2 minutes of audio depending on the sampling rate [e.g., 60 seconds at 8 kHz). The pinout is essentially the same as that of the '1016A, with the addition of two more address lines and an OVF* pin that facilitates cascading.

So, don't hesitate to take advantage of these unique talking chips; just make sure they mind their manners. ❑

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

Information Storage Devices, Inc.
2841 Junction Ave.
San Jose, CA 95134

Parallax, Inc.
6359 Auburn Blvd., Suite C
Citrus Heights, CA 95621
(916) 721-8217
Fax: (916) 721-1905
BBS: (916) 721-9607

I R S

416 Very Useful
417 Moderately Useful
418 Not Useful

MOVE OVER INTEL

MICROMINT SOURCES

80C52 CMOS BASIC CHIP

Micromint has a more efficient software-compatible successor to the power-hungry Intel 8052AH-BASIC chip. The 80C52-BASIC chip was designed for industrial use and operates beyond the limits of standard commercial-grade chips. Micromint's 80C52-BASIC chip is guaranteed to operate flawlessly at DC to 12 MHz over the entire industrial temperature range (-40°C to +85°C). Available in 40-pin DIP or PLCC

80C52-BASIC chip	$25.00
OEM IOO-Qty. Price	$14.50
BASIC-52 Prog. manual	$15.00

MICROMINT, INC.
4 PARK ST., VERNON, CT 06066
TO ORDER CALL
1-800-635-3355

#119

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.

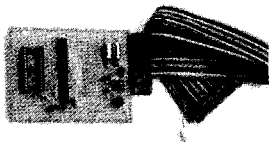
MVDISK1 64k.....$75
MVDISK2 360k....$150
MVDISK3 1.44m..$195

Quantity discounts!

DOS IN ROM!

$75





$95 EPROM PROGRAMMER

-DO ROMS FOR SBC OR MVDISK
-PROGRAM 2764/27010 EPROMS

WORLDS SMALLEST PC !!!

ROBOTS ALARMS RECORDERS DOS

THREE EASY STEPS:
1. Develop on PC
2. Download to SBC
3. Burn into EPROM

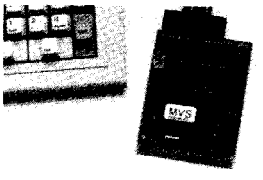
$27 1K QTY
$95 SGL QTY

-2 PARALLEL -LCD INTERFACE
-3 SERIAL -KEYBOARD INPUT
-PC TYPE BUS -REAL TIME CLK
-BIOS OPTION -BATTERY OR 5V

FREE SHIPPING IN U.S.

5 YEAR LIMITED WARRANTY

8088 SINGLE BOARD COMPUTER



.MVS Box 850 Merrimack, NH (508) 792 9507

#135

The Art of Battery Management

EMBEDDED TECHNIQUES

John Dybowski

More and more equipment these days is battery operated. Battery technology continues to improve, but to maximize a battery's useful life, the charge management circuitry must be properly designed.

You've managed to embed all the requisite functions into your design. Now it's time to consider your options for power supplies. Hopefully, if you decide to embed the power source as well, you will pause and give serious consideration to some of the outlying functions associated with this decision. Sometimes you can get away with just a rechargeable battery, a diode, and a resistor. Often you will need more functionality than this simplistic approach can provide. Each battery system can be as unique as the product it is supposed to power, and the choice of a battery system often has implications beyond just providing a source of power to your product.

Too many battery-powered instruments rely on extremely primitive forms of battery management: You know the battery needs charging when the device stops operating; you know you have to replace the battery when it can't hold a charge anymore. How's that for a system that promotes happy users?

As an example, consider the case of the weekend camcorder-jockey. Following the usual charging interval, he snaps the battery pack into the camera. The battery indicator shows all systems are go. The awareness that this indicator means next to nothing may eventually surface, because after ten minutes the low-battery light comes on. Obviously there's something wrong here. If this person is lucky enough to know someone who is familiar with electronics, he probably could persuade them to look at the offending battery. With some luck (and several discharges followed by slugs of charging current) the battery may come back to life-then again, maybe not.

This intolerable state of affairs is very inconvenient at best and as a rule usually happens at the worst possible time. Most people feel that given the cost of modern consumer gadgets, they should be designed properly to begin with. "Properly designed" means different things to different people but even the most undemanding want to know when the battery is on its way out.

Now, the battle for the decent battery supply has usually been fought incrementally and has commonly been conducted based on product performance and product differentiation. For example, designer A decides to use a battery to power his product. Designer B did one better and uses a rechargeable battery, only he was a cheapskate and used a diode and a resistor for the charger. Designer A notices designer B's shortcoming and came back with a better constant-current charger. Designer B has a bright idea and strikes back with a two-stage fast charger and provides three LEDs that presumably show the amount of battery capacity that is available. Designer C was watching designer A and designer B and decides to enter the game. His solution to everything is putting bigger cells in his aftermarket battery on the assumption that bigger is better.

To an extent, this progression is a case study in the natural evolution of products and cannot be considered unreasonable. But your perspective still boils down to where you draw the line on what can be called a good product. This is especially true in the very vital power source because shortcomings in the supply are immediately perceptible. The heat is on to provide better battery supply systems since electronics have evolved to such a degree of miniaturization that the battery is usually the bulkiest and heaviest component in the system. The customers have their expectations when it comes to battery-operated gear and they are merciless. Rightly so,

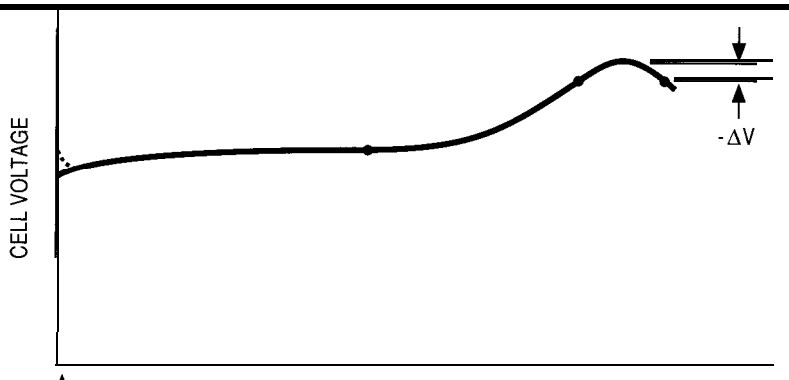
since they know their needs and they expect you to know them too!

It's not adequate to keep increasing the battery size just because more capacity is needed. There's no place in today's market for battery status indicators that tell you the battery just went flat. A brute force mentality and slipshod design practices seem to be finally dropping out of vogue.

The design of electronic systems that consider battery operation alleviates this problem somewhat. The rules for designing for low power are as follows: apply low-voltage CMOS circuitry, use selective power control to various electrical subsystems, and place more emphasis on thoughtfully designed power supplies.

Attention is now being focused on the battery side of the equation also. These changes involve the development of new battery chemistries, the improvement of older ones, comprehensive battery charging and conditioning regimes, and the accurate gauging of the actual usable battery capacity.

Combining intelligence in the battery management strategy and a sound power management approach in the electronics along with one of the newer battery technologies, such as Nickel-Metal Hydride, can yield truly impressive results. The consequence of such an approach is evidenced by the level of performance attained by notebook computers and pocket cellular phones. I'll be covering battery management and conditioning techniques in due time. However, it's unavoidable to recap some battery basics before I do so.



METHOD OF CHARGING	CHARGE RATE: MULTIPLES OF C-RATE	CHARGE RATE: FRACTION OF C-RATE	RECHARGE TIME* (HOURS)	CHARGE CONTROL
STANDARD	0.05C 0.1C	C/20 C/10	36–48 16–20	NOT REQUIRED
QUICK	0.2C 0.25C 0.33C	C/5 C/4 C/3	7-9 5-7 4–5	NOT REQUIRED
FAST	C 2c 4c	C 2c 4c	1.2 0.6 0.3	REQUIRED
TRICKLE	0.024 1C	C/50–C/10	Used for maintaining charge of a fully charged battery	

*RECHARGE TIME = STANDARD TIME TO FULLY CHARGE A COMPLETELY DISCHARGED BATTERY AT 23° C

Figure I--The *classic Nickel-Cadmium fast-charge profile (top) shows a characteristic bump and slight drop in voltage when fhe cell is fully charged. This V may be used to sense full charge. NiCd cells are typical/y charged using one of four methods (bottom), each of which has its uses.*

COMMON CHEMISTRY

Three popular battery technologies in common use are Nickel-Cadmium, Nickel-Metal Hydride, and Lead Acid. Each has its specific uses based on size, energy density, and, perhaps most importantly, cost. I'm sure most users of NiCds would be more than happy to step up to NiMH batteries, but often cost is a deterrent to their doing so. Rechargeable batteries are differentiated by several key parameters such as storage capacity, nominal voltage, cutoff voltage, and maximum tolerated rate of charge.

Amp hours (or milliamp hours) is the unit of measure used to describe the battery's storage capacity. The C rate is defined as the rate of amps (or milliamps) equal to the capacity rating of the battery. The use of this capacity yardstick (along with its multiples and fractions) serves to expedite the discussion of different types of batteries and a range of battery sizes.

This battery storage capacity is dependent on the discharge rate. When the energy is withdrawn from the battery at a faster rate, there will be less effective capacity. That is, the usable capacity is time dependent and because of this, a known discharge rate must be implied to properly describe battery performance. NiCd and NiMH cell capacity is usually specified at a five-hour discharge rate. Because of this consistency, published capacities for these types of cells generally correlate among the different manufacturers. Since the five-hour rate is almost universally used, it is seldom given in the battery specifications. The same is not true of lead acid batteries. Due to the dissimilar applications for lead acid batteries, the capacity ratings are given for different rates of discharge.

Generally, lead acid batteries can be categorized into three broad divisions according to their usage: standby, cyclic, and motor starting. Batteries used in standby service are usually held on a trickle charge for long periods of time and may be called upon to deliver power for periods ranging from several hours to several days. Examples of this type of service would be backup power for emergency lighting systems or electronic equipment. Batteries in cyclic operation can supply power for several hours before charging is required. Cyclic operation would be characterized by cordless

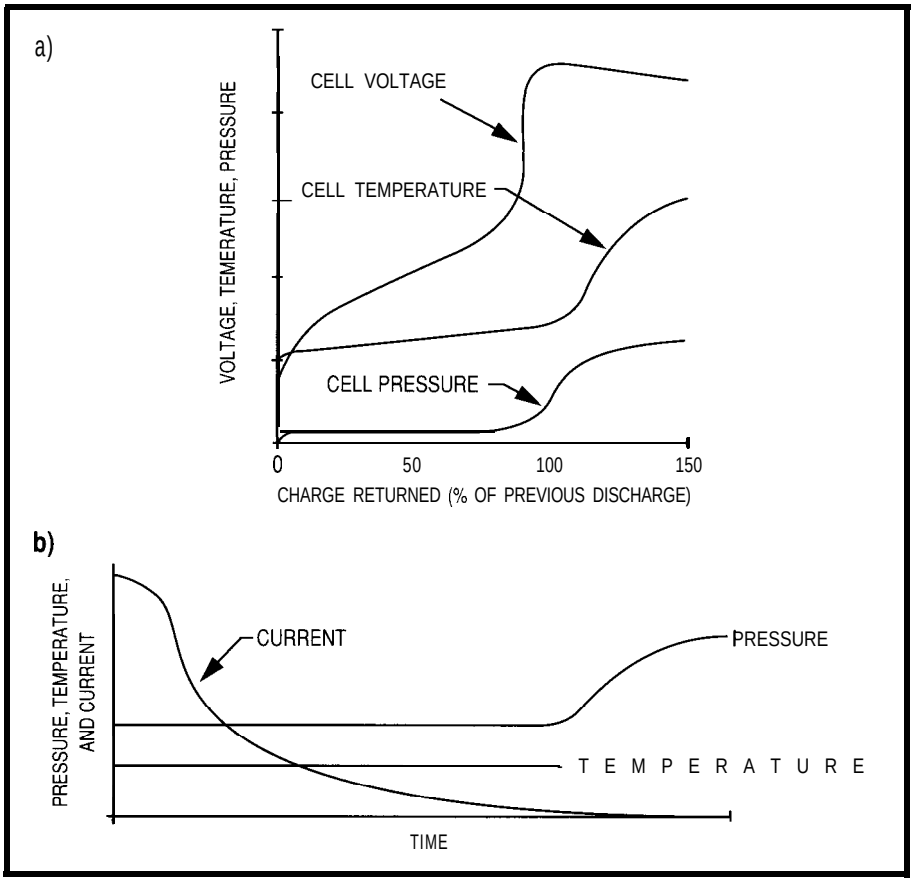


Figure 2-a) The *typical relationship of cell voltage,* pressure, and *temperature during* constant-current *charging. b) The typical relationship of cell current, pressure, and temperature during constant-voltage charging.*

power tools. Motor starting applications require high peak currents that may discharge the battery over a period of a few seconds.

Because of these different uses, lead acid batteries are rated based on their expected application. Batteries intended for standby usage are generally rated at a 20-hour discharge rate. Cyclic batteries are usually specified at an 8- or IO-hour rate. The nature of motor start and other short duration high discharge applications dictates that the l-hour rate be applied for these. In all these cases, the discharge rates are clearly stated in the battery specifications.

Nominal voltage is another important restriction which determines how many cells are required to achieve the total desired battery voltage. NiCd and NiMH cells nominally supply 1.2 volts whereas a lead acid cell produces 2 volts.

The terminal voltage decreases as charge is withdrawn from a battery. The cutoff voltage is the minimum voltage the battery can be taken down to before cell damage occurs. This voltage varies among the various battery compositions. It's important to note that the cutoff voltage determines the amount of energy that can be drawn from the battery. Should you have a situation where the device that is operating from the battery cannot function down to the cutoff voltage, all of the battery's useful energy will not be depleted. To determine the amount of energy that can be drawn from the battery in such applications, you must consult the discharge curves that plot terminal voltage versus discharge rate versus time.

Charging is accomplished by applying a current of proper polarity to the battery. This can be a pure DC current or can contain a significant ripple component. This ripple generally results from the use of a rudimentary half-wave or full-wave rectifier which represents the baseline of battery charger schemes. The situation can (and should) get more complex.

Constant voltage or constant current charging illustrate two variations that are frequently applied to deliver energy faster to batteries. With higher charge rates, a variety of feedback techniques can be used to determine the point at which the charging current is to be reduced. Many forms of temperature sensing, voltage sensing, or current sensing (or combinations of any and all of these) are used in this regard. This reduction in charging current becomes necessary to prevent damage to the battery and is not only dependent on the battery chemistry but on the specified charging capabilities of the batteries. NiCd batteries, for example, can be obtained with capabilities that can be defined as Standard Charge, Quick Charge, or Fast Charge.

Overcharge is the continued charging of a battery after the battery has reached its maximum state of charge. Batteries are designed to handle continuous overcharge at their cell specification rate. This rate is the determining factor that indicates when charge control is required to drop the charging current down to a lower rate. The Standard Charge, Quick Charge, and Fast Charge NiCds that I mentioned not only have different restrictions on their maximum charge rates but also on the amount of continuous overcharge they can handle.

NICKEL CADMIUM BATTERIES

Perhaps the most widely used rechargeable battery chemistry, NiCd batteries have become the established battery system for many applications. Possessing excellent high-drain discharge capabilities and cycle life characteristics of 500 to 1000 cycles along with decent self-discharge properties, NiCd batteries are used for everything from cordless power tools to memory backup applications. Although NiCd batteries use a fairly stable technology, performance improvements continue to be made. A 600-mAH capacity from an AA cell was typical a couple of years ago and now you can get 850-mAH capacity at a reasonable cost and from the same size battery.

Several different charging schemes are commonly applied to NiCd batteries. Naturally, the method used

determines the complexity of the charging circuit; constant-current methods are generally preferred. Furthermore, different types of batteries are produced that are specifically designed to tolerate higher charging rates or that have the capability of operating at higher temperatures. Some of these allow simple charging circuits whereas others require some form of charge control that will fall back to a lower rate of charge once a full charge is attained.

Detection of a full charge condition is often based on the ramp in heating that occurs at the end of charge cycle. This condition can be sensed by directly monitoring the battery temperature using a sensor housed in the battery. This method is not without problems since the temperature sensor will usually contact only one or two cells, leaving the remainder out of the picture.

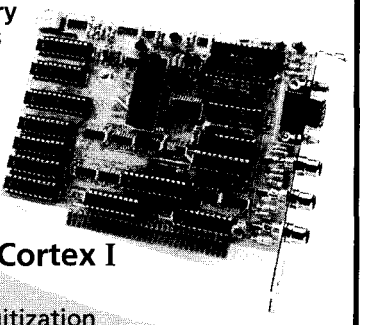
A popular derivative of temperature sensing is the ***negative delta voltage*** method. This voltage phenomenon occurs at the end of charge cycle following a steady increase in voltage when a battery's temperature increase causes a decrease in internal resistance that, in turn, results in a drop in voltage. Since all the cells have a say in the charge termination decision, this proves to be a reliable indication of the overall state of charge of the battery. Figure 1 illustrates this voltage profile and also contains some pertinent figures on the various charge rates that are commonly used for the different NiCd battery types.

When a NiCd battery is charged, a small amount of energy goes into converting active materials into an unusable form. ***Charge acceptance*** is the term used in describing the general charging effectiveness. This defines the charging efficiency and the amount of discharge capacity that can be put into the battery.

NiCd batteries accept deliverable charge at different rates depending upon factors such as the state of charge of the battery, charge rate, and temperature. Battery history, which refers to the type of usage the battery has been subjected to, also figures in the charge acceptance criteria.

Video Frame Grabber

- $495 Including Software with "C" Library
- Half Slot Card for Compact Applications
- Real Time Imaging with Display Output
- 8 Bit (256 Gray Levels)



Cortex I

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.

Call today for volume pricing or to discuss your application.

ImageNation Corporation
Providing Imaging Solutions

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

FAX (503) 643-2458 • (800) 366-9131

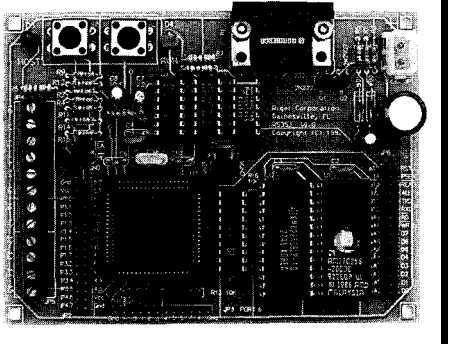
#136

NEW 8031 FAMILY PRODUCTS from RIGEL

FUZZY-LOGIC CONTROL CODE GENERATOR

FLASH (Fuzzy-Logic Applications Software Helper) generates MCS-51 language subroutines to perform fuzzy-logic control tasks from a high-level description of fuzzy rules written with linguistic variables. An extensive tutorial and illustrative examples **are** provided. ($100.)

R-5355 / READS TRAINER



READS (Rigel's Embedded Applications Development System) and the R-535J board constitute a complete hardware/software development and debugging system in one user-friendly menu-driven environment which runs on an IBM PC host. Programs in the MCS-51 language may be written, edited, assembled, downloaded and debugged without leaving the integrated environment. The R-535J board uses the powerful 80C535 microcontroller. R-535J / READS with User's Guide on disk and example programs is priced at $150, $130 as a kit.

8031 FAMILY MICROCONTROLLERS EXPERIMENTER'S GUIDE

This 300+ page textbook covers the MCS-51 assembly language, using the on-chip facilities of the 8031 family microcontrollers, and software / hardware experiments. Programming nuggets are given for each instruction and each operating mode of the family. Features of the SAB80C535 are highlighted. ($30.)

RIGEL CORPORATION

PO BOX 90040, GAINESVILLE FL, 32607 (904)373-4629

#137

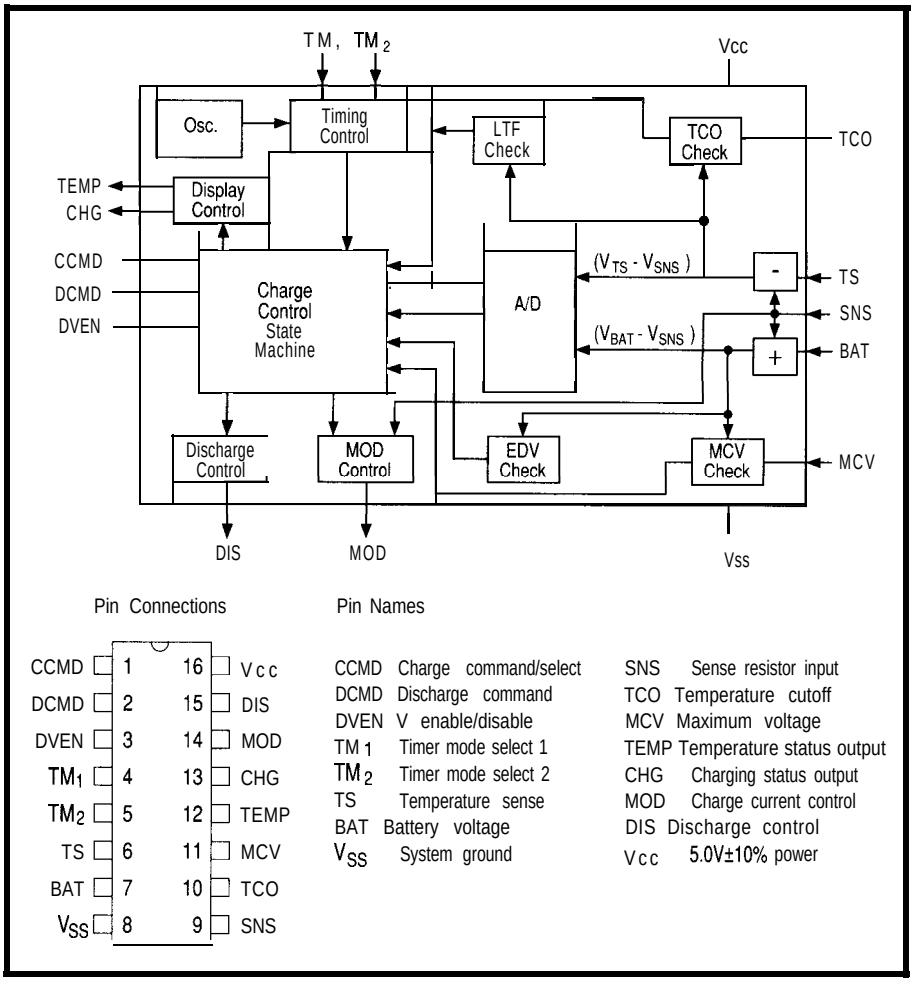


Figure *3-The Benchmarq bq2003 can be used to monitor and control the charging of numerous kinds of rechargeable batteries. The chip handles one-, two-, and three-stage charging and has outputs that directly drives LEDs to show the current status.*

Both the charge efficiency and the actual capacity of a NiCd battery are reduced when charged at high temperatures compared with those charged at room temperature. A couple of examples will help to put this in concrete terms, with a cell temperature of 45°C and a charge input of 200% of nominal capacity, no more than 70% of nominal capacity will be attained. Capacity levels become adversely affected with increasing temperatures. With a battery temperature of 60°C and a charge input of 200%, the actual battery capacity will be no more than 45%. Charging the same battery at temperatures ranging 0–25°C, the charge acceptance is much better, and a charge input of 160% would yield 100% capacity.

Charge rates also influence charge acceptance. Charge rates below 0.05*C* will not allow the battery to ever achieve its full capability. Efficiency is enhanced by rates higher than 0.1 C for battery types that can accept charging at higher rates. Using a fast-charge battery, charging it at 1 C approaches 100% of nominal capacity at an input of 120%.

NICKEL METAL HYDRIDE BATTERIES

NiMH batteries offer a significant increase in power density over NiCd batteries. This increase comes at a significant cost, however, since these are about twice the price of a NiCd battery of the same size. Looking again at a modern AA NiCd battery which is available with a capacity of 850 mAH, a NiMH battery of the same size typically has a capacity of 1200 mAH. These batteries are finding uses in applications where longer run time is a key competitive feature and where the battery price represents a relatively small portion of the product cost.

Although there are some important differences to consider, NiMH batteries perform in a similar manner to NiCd batteries.

Generally, the same charge method used for NiCd batteries can also be used for NiMH batteries, although certain control limits may have to be changed to ensure proper operation of two-stage chargers. The most notable difference between the charging characteristics of these two batteries is the lack of a pronounced decline in cell voltage (negative delta voltage) at peak charge. If you decide to use a voltage-sensing system to charge NiMH batteries, you would be wise to carefully evaluate the pertinent circuit parameters. It might not be a bad idea to dismiss a voltage-monitoring approach completely and instead consider using a temperature-profile cutoff method instead.

Service life is also similar between NiCd and NiMH batteries, although self-discharge is presently still slightly higher and the NiMH battery is not quite up to the high discharge capabilities of the NiCd.

LEAD ACID BATTERIES

Lead acid batteries find uses in many of the same application areas served by NiCd batteries. Starved electrolyte sealed lead batteries are one of the more advanced forms of lead batteries in use today. With low self-discharge characteristics and a lifecycle approaching ten years under float conditions, this battery chemistry has obviously continued to evolve along with the other technologies that I've already described.

When considering which charge method to use for this kind of battery, you must give attention to the time available for charging, the temperature extremes the battery will operate under, the number of cells required, and the way the battery will be used. As with the other battery chemistries, there are alternate methods that can be selected for charging lead acid batteries. Constant-voltage charging is often used and is usually considered the conventional charging method for charging lead acid batteries. Constant current, taper current, and other

variations may also be used with good results.

Using constant-voltage charging, the charger holds a uniform voltage regardless of the battery's state of charge. Here, the charging current varies depending on the difference of potential between the input voltage and the battery voltage, thus the battery will draw greater current when it is at a discharged state. As charging continues, the battery voltage rises and the charge current diminishes. Current limiting is not required for many types of lead acid batteries using this charging method. Constant voltage charging can be used for fast charging as well as in float charge applications. Since lead acid batteries have low internal resistance, high currents-up to 4*C*—will flow into a discharged battery if the current is not limited by the charger's current supply.

As a result of the large current flow involved, up to 70% of the previous charge may be returned in the first 30 minutes of charging. In this case, though, as the battery voltage increases quickly, the resulting reduction in potential difference causes the current to drop off significantly. As a result, although the battery regains a significant portion of the charge quickly, a prolonged period will elapse before a full charge is realized.

A typical float maintenance voltage would be in the range of 2.30-2.40 volts per cell. If you wanted to recharge a battery in 2-16 hours, then 2.45–2.50 volts per cell would be required. Anyhow, it's not a good idea to drive a source voltage over 2.40 volts per cell into the battery once a full state of charge is attained.

When applying fast constant-voltage charging schemes, you must reduce the charge rate when the battery becomes fully charged. This condition could be handled by sensing the current decay when the battery was almost fully charged and then switching to a lower constant-voltage level. A simpler method would be to just time the duration during which the high constant voltage was applied before dropping to a reduced constant-voltage float level.

Finally, higher battery temperatures increase the chemical reactions taking place in the battery. Because of this increase, less charging voltage is required to fully charge the battery at high temperatures in a given length of time. On the other hand, it takes a higher voltage to fully charge a battery at low temperatures in the same amount of time. In either case, when operating in the temperature range of 5–45°, it's generally not necessary to worry about changing the charging voltage. However, when operating out of this range, a negative temperature coefficient from 2.5 mV to 3 mV per degree Celcius is recommended.

Constant-current charging is, in many applications, the best means of restoring battery capacity quickly without adversely affecting battery life. This charging method is especially effective when several cells or batteries are charged in series. The idea behind constant current charging is to apply a uniform current to the battery regardless of the battery voltage (the state of charge). Overcharging lead acid batteries does become an area of concern when the constant current charging method is applied.

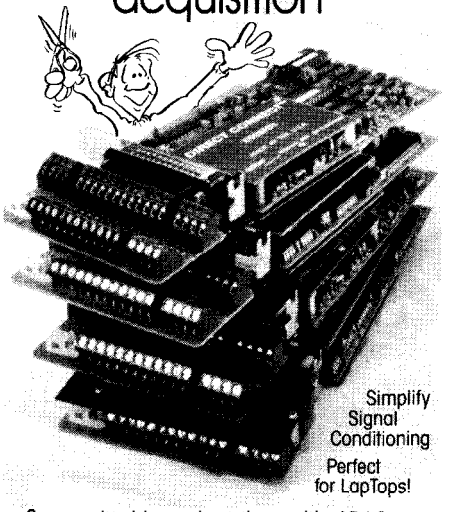
At a low state of charge, lead acid batteries are tolerant of high charge rates. As full charge is approached, manufacturers recommend cutting back the charge rate to prevent unpleasant situations such as venting of internal gasses. If you're charging at rates below 1 C, then you most likely can get away with just a simple single-stage charger. Charge acceptance of lead acid batteries is good at low constant current rates. Charging at a 0.01 C rate, charge acceptance is about 90% at 25°C throughout most of the charging period. Although charging at such low rates will extend the battery life, realize that the recovery time for a dead battery can be up to 100 hours at such low current levels.

If battery capacity must be recovered quickly, a two-stage constant-current charger makes the most sense. When the battery is at a low state of charge, hit it with a reasonably high current falling back to a trickle current when the battery approaches a state of full charge. The voltage across

PC/XT/AT/386 Users!

DIRECT CONNECT

A time-cutting approach to plug-in data acquisition



Save volttable project time with ADAC's DIRECT CONNECT™ data acquisition modules. FREE DIRECTVIEW™ Board Tutorial and Data Acquisition software means a shorter learning curve and quicker results. CALL FOR A FREE COPY

Thermocouple 8 Channel A/D Board $650
- Direct Connection to Thermocouple wires and shields
- 0.1°C resolution
- *Software selection of J,K,T,R,S,&B

Strain Gage 8 Channel A/D Board $795
- Direct Connection to 3 & 4 wire Strain Gages
- 1 µstrain resolution
- *Quarter. Half & Full bridge completion
- On-board excitation voltage

RTD 8 Channel A/D Board $795
- Direct Connection to 4-wire RTDs
- 0.02" C res., 100 ohm platinum
- 1 mA current source per channel

High Resolution 8/16 Channel A/D Board $895
- 16 bit A/D resolution
- 15 kHz throughput (50 kHz option]
- DMA
- 8 lines digital I/O
- 3 channel counter/timer

Multifunction 8/16 Channel A/D Board $595
- 12 bit A/D resolution
- 25 kHz throughput
- DMA, Prog Gain
- 8 lines digital I/O
- 3 channel counter/timer

Many other models available, Call:

1-800-648-6589

adac corporation

70 Tower Office Park, Woburn, MA 01801
FAX (617) 938-6553 TEL (617) 935-6668

#138

lead acid batteries is a pretty good indication of the state of charge. When charged at constant current, there is a definite voltage increase as the battery becomes 90% charged. Voltage sensing is a convenient and simple method of detecting when a high charge rate should be cut back. Use a level of about 2.50–2.65 volts when charging at a 0.1 C rate to terminate fast charging and switch to a trickle level of about $0.002C$. Figure 2 shows some of the charging attributes using constant-voltage and constant-current methods.

FAST CHARGE HELP

Some of you have probably concluded by now that the proper care and feeding of rechargeable batteries is no small feat. This is particularly true when it comes to fast charging, especially with NiCd or NiMH batteries. The situation is made more difficult when employing sophisticated capacity determination techniques to conclude the fast charge portion of the sequence. The whole thing is made more intricate by the fact that you will most likely want to employ auxiliary cutoff mechanisms to prevent battery destruction if the primary charge determination method fails. Furthermore, it would be wise to detect an out-of-spec battery temperature and then to withhold the application of the charging current until the battery was at an acceptable temperature. Finally, secondary conditioning functions such as discharge before charge are very useful to help extract maximum battery capacity and should be considered if you're crafting a deluxe battery management system.

Previously, such functions were only attainable using a microcontroller with an analog-to-digital converter, a power source (usually constant current), various switching transistors, and a small pile of discrete components. The most costly ingredient—and most difficult to get right, of course-was the firmware. Considering the amount of work involved in such a design undertaking, it's not surprising that many people found that a resistor and diode didn't look so bad after all. Luckily, things change for the better sometimes. Dedicated ICs are available now that perform all the functions I just mentioned and cost just a few bucks. There are a lot to select from, but I found the bq2003 fast charge IC from Benchmarq to be truly one of the better parts on the market.

Usable as an efficient switched-mode constant-current source, the bq2003 can operate as a frequency-modulated controller for charging current. Alternatively, the bq2003 can be used with a transistor or SCR to gate an external current source, usually in a pass configuration. Monitoring of temperature, voltage, and time throughout the fast charge cycle allows termination of fast charging using delta temperature/delta time, negative delta voltage, maximum temperature, maximum time, or maximum voltage. Delta temperature/delta time and/or negative delta voltage are generally used to make the decision of when to cut off the fast charge to NiCd batteries. When used

ATTENTION SUBSCRIBERS

Please direct all subscription requests and questions to our NEW Customer Service Department:

THE COMPUTER APPLICATIONS JOURNAL
P.O. BOX 7694
RIVERTON, NJ 08077

TEL: (609) 786-0409
FAX: (203) 872-2204

The Ciarcia Design Works

Does your big-company marketing department come up with more ideas than the engineering department can cope with? Are you a small company that can't afford a full-time engineering staff for once-in-a-while designs?

Steve Ciarcia and the Ciarcia Design Works staff may have the solution. We have a team of accomplished programmers and engineers ready to design products or solve tricky engineering problems. Whether you need an on-line solution for a unique problem, a product for a startup venture, or just experienced consulting. the Ciarcia Design Works is ready to work with you. Just fax me your problem and we'll be in touch.

Remember...*a Ciarcia design works!*

Call (203) 875-2 199 Fax (203) 875-8786

with NiMH batteries, the delta temperature/delta time method proves to be a very reliable fast charge termination method. Requiring a single thermistor to monitor the rate of increase of temperature for contacted cells, this method is more efficient than a typical two-thermistor arrangement. Such an arrangement, used in the standard delta temperature method, uses one thermistor at the battery and a second to monitor the ambient temperature. For safety, backup termination based on absolute temperature, maximum time, and maximum cell voltage is available.

Configurable as a one-stage, two-stage, or three-stage charger, the bq2003 provides the level of sophistication required to suit the particular application. Two-stage charging consists of an initial fast charge followed by a continuous trickle charge that is set by an external current limiting resistor. With three-stage charging, the initial fast charge is followed by a topping off charge at one eighth the rate of the fast charge rate. Following this interval, an externally controlled trickle charge of about $0.025C$ can be applied as a minimal charge sustaining level.

The discharge before charge feature is switch selectable and provides for automatic discharge of the battery to a nominal one volt per cell. Once this level is reached, the fast charge is automatically started. Discharge before charge provides conditioning services useful to prevent the dreaded memory effect that commonly afflicts NiCd batteries and to provide capacity calibration capabilities.

Direct LED control is provided on-chip to show charge status conditions such as charge pending, discharge, fast charge in progress, charge complete, and charge aborted. Figure 3 depicts the pinout of the bq2003 fast-charge IC.

ALL CHARGED UP AND READY TO GO

Next month I'll apply the bq2003 to some real charging tasks and wrap things up with a discussion of battery conditioning. I'll also present a complete, although somewhat intimidating, battery management circuit based on a chip that possesses even more capabilities than the highly integrated bq2003. Until then, hold off on those resistor and diode chargers. ❑

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

CONTACT

Benchmarq Microelectronics, Inc.
2611 West Grove Drive, Suite 109
Carrollton, Texas 75006
(214) 407-0011

I R S

419 Very Useful
420 Moderately Useful
421 Not Useful

Device Programmers
Connects to PC parallel printer port
48 PIN ZIF
EMP-20
$449.95
SA-20 8 Gang Eprom
20 X 4 Line LCD Display
$750.00
Key Keypad
PB-10 Internal Card for PC
40 PIN ZIF
2 ft. Cable
$139.95
• Easy to use software, on-line help, full sceen editor
• Made in USA
• 1 & 2 Year Warranty
• Technical Support by phone
. 30 day Money Back Guarantee
• FREE software upgrades available via BBS
• Demo SW via BBS (EM20DEMO.EXE) (PB10DEMO.EXE)
• E(e)proms 2716 8 megabit, 16 bit 27210-27240, 27C400 & 27C800,
• Flash 28F256–28F020, (29C256–29C010 (EMP-20 only))
• Micros 8741A, 42A, 42AH, 48, 49, 48H, 49H. 55, 87C51, 87C51FX, 87C751, 752
• GAL, PLD from NS, Lattice, AMD-16V8, 20V8, 22V10 (EMP-20 only)
FOR MORE INFORMATION CALL
NEEDHAM'S ELECTRONICS, INC.
4539 Orange Grove Ave.
Sacramento, CA 95841
(Monday-Friday, 8 am-5 pm PST)
MasterCard VISA
C.O.D.
(916) 924-8037
BBS (916) 972-8042
FAX (916) 972-9960
#139

Fast Code Relief!
•Supports 8051 and Dallas DS5000 families
•Program in easy industrial BASIC (similar to GW-BASIC)
•Artificial Intelligence engine assists you with setup tasks
•Ring buffered interrupt driven serial I/O
•True time-based and external interrupts
•Supports unlimited in-line assemby code
•Your program can coexist with BASIC-52
•Same day shipping and 30 day money-back guarantee
•Competent technical support
"A potent development combination..." EDN January 20, 1992
Systronix Inc.
555 South 300 East
Salt Lake City, Utah 84111
TEL: 801-534-1017
FAX: 801-534-1019
#140

PATENT TALK by Russ Reiss

Often the patent searching process results in a collection of "nitty-gritty" entities which seem to apply only to the detailed, inner workings of specialized devices. While these "gems" may be invaluable to someone intimately involved in a particular field, they often don't relate to a broader audience. This month, however, I've come up with a collection of recent patents which not only appear to be immediately practicable, but also could result in products that might affect our everyday lives while at home, traveling, or at work.

The first is something that I have often wished for myself: a method of providing the airline passenger with greater information, both about his flight in progress as well as his itinerary. Abstract 1 proposes a flight-worthy electronics package that links into the aircraft navigation system, the airline database, and the passenger display system. It provides the passenger with real-time technical information (complete with maps) regarding the flight path, the current location, temperature, speed, altitude, and so forth. Additionally, it presents useful information about the destination airport such as the location of the arrival and other gates, as well as connecting flight information showing flight numbers, departure gates and times, and destinations. As aircraft design moves more toward personalized video displays for passengers, this represents one of many kinds of useful information that could be provided to the passenger. How far away is the day when each seat will be provided with a full multimedia personal information system (a.k.a., "personal computer")?

On another front, electronics and microcomputer technology have already affected the way we pay for commodities and services. These range from automated teller machines, to telephone-linked credit card authorization devices, to electronic shopping and electronic funds transfer for paying bills. The "electronic key" device of Abstract 2 presents yet another option. It carries the security key concept we've seen on PCs a step further in order to control access to a wide range of commodities. As discussed in the abstract, an EEPROM-based device would contain rate and quantity access information, on the basis of which the holder of the key would be permitted to use the metered resource. As presented it appears to apply specifically to metered utilities such as electricity, natural gas, and water. However, the broader concept in which debit cards control our access to gasoline stations, laundromats, movie theaters, sporting events, highways, and on and on and on...surely appears to be in our future.

While the device presented by Square D Company in Abstract 3 might not affect the daily lives of everyone, it would have profound effect on the operation of sundry manufacturing operations. Perhaps one of the most noted

1

Patent Number	4,975,696
Issue Date	1990 12 04
Inventor(s)	Salter, Richard J., Jr.; Long, George S., III
State/Country	CA
Assignee	Asinc, Inc.
US References	2,391,469 2,484,462 3,088,107 3,706,969 3,715,716 3,855,571 3,886,515 3,944,998 3,988,735 4,041,529 4,086,632 4,138,726 4,163,387 4,297,672 4,360,876 4,428,057 4,489,389 4,528,552 4,584,603 4,598,292 4,647,980 4, 674,051 4,740,838
US Class	340/973 340/990 358/103 364/460
Title	Real-time flight and destination display for aircraft passengers
Abstract	A flight-worthy electronics package that connects into the airborne electronics of a passenger aircraft and to that aircraft's passenger visual display system provides passengers with a variety of information real-time displays. Information that may be displayed, as desired, is flight information such as ground speed, outside air temperature, or altitude. Other information that may be displayed is a map of the area that the aircraft is flying over at any particular moment. Perhaps most useful to the passengers is destination information. Besides being able to display a chart of the terminals with all its aircraft gates, and identifying the gate at which the aircraft will be arriving, connecting flight information listing flight numbers, times, gates, and destination may be displayed in conjunction with the terminal chart. A flight-worthy electronics package causes these displays to appear automatically on a real-time basis as determined by the programming of the system.

PATENT TALK

drawbacks of the proliferous programmable logic controller (PLC) is its poor ability to interact with personnel and information systems. This patent links the ubiquitous spreadsheet, with its inherently natural means for assembling and presenting data, to the equally ubiquitous PLC found in nearly every automation situation. This appears to be a winning combination of talents and capabilities for today's integrated manufacturing environment.

A novel means for storing digital information using a conventional composite video tape recorder is presented in Abstract 4. Although quite simple in principle, it is very general in applicability. The normal chrominance (color) component of a composite video signal is replaced with a phase-modulated high-frequency carrier. Nearly any type of information could be modulated onto this carrier and stored on tape. Moreover, monochrome video information could



Patent Number	4,908,769
Issue Date	1990 03 13
Inventor(s)	Vaughan, John E.; Patry, Bernard
State/Country	GBX
Assignee	Schlumberger Electronics (UK) Limited
US References	4,019,135 4,162,530 4,240,030 4,351,028 4,355,361 4,465,970 4,731,575
US Class	364/464.04 324/142 340/825.35 364/483
Title	Commodity metering systems
Abstract	A multirate commodity metering system comprises an electronic key containing an EEPROM, and a meter for metering the commodity and having a receptacle for receiving the key. The meter also contains a real-time clock. The key can operate as a prepayment device, a programming device, and a meter-reading device. Thus the key is taken to the commodity supplier's premises, where, in return for a payment, a credit signal representative of the payment is stored in the key. At the same time, a new program for the meter, that is, new times for switching between rates and new prices per unit of commodity for each rate, together with the date and time at which the new program is to come into operation, can also be stored in the key. Then, when the key is inserted in the receptacle in the meter, the payment signal and the new program are entered into the meter. Additionally, the current set of meter readings (i.e., the respective readings for each rate) are read into the key, along with the date and time. The next time the key is presented at the supplier's premises for the entry of a new credit signal, the meter readings, and the date and time they were entered in the key, are read from the key.



Patent Number	5,038,318
Issue Date	1991 08 06
Inventor(s)	Roseman, Brooks T.
State/Country	NC
Assignee	Square D Company
US References	4,038,533 4,149,235 4,180,860 4,215,398 4,228,495 4,363,090 4,517,637 4,648,062 4,751,635 4,791,556
US Class	364/900 364/926.9 364/949 364/927.99 364/921
Title	Device for communicating real-time data between a programmable logic controller and a program operating in a central controller
Abstract	Add-in program instructs operator through a general-purpose spreadsheet program in a personal computer to move real-time status and control messages directly between cells in the displayed spreadsheet and addressed registers of programmable logic controllers (PLCs) is disclosed. The PLCs operate such as machine tools for processing stations and connect together and to an interface card in the personal computer over a network. The invention facilitates a user's real-time monitoring and control of the manufacturing performed at the machine tools or processing stations through mathematical and logical features of the spreadsheet, which are well known and easy for a user to implement.

PATENT TALK

be stored as well on the same tape. While mention is made of permitting the VCR to be used as a mass storage device, one could also envision this technique being used to store time-stamp data or other timely information related to the recorded video information. Applications in security and manufacturing rapidly suggest themselves.

Again in the computer video area, Abstract 5 from Personal Computer Cameras Inc. makes your PC into an electronic still-video camera. Their novelty in this patent seems to stem from compression and decompression of the video image and its compatibility with standard PCs, specifically IBMs (or compatibles) and the Apple Macintosh.

4

Patent Number	5,01 9,914
Issue Date	1991 05 28
Inventor(s)	Dropsy, Patrick J.
State/Country	FRX
US References	31,863 4,138,694 4,380,047 4,530,048 4,575,773 4,752,833 4,812,920 4,819,059
US Class	358/310 358/335 358/12 360/32 360133.1
Title	System for recording and/or transmitting binary information at a high data rate via known means for recording and/or transmitting video information, and for decoding the digital data
Abstract	A method and an apparatus encode and decode binary signals representative of digital information onto or from a high-frequency carrier wave. The carrier wave replaces the chrominance signal of a composite video signal. Encoding and decoding of the binary signals are by phase modulation and demodulation of the carrier wave. The method and apparatus allow simultaneous transmission of both the binary signals and the monochrome portion of the composite video signal. The apparatus can be configured to serve as an interface between a minicomputer and a composite video recorder, so that the composite video recorder can be used as a mass storage peripheral.

5

Patent Number	5,138,459
Issue Date	19920811
Inventor(s)	Roberts, Marc K.; Chikosky, Matthew A.; Speasl, Jerry A.
State/Country	VA
Assignee	Personal Computer Cameras, Inc
US References	4,074,324 4,131,919 4,302,776 4,456,931 4,571,638 4,614,977 4,758,883 4,803,554 4,829,383 4,837,628 4,847,677 4,903,132 4,905,092 4,963,986 4,972,266
US Class	3581209 358/140 358/93 3581903
Title	Electronic still video camera with direct personal computer (PC) compatible digital format output
Abstract	An electronic still camera comprising a lens, shutter, and exposure control system, a focus and range control circuit, a solid state imaging device incorporating a Charge Couple Device (CCD) through which an image is focused, a digital control unit through which timing and control of an image for electronic processing is accomplished, an Analog-to-Digital (A/D) converter circuit to convert the analog picture signals into their digital equivalents, a pixel buffer for collecting a complete row of an image's digital equivalent, a frame buffer for collecting all rows of an image's digital equivalent, and a selectively adjustable digital image compression and decompression algorithm that compresses the size of a digital image and selectively formats the compressed digital image to a compatible format for either the IBM Personal Computer and related architectures or the Apple Macintosh PC architecture as selected by the operator so that the digital image can be directly read into most word processing, desktop publishing, and database software packages including means for executing the appropriate selected decompression algorithm; and a memory input/output interface that provides both temporary storage of the digital image and controls the transmission and interface with a standard Personal Computer (PC) memory storage device such as a digital diskette. The digital diskette is removable inserted into the housing of the camera prior top use in recording digital image data.

PATENT TALK

Regardless of where this particular patent fits, I do believe it is only a matter of time before digital still-video takes its place in our everyday life.

My professional involvement with touch-input industrial display systems leads me to closely monitor new developments in this area. Abstract 6 presents a novel way for a user to interact with a video display by using his finger as a pointing device. This Digital Equipment Corp. patent apparently involves electronically capturing an image of the user's finger and the background field to which he is pointing. It does not seem to be limited to pointing to a computer display screen, and in fact might work as well or better if he were pointing to a large control console. The device uses some fancy image processing and pattern matching techniques for extracting various responses from the image. From all this, the location of "blobs," object edges, lines, and terminated line segments are identified. Finally, the position of a pointing object such as a finger is recognized from all this and is used to "direct an application program in a most natural way without the distraction of manipulating a data input device."

Finally, a method of correlating recorded video and textual information is presented in Abstract 7. This "video transcript retriever" would permit text to be searched for in a computer transcript. It then links this text to the position on the video recording where these words were recorded, and then accesses that spot on the videotape. There are many diverse applications for such a device. Certainly, educational and courtroom applications come quickly to mind. But if one had an adequate speech-to-text conversion mechanism available also, then any type of videotape recording could be processed to produce a computer-searchable document that, in turn, could be used to index, access and retrieve the corresponding video information. ❑

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

SOURCE

Patent abstracts appearing in this column are from the Automated Patent Searching (APS) database from:

MicroPatent
25 Science Park
New Haven, CT 065 11
(203) 786-5500 or (800) *648-6787*

I R S

422 Very Useful 423 Moderately Useful 424 Not Useful

6

Patent Number	5,168,531
Issue Date	1992 12 01
Inventor(s)	Sigel, Claude
State/Country	c o
Assignee	Digital Equipment Corporation
US References	3,701,095 4,468,694 4,783,833 4,884,225 4,905,296 5,014,327 5,059,959
US Class	382148 382142 382116 340/709
Title	Real-time recognition of pointing information from video
Abstract	An occurrence of a predefined object in an image is recognized by receiving image data, convolving the image data with a set of predefined functions to analyze the image data into occurrences of predefined elementary features, and examining the occurrences for an occurrence of a predefined combination of the elementary features that is characteristic of the predefined object. Preferably the image data are convolved directly with a first predefined function to determine blob responses, and a second predefined function to determine ganglia responses indicating edges of objects. Then the ganglia responses are convolved with a third predefined function to determine simple responses indicating lines in the image, and the simple responses are combined with the ganglia responses to determine complex responses indicating terminated line segments in the image. A pointing finger, for example, is recognized from the combination of a blob response and a complex response. The method, for example, permits a data input terminal to recognize in real time the presence, position, and orientation of a pointing finger, to eliminate the need for data input devices such as "mice" or "joysticks." Therefore a user can direct an application program in the most natural way, without the distraction of manipulating a data input device.

PATENT TALK

Patent Number	5172,281
Issue Date	1992 12 15
Inventor(s)	Ardis, Patrick M.; Markovich, Marko R.; Thompson, Kevin W.
State/Country	TN
US References	4,641,203 4,924,387 4,941 ,125
US Class	360172.2 360/33.1 3641409 369/14
Title	Video transcript retriever
Abstract	A video transcript retriever includes a control unit. a control interface, a tape unit, and a display unit. The control unit includes a control computer having a software package consisting of control software, text software, and edit software. The control software has the capacity to permit simultaneous operation of both the test software, which is capable of storing and searching voluminous documents, and the edit software, which has the capacity to operate the tape unit with precision. The text software is capable of performing a search function that at any time can provide the exact location of a specific passage within the searched document in terms of page and line. The edit software has the capacity to provide at any time the timecode number prerecorded on the videotape that corresponds to a specific passage. The process for locating and retrieving specific information on a videotape includes the steps of striping the videotape by assigning a numerical address for every one-thirtieth (1/30) of a second segment of the videotape; indexing the words written in a computer transcript to the words spoken on the videotape by assigning a timecode number to both the computer transcript and the videotape segment where each question/answer passage begins; and instructing the tape unit to shuttle to a precise tape location determined by the timecode numerical address located during the search of the computer transcript.

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 805 1/803 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 SXA51 cross-assembler & hex file manip.util.*
- *Extensive documentation*
- *Tutorial included*
- *Runs on IBM-PC/XT or compatibile*
- *Compatible with all 8051 variants*
- ***BXC51$ 295.***

508-369-9556
FAX 508-369-9549

Binary Technology, Inc.
P.O. Box 541 • Carlisle, MA 01741

#143

At last!
Real time industrial control for your P.C. under Windows ™ or DOS ™

The I/O Bits Machine Programming System

I/O Bits is the innovative software using icon base programming that turns any IBM PC compatible into powerful real time controller for mechanisms an machines.

- Intuitive "flow diagram" style programming using easy-to-understand icons
- Program your machine without ladder diagrams or coding.
- Test machine operation in minutes or hours
- Full function industrial control elements
- On-line help with examples and tutorials
- Use the printer port and get 17 I/O lines for free.

I/O Bits for the parallel port.	$99.
I/O Bits for Mechanisms	$499.

Position Sensitive Robots, Haverhill, MA
508-521-9580 voice, 508-521-9584 fax.

#134

CONNECTIME conducted by Ken Davidson

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

This month in ConnecTime, we stat-t off with a discussion of grounding issues and noise. How many times have you attributed a problem to noise?

Next, we talk about the best way to make a homemade accelerometer. It turns out to require more mechanical than electrical skills.

Third is a thread about winding matching transformers for an RF power amplifier. We don't often stray into high-frequency analog design, so there are some interesting tidbits in this one.

Finally, there are many methods for defecting the zero crossing of an AC signal. Which is the right way for you?

To ground or not to ground

Msg#:12665
From: RONALD HORNER To: BOARD DESIGNERS

I have Z80 controller board that I built and I am not sure if I should ground the chassis of the enclosure to the same ground as on the 5-V supply or the DC 5-V power supply? I have been told that all grounds should be common, but I wonder if some isolation of the 5-V supply should be considered to prevent noise from entering the system?

Msg#:12749
From: ED NISLEY To: RONALD HORNER

I think the answer is "it depends"-how's that for advice?

From a safety standpoint you want the enclosure connected to earth ground so it cannot become "hot" if it's shorted to a power supply. If any of your connectors use the enclosure as the common side, you want it connected to the logic ground to minimize the noise injected into the inputs. ESD and EM1 considerations tell you to keep all signal currents off of the enclosure, but connect it to the power supply common so it shields the circuitry.

You pay your money and you take your options.. .but leaving the enclosure floating is a bad idea on all sides!

Msg#:13049
From: RONALD HORNER To: ED NISLEY

Yes, I always ground the enclosure to earth ground or the same ground on the neutral/ground of a 120-V side, but the low voltage? Shouldn't the 5-V ground not be part of the enclosure ground?

Msg#:12811
From: JAMES MEYER To: RONALD HORNER

If you are going to make connections to the outside world, the best way is with optically isolated I/O modules. If you do that, then there is no necessity to ground the supply to any part of the chassis.

Msg#:13050
From: RONALD HORNER To: JAMES MEYER

Right, that's what I believe in for absolute integrity!

Msg#:13171
From: ED NISLEY To: RONALD HORNER

To ground or not, that is the question!

I know I've had trouble with widgets on my desk that came about because they were not referred to power line ground. After I added a few clip leads here and there, the mysteries Went Away.

Betcha Pellervo has some cogent observations on this topic dating back to when he did something involving kiloamperes and megavolts! Come in, Pellervo.. .

Msg#:13186
From: RONALD HORNER To: ED NISLEY

You know, I don't believe that ground is really ground. I mean, in my experience "ground" is only an antenna to draw in more noise. I like isolation. Absolute isolation using fiber optics, optical encoders, isolation transformers, and the like. Everybody has their theories on the stuff, but for me it's always trouble.

Msg#:13425
From: ED NISLEY To: RONALD HORNER

Yeah, Steve's opinion on isolation is that a laser diode across the driveway beats a half-inch copper ground strap any day...

Msg#:13691
From: PELLERVO KASKINEN To: RONALD HORNER

The unfortunate side effect of the term "grounding" is that several people expect a connection to a ground rod to

solve all their interference problems. That simply is not possible. Maybe a term such as "return path" would be more appropriate. Even then, you have to realize there are different mechanisms of noise coupling. All of that, of course, is covered by a gentleman named Maxwell (which coverage, by the way, was the basis for Einstein and his grand achievement).

Frankly, any and all signals are referenced to SOMETHING. If you use an optocoupler, your reference is closer to the place where you need your signal, but the reference is still physically there. Now, generally the difficulties people have with "grounding" is because their signals run through too long, or mixed, paths or there are more signals than the one of importance to you in the common path (the "ground"). If you can make a truly zero impedance (not only resistance] for your "ground," there would be no interference. In real life, you cannot defeat inductance in any physical length of wire, whatever thickness, so you are bound to get voltage drops along the length of it. If you happen to have two of the wires, to establish an independent reference, you start seeing the signals and interference.

The different types of noise coupling that I have been dealing with are direct voltage drop in common runs, capacitive coupling to nearby high-impedance circuits, and inductive coupling between parallel wires or wire loops. You might add a radiated signal (i.e., your "antenna"], but that is just a generalized case of the inductive and capacitive coupling mechanisms. And the remedies are slightly different depending on the MAIN mechanism of coupling.

Electrostatic (capacitive) coupling is combated with increased distance, either physically or by shielding, and/or changed impedance levels at the receiving side.

Electromagnetic coupling is reduced by reducing the loop size. The traditional use of twisted pairs is the prime example. It is effective for both the transmitting and for the receiving side. Your use of optocouplers works also partly on this principle, reducing the loop size. Another way is to reorient some of the wire runs. Make the signal line run across the noisy line rather than along with it.

As far as the common grounding rules go, they are not necessarily optimum for some signal situations, but they are necessary for the safety. There is another loop or return path that is mandatory. In fact, your desire to separate your signals ("not grounding") is one way of reducing the coupling through a shared path. The power line grounding efficiency varies greatly and the resistance is always large enough to cause problems to the unwary.

Basically, use common sense for selecting the points of signal common and do some experimentation, like Ed suggested, with clip leads and you may find a point that pretty much counterbalances the stray signals your power supply and other necessities cause. Once you have found a good point, you may be able to use that until you make some (almost any) changes to your circuit.

Msg#:14009
From: RONALD HORNER To: PELLERVO KASKINEN

Thanks for the info. What you said pretty much sums up what I know and suspected. I'll be doing more research on this. The main problem I have had is one of credibility! Noise appears to be both a hidden problem for machines that don't work, but it is also used as an excuse for bad programming or a bad design. So, I am starting to gather some written material from professionals on their studies and experiences to use as a wedge between the truth and the fairy tales.

Msg#:14041
From: RUSS REISS To: RONALD HORNER

Wow! You struck a sensitive nerve with that remark! While it's commonplace for programmers and engineers to point fingers, one at the hardware and the other at the program, it seems they BOTH love to blame "noise" for any problem that cannot be identified within a short time! I have seen precious few cases of true noise problems, and those that have appeared were usually the result of poor engineering practices by neophyte designers. Your compendium could be a real service to newcomers on how to do it right!

As a postscript, while sometimes necessary to get a product out the door, I don't see "software filtering" as a cure-all solution to inadequate hardware design, either.

Accelerometers

Msg#:13013
From: STEPHEN PHILLIPS To: ALL USERS

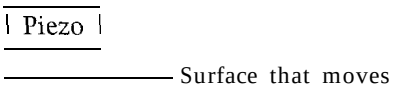
I would like to use a strain gauge as an accelerometer. The sensitivity would have to be high (for my ignorance of these devices, I could be wrong). Since f = *ma* and the mass I intend to use is 1 gram, this would mean a strain gauge at least capable of sensing <<1/1000th of a Newton. I do not expect a force grater than 1 N ever being placed on the strain gauges (however, you never know].

I would like to use these to detect sudden (i.e., erratic) changes in the stability of a model airplane. I intend to process these changes by comparing them with some preset tolerance. A computer will decide what action is necessary. Since the moment of inertia is large enough, the computer would not need to be fast since a resolution of <1/60th of a second goes beyond the timing of pulses to standard hobbyist servos.

CONNECTIME

Msg#:13051
From: TOM MAIER To: STEPHEN PHILLIPS

A home-brewed sensor could be made from a piezo crystal taken from a buzzer:



This could be used as an accelerometer for a single axis. You might have to electrically filter out some of the resonant vibrations that will be coming out of it also.

Increasing and decreasing the mass will change its sensitivity.

Msg#:13336
From: STEPHEN PHILLIPS To: TOM MAIER

Hadn't thought of cheap piezo elements. I need to measure fairly small changes. I was planning on converting the output into a PWM signal for a small computer to process. The output of the sensor goes to a sample-and-hold circuit. Then that reference is converted to a PWM signal that is read by a computer, which of COURSE converts it into some value that has meaning to the computer.

Any way of projecting the sensitivity? Trial and error, I suppose. What I want to do is use a small computer to stabilize a model plane that is inherently unstable.

Msg#:13687
From: PELLERVO KASKINEN To: STEPHEN PHILLIPS

Using strain gauges for an accelerometer is mainly an exercise in handiwork. You need to build either a membrane or a reed that is anchored at one end (rim, in case of the membrane) and has your mass attached to the other end or center. There has to be enough room for the strain gauge elements to be glued to opposite sides of the flexing member. Thereby they become elongated/compressed with any deflection of the support member and a half-bridge or a full-bridge circuit can be used to measure the relative difference.

The strain gauges that I have used were capable of producing up to 10% change in resistance before failure, provided the glue was equally good. You could count on reaching over **1** %, but don't need that much with good amplifiers, even with DC coupling. I recommend an OP-27. When I did use the strain gauges more often, I had a commercial AC excitation bridge instrument. But I have done or seen some pretty good DC-coupled units as well. One of them even used a frequency-modulation principle to send the strain indication from a moving shaft via radio.

The key issue is selection of your materials. Too thin a membrane appears to give a high sensitivity, but loses some of it because the strain gauges on the opposite sides are not affected as much as they would be if attached to a thicker membrane with the same flexing. Also note that if you anchor two ends of a reed solidly, then the center moves very little under a transversal force. If you can, use only one anchored end. Otherwise, you better make some pivoting help (i.e., convolutions). The strain gauges should be mounted as close to the maximum BENDING position as possible. In the single-end reed case, it is as close to the fixed mounting as possible. Run the connection leads away from the mounting.

So, pick up some shim stock, probably about 0.004" to 0.006" stainless steel would be a good starting point. Cut it to the width of the strain gauges and anchor one end of a longer-than-your-final-target piece. Attach your mass to different positions and start shaking the construct. Try to evaluate the amount of deflection resulting. Cut the strip shorter. Is it still flexing OK? If yes, cut it more until you have reached the smallest practical size that seems to do the job. Then glue the strain gauges in place with good epoxy (probably requires the regular 24-hour cure variety).

For the final implementation, you probably don't need to worry about excessive deflection, but if you do, just put mechanical stops on each side of the free end at suitable distances.

I hope this is not too cryptic without a picture. Just don't have time to do any ASCII graphics.

Msg#:13752
From: JAMES MEYER To: PELLERVO KASKINEN

The only thing I would add to your excellent tutorial is a word about resonance. Homemade accelerometers often suffer from insufficient damping. Putting a strain gauge on a piece of springy shim stock is asking for a big resonant response at some frequency. Murphy says that that response *will* be in the center of the range of signals that you're trying to measure. 8-)

The cures are varied. Most often involving some sort of added semiviscous material. Testing and curing those resonances is partly responsible for the cost of commercial accelerometers.

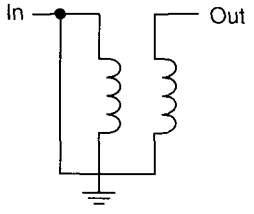
RF transformers

Msg#:13318
From: TERRY NORRIS To: ALL USERS

As part of my continuing training as "Jack of All Trades, Master of None," I have recently been given the simple task of building an RF power amplifier. Luckily, I

found a schematic for exactly what I need. Alas I don't know how to build the matching transformer.

Book description:



T1: 20 turns 30-ohm coax cable, #30 bifilar on micrometals T-50-6 toroid.

T2: 1 turn of two 50-ohm coax cables in parallel through two balun cores stackpole #57-9130. μ_O=125.

Msg#:13451
From: JAMES MEYER To: TERRY NORRIS

First, get all the parts you'll need. Check out some ham radio magazine advertisements.

T1 uses ordinary enamel insulated "magnet" wire. Bifilar means to take two (bi) wires (filar) long enough to make 20 turns and twist them together with about one or two twists per inch. Take the twisted wires and wind them 20 times through the toroid. Space the turns out to fill the toroid. If you use wires of different colors, it will make identifying the windings easier. The start and end of each winding is important when the connections are made. My guess is the starts are on one side of your drawing and the ends are on the other.

T2 is a bit more complicated. In fact, I can't help you there without a little more info. Try another message with a drawing for T2.

Msg#:13679
From: TERRY NORRIS To: JAMES MEYER

Alas, the schematic symbol for T2 is the same as T1. Also, I was thinking about the word bifilar wound. Isn't that the winding technique on noninductive wirewound resistors?

Msg#: 13748
From: JAMES MEYER To: TERRY NORRIS

No. It isn't. "Bifilar" simply means "two wires." Usually side by side, and wound at the same time.

If T2 is shown on the schematic exactly like T1, then the author must not have been using the coax as anything other than large-diameter wire. By that I mean the shield and center conductor at each end of the cable must have been connected together.

In that case, just use the description of T1 and change the number of turns.

Msg#:13940
From: RUSS REISS To: TERRY NORRIS

You seem to be having problems with the meaning/construction of these "transmission line transformers," particularly T2, which uses coax as the transmission line.

First off, there are a few good books on the subject, if you should like to become an "expert." The "bible" is Jerry Sevick's "Transmission Line Transformers" published by ARRL. Then there are the many excellent/practical/readable books by Doug DeMaw which address this and solid-state RF amplifiers in great PRACTICAL detail. I suggest his "Practical RF Design Manual" published by Prentice-Hall.

But let's take a look at your transformer. If you, for a moment, reverse the "input" and "output" and feed a signal into the OUT/GND terminals, you can see that it is impressed across the lower inductor. Due to the phasing of the two windings and the transformer action, a similar voltage should be coupled to the upper coil in such phase that it ADDS to the applied signal producing twice as much voltage at what you have shown as the "input" terminal (with respect to ground). Due to conservation of power (if there were no losses), twice the voltage must mean there is half the current. So the impedance is 2V/0.5I or four times what it looks like on the right-hand side. Thus (using your I/O specification], we can consider this a 1:4 impedance ratio matching transformer (or 4: I going the other way from out to in). You can use it in either direction, it doesn't matter...just depends on your application and if you are stepping UP or DOWN the impedances.

Now, what about the transmission line? Doesn't matter much if it is bifilar (parallel wire) line or coax. The bifilar approach does not REQUIRE that the wire be twisted. That makes it more convenient to work with and somewhat lowers the characteristic impedance of the transmission line it represents. Coax comes in only some characteristic impedances, and 50 ohms is very common. Since, for T2, they wanted 25-ohm coax, they simply paralleled two pieces of coax, connecting inside to inside and outside to outside at each end, and then consider it a SINGLE piece of 25-ohm coax. You could also buy and use 25-ohm coax such as Microdot #260-4118-000. If you choose to use paralleled 50-ohm cable, I suggest RG- 174-U for low-power applications. Just be VERY careful when soldering to it (that you don't melt the inner insulation and short inside to outside], and consider a dab of RTV at the ends to both seal it against moisture and to strain-relieve the end where it is often subject to stress, If you use bifilar wire, you can expect rough characteristic impedances of around 30 ohms for #30 wire and 60 ohms for #32 wire.

The design of these transformers is not critical unless you are looking for optimum power transfer, very wide band

operation, and so forth. Typically, the characteristic impedance of the cable used is equal to the square root of the in/out impedances. My guess is your circuit is matching 12.5 ohms to 50 ohms, which would require the use of 25-ohm coax {sqrt[12.5 * 50]}. But these have been built with widely different cable and still work well. What you want is a line length that is short compared to a wavelength at the highest frequency it will pass, and yet sufficient inductance (due to number of turns and the core permeability) to block circulating currents at the lowest frequency of interest. But again, I say it's NOT critical!

Hope that gives you something to get started with. There are a wide variety of these transformers-in both unbalanced or balanced form at both input and output. The widely mentioned BALUN transformer is a BALanced to UNbalanced form of these, and may have 1: 1 or other transformation ratio, as required.

Msg#:13972
From: RUSS REISS To: TERRY NORRIS

In my other message I forgot to answer what is perhaps your biggest question with respect to using coax in a transmission line transformer. How do you connect the coax!? Well, if you followed my discussion there, you see that it doesn't really matter what the transmission line is—parallel wires or coax. In either case, there are TWO wires and they are wound "bifillarly" through the core. In the case of coax, one wire is the outer conductor, while the other is the inner conductor. So, to get the phasing correct, you will need to connect the inner conductor at one end of the coax to the outer conductor at the other end AFTER winding it through the core.

Zero-crossing detectors

Msg#:11433
From: GREG PRICE To: ALL USERS

Any ideas out there on a simple circuit to detect the zero-crossing point of the AC signal so I can switch a nonlatching relay on and keep it on until a control signal (8255 buffered) goes low. Thought about an AND gate with a flip-flop or a PAL. I am sure this has been done many times and many ways. Any help would be appreciated.

Msg#:11877
From: JOHN CONDE To: GREG PRICE

Well, the easiest way I know of to detect the zero cross would be with a zero-crossing detector chip (3059 or 3079, if memory serves). This will output a pulse when the signal crosses zero (in either direction). The pulse can gate an SCR which will keep your relay on. Turning it off is another problem; the simplest would be to have the control signal supply the current to the relay (through the SCR), then when the signal went low, the SCR would turn off and the relay would drop out. Of course, if the control signal can't supply enough current, you can have it control a transistor that will. Hope this helps.

Msg#:11889
From: PELLERVO KASKINEN To: GREG PRICE

First, get a zero-crossing signal as a narrow pulse. Then feed it into the clock input of a D-latch or make one out of a pair of 3-input NAND gates. One input on each for data, a second one for the cross connection, and the third one tied together for the clock.

I used an optocoupler to provide the basic zero detection, or actually the signal polarity detection. I fed the output to an XOR directly to one input and through a 0.1-ms RC time constant filter to the other input. Got 0.1 -ms narrow pulses on every zero crossing.

Msg#:11915
From: LARRY G NELSON SR To: GREG PRICE

How about a Motorola MOC3011 or similar. These are optoisolators with zero-crossing detect and triac output. Not sure the exact part number you would want, but this could be the ticket for what you are looking for.

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.

ARTICLE SOFTWARE

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

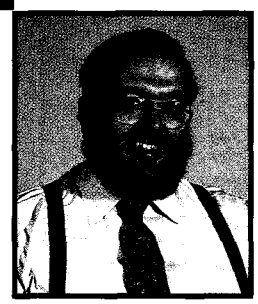
To order Software on Disk, send check or money order to: The Computer Applications Journal, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your VISA or Mastercard and call (203) 8752199. Be sure to specify the issue number of each disk you order. Please add $3 for shipping outside the U.S.

I R S

425 Very Useful 426 Moderately Useful 427 Not Useful

STEVE'S OWN INK

PC Clone ~~$5,000~~ SALE $1.98



While I don't claim to be any sort of an economist, I still can't help but think of world events in terms of mathematics. I guess my math professors would be proud of me for that. One "formula" I have been attempting to understand lately is the following set of relationships and what they mean:

Consumer willingness to purchase and their satisfaction is inversely proportional to computer pricing. Computer pricing trends are inversely proportional to profit margins. Consumer satisfaction is proportional to profit margins. The derivative of reduced computer pricing is increased consumer performance expectations. Increased consumer expectations has a direct linear relationship to a vendors need to invest heavily in R&D, which is inversely related to quarterly profits. Companies driven by margins must raise prices or cut costs.

Based on the first relationship, cost cutting is the only alternative for many. This means staff reductions or using lower-cost materials. Lower-cost materials means lower quality (in some cases), which is inversely proportional to consumer satisfaction. Lower-cost materials is inversely proportional to company profits? Lowered staff counts leads to lower morale. Lower morale is inversely proportional to productivity. Lowered productivity is inversely proportional to company profits. What a no win situation!

As you can see, this is a complex set of relationships with many interrelated factors. Although I still haven't found any unifying factors that lead to a steady state or predictable response curve for the equations, I doubt anyone else has yet, either. Witness the recent turmoil in the appliance (er, commodity) market for home computers. Tumbling prices, fallen kings, paradigms shifting as rapidly as the Sahara sands, and former industry powerhouses crippled in the channel and bailing water. This state of affairs has got to have the marketing and accounting departments quaking in their shoes.

I remember attending one seminar where it was said that if the car industry had advanced as rapidly as the computing industry, you would be able to buy a car that was the size of a matchbox, went 500 miles an hour, and cost about a dollar. Car prices are going up and things haven't changed much. Computer prices are avalanching downward and performance is skyrocketing.

Dog eat dog is one thing, but an industry eating itself alive is quite another. Many pundits claim that the consumer is benefiting from this chaos with more MIPS and computing power on their desk than most nations were able to afford as few as twenty years ago.

But I wonder, if many companies fall by the wayside, or simply give up, from this unbridled shake out...Will the truly gifted developers look for more peaceful pastures to explore, thus slowing the pace of innovation? Will this reduction of players in the marketplace lead to a kind of Orwellian nightmare of one company providing the computing platforms. Imagine if the Post Office were the ones in charge of computing development....price increases without a lot of performance gains. Not a pretty sight.

So are all of these signs indicative of a vibrant, exciting market? Or is this chaos a portent of things to come? Well, a lot of these companies started out in garages. Maybe some of them will be back there again soon.

