CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

December 1994 — Issue #53

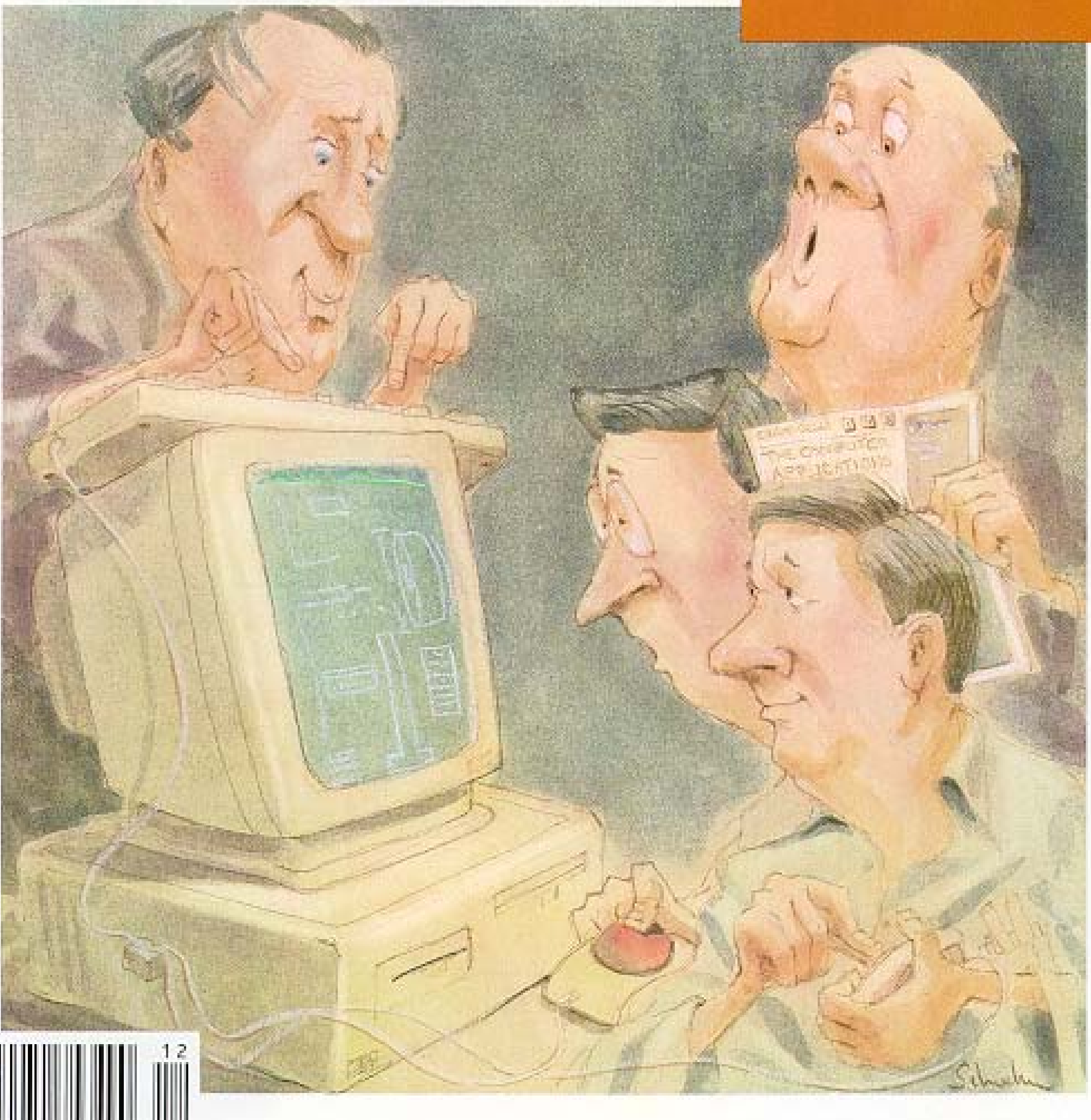
COMPUTING IN REAL TIME

Real-time Programming Techniques

Developing ACCESS.bus Software

Do On-the-fly Signal Processing with your Sound Blaster

Automated Plant-watering System



$3.95 U.S.
$4.95 Canada

EDITOR'S INK

Putting the Cart Before the Horse

before I talk about what's in store in this issue, I thought I'd get ahead of myself and talk about what's coming in the new year. If you haven't read Steve's editorial yet (you mean you don't read the last page first?), be sure to flip back there when you're done here. In the meantime, let me be the first to warn you that we kick off our seventh year of publication with a completely new cover. The insides won't change one bit, either graphically or editorially. We'll simply be easier to find on the newsstand (or in the mailbox).

To make storage of back issues easier, we're also going perfect bound (flat edge) as opposed to saddle stitched (folded and stapled) as we've been since the first issue. Finding a specific issue will be as easy as scanning the spines on your bookshelf.

But, back to the issue at hand, we're focusing on real-time programming. We start with a primer article that draws a comparison between real-time programming and state machines. By representing the task with a relatively simple model, writing the code often becomes more manageable.

Next, we take another look at ACCESS.bus. Not only can it be used to tie together desktop peripherals for a single user, but it can be the basis to connect multiple users to a single computer for real-time applications ranging from interactive educational software to competitive gaming.

When it comes to real-time signal processing, you may not have to buy a new, dedicated DSP board to get the performance you're looking for. The Sound Blaster board already in your PC probably contains a DSP that can be completely reprogrammed to do more than just enhance your multimedia experience. Check out just what's involved.

In our final feature, we follow up Octobers introduction to the ARM processor with a more in-depth look at its hardware and development boards.

Be sure not to miss this years Circuit Cellar Design Contest winner spread. We had our toughest time yet trying to pick winners from a field of highly creative entries. We'll be featuring many of the projects in feature articles in the coming year, so watch for them.

In our columns, Ed continues through the protected land by adding some LCD display code to track execution, Jeff gets his green thumb wet by creating an automated plant-watering system, Tom tries to find his way through Silicon Valley using the latest in compass technology, and John lays the groundwork for working with bar codes.

Ken

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Janice Marinelli

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITOR
John Dybowski

NEW PRODUCTS EDITOR
Hat-v Weiner

ART DIRECTOR
Lisa Ferry

GRAPHIC ARTIST
Joseph Quinlan

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Sue Hodge

CIRCULATION COORDINATOR
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 675-2751 Second class postage paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U S A. and possessions $21 95, Canada/Mexico $31 95, all other countries $49.95 All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S bank. Direct subscription orders and subscription related questions to The Computer Applications Journal Subscriptions, P 0. Box 696. Holmes, PA 19043-9613 or call (800) 269-6301

POSTMASTER: Please send address changes to The Computer Applications Journal. Circulation Dept P.O Box 698, Holmes, PA 19043.9613.

Cover Illustration by Bob Schuchman
PRINTED IN THE UNITED STATES

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

NORTHEAST & MID-ATLANTIC	SOUTHEAST	WEST COAST
Barbara Best	Christa Collins	Barbara Jones & Shelley Rainey
(908) 741-7744	(305) 966-3939	(714) 540-3554
Fax: (908) 741-6823	Fax: (305) 985-8457	Fax: (714) 540-7103
	MIDWEST	
	Nanette Traetow	
	(708) 789-3080	
	Fax: (708) 789-3082	

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

12 **Designing Real-time Embedded Software Using State-machine Concepts**
by David Tweed

20 **All Together** Now/Writing Multiuser Application Software for ACCESS.bus
by David Rodgers & Peretz Tzarnotzky

30 **Make the Most of Your DSP-based Sound Card**
by Bob Fine

40 **Circuit Cellar Design Contest Winners**
compiled by Janice Marinelli

44 **A RISC Designer's New Right** ARM/Development Boards and Design Specifics
by Art Sobel

56 ❑ **Firmware Furnace**
Journey to the Protected Land: Fancy Text Output and a Boot Mystery
Ed Nisley

66 ❑ **From the Bench**
Engineer Seeks Personal Gardener/Assisting Vocally Challenged Vegetation
Jeff Bachiochi

72 ❑ **Silicon Update**
Do You Know the Way to San Jose?/Precise Navigation Technology
Tom Cantrell

78 **Embedded Techniques**
Between the Lines/Bar Code and Decoding Bar-code Algorithms
John Dybowski

INSIDE ISSUE 53

2 Editor's INK
Ken Davidson
Putting the Cart Before the Horse

7 New Product News
edited by Harv Weiner

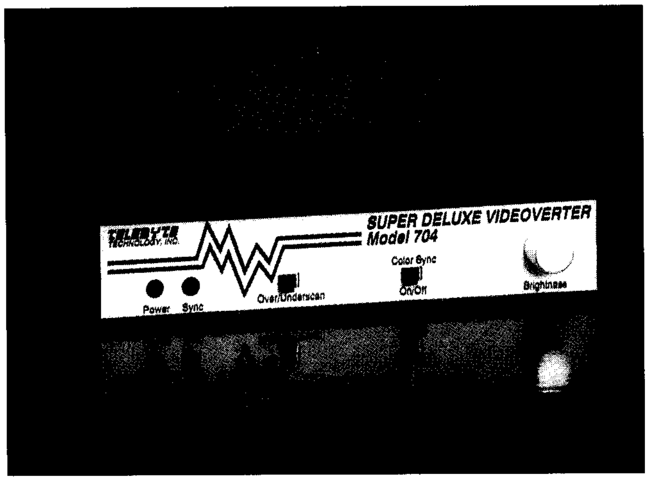
ConnecTime 85
Excerpts from the Circuit Cellar BBS
conducted by Ken Davidson

Steve's Own INK 96
Steve Ciarcia
What's in a Name?

Advertiser's Index 81

NEW PRODUCT NEWS

Edited by Harv Weiner



VGA-TO-VIDEO CONVERTER

Telebyte Technology has announced a device for converting VGA images to television's NTSC and PAL formats. The **Model 704 Super Deluxe Videoverter** enhances presentations, training sessions, and the creation of video tapes.

The Model 704 offers a complete list of features for a VGA-to-NTSC converter at an economical price. It is equipped with three outputs: S-VHS, AV, and RF. Any video device—from a standard television with RF input to the most sophisticated television monitors, projectors, or VCRs with S-VHS inputs-may be connected. Since the outputs are available simultaneously, the Model 704 can drive three different output devices.

The Model 704 provides flicker-free operation for displaying a VGA image on a TV. The VGA-to-video conversion features a three-line, video-RAM memory. This memory gives independent retiming of the NTSC signal which provides simultaneous display on the VGA monitor and television. The Model 704 operates under the control of a DOS TSR software driver. This driver assigns a group of hot keys to adjust the TV display for position and overscan.

The Model 704 supports Windows 3.0 and 3.1 operation and VGA modes up to 640 x 480 with 64K colors. The Videoverter supports all VGA modes in which the horizontal frequency is fixed at 3 1.5 kHz.

The Model 704 is 7.5" x 5.5" x 1.5" and sells for $545.

Telebyte Technology, Inc.
270 Pulaski Rd.
Greenlawn, NY 11740
(516) 423-3232
Fax: (516) 385-8184

#500

SERIALLY CONTROLLABLE AC POWER CENTER

International Micro Electronics Group (IMEG) has just released their flagship power-control device, **Pow-R-Bar.** Pow-R-Bar is an intelligent, configurable, six-outlet AC power center, which connects to a computer using RS-232 to individually control each of the unit's six 1 10-V AC outlets.

With Pow-R-Bar connected to a computer, each outlet on the back of the unit can be controlled through a computer's serial port. The user types in simple commands using the included demonstration software or through custom programming (all necessary codes are provided in the owner's manual). All functions are also available through any standard, serial-communications package. Virtually any computer can control Pow-R-Bar.

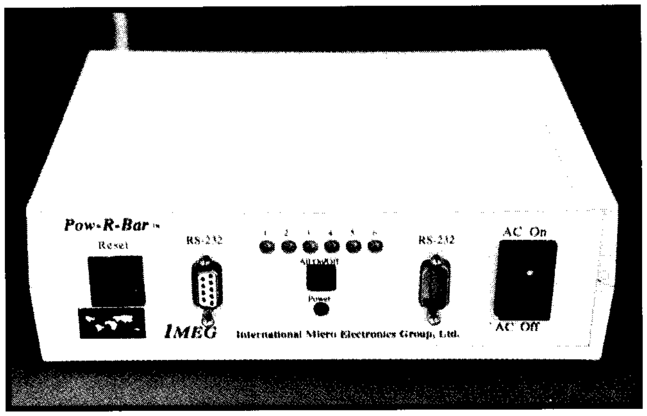
With each Pow-R-Bar, IMEG supplies a Microsoft Windows-based, graphical-interface demonstration program and a DOS-based demonstration program, which operate l-26 Pow-R-Bars daisy chained together. Using the included software, the bars can be labeled using the letters A-Z and each outlet can be individually named (names can be any character length).

With 26 Pow-R-Bars linked together, up to 156 outlets, each individually controllable, are joined. The distance from the computer to the first Pow-R-Bar can be up to 50', which means that a total distance of 1300' can be on-line. Greater distances are possible using short-haul modems or phone-line modems.

Pow-R-Bar is rated for 110-V AC, 60-Hz operation, and has a total capacity of 10 A.

Pow-R-Bar has a manufacturer's suggested reselling price of $149.95 alone or $159.95 with a serial cable and DE9–DB25 connector. Future release models will include a wall-mount model, a 19" rack-mount model, and a 220-V industrial model.

International Micro Electronics Group, Ltd.
155 West Tivetton Way • Lexington, KY 40503
(606) 271-0017 • Fax: (606) 245-1798 **#501**

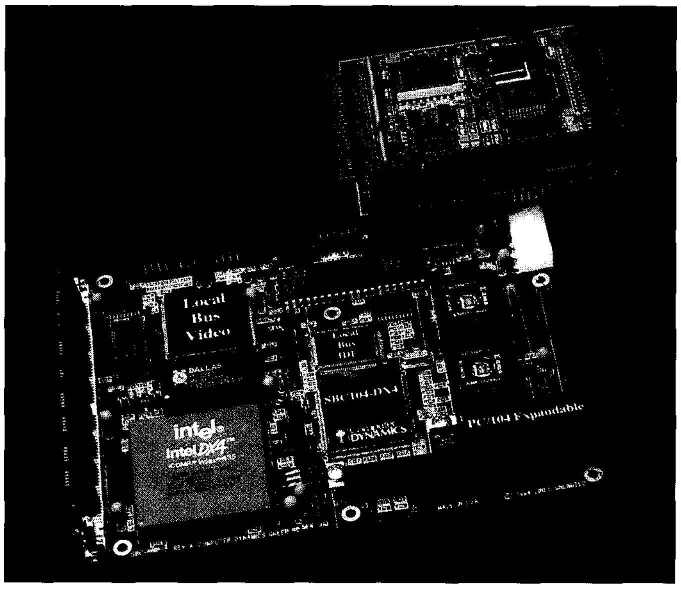


NEW PRODUCT NEWS

SINGLE-BOARD COMPUTER

Computer Dynamics has introduced the SBC104-DX **single-board computer** which features a powerful GUI engine. It combines a 486DX4 CPU at 100 MHz with state-of-the-art Local Bus video and IDE hard-disk-drive interface. This combination effectively removes the bottlenecks of video processing and hard disk I/O which slow down graphics-intensive applications.

The 486DX4 CPU is the fastest processor available for the PC/IO4 standard. The SBC 104-DX Local Bus video controller offers 32-bit video access for maximum throughput and faster graphics. Windows accelerators are built-in for up to 100 times faster Windows graphics throughput. The CPU supports both 1024 × 768 and 640 × 480 color TFT LCDs and displays 256 simultaneous colors from a palette of 262,000.



The Local Bus IDE hard-disk-drive interface runs at 33 MHz, four times faster than the original IDE interface, and has transfer rates up to 10 MBps. Strap options support a wide variety of IDE drives.

The SBC 104-DX also has on board up to 16 MB of DRAM, keyboard and speaker interfaces, a real-time clock, and the PC/104 8- or 16-bit interface. A variety of PC/104 expansion modules are available.

The SBC 104-DX is priced at $1895 for the 486SX version with 4 MB of DRAM.

Computer Dynamics
107 S. Main St.
Greer, SC 29650
(803) 877-8700
Fax: (803) 879-2030

#502

PROTOTYPE DEVELOPMENT BOARD

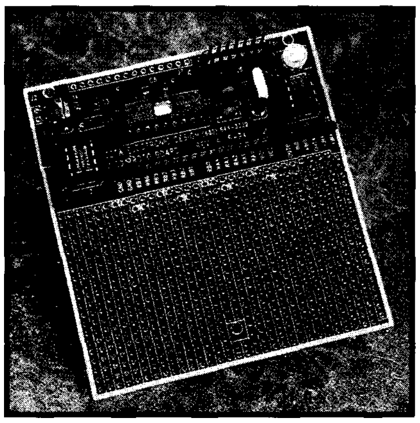
The **Zeta Z100 board** is a prototype development board designed for Microchip's PIC16C71 and PIC16C84 microcontrollers. The Z100 helps design engineers reduce their time to market by providing a quicker, more efficient method of prototype development.

The Z100 board uses a +5-V supply generated from a 9-V battery. The board holds a crystal, dual rail-to-rail I/O op-amps, an RS-485 serial driver, an alphanumeric LCD-display-module interface, and a large prototype area. Additional features include software routines which give examples of how to interface the LCD and serial port. The board fits in a standard PacTec enclosure with a battery compartment.

The PIC16C84 is pin compatible with the PIC16C71, but features an EEPROM without the A/D converter. The PIC16C84 board has a 10-MHz crystal and the PIC16C71 board, a 16-MHz crystal.

The Z100 provides connections for common LCD display modules. A potentiometer is available for contrast adjustment and software driver routines are included. A 2" x 3" prototyping area accommodates wire-wrapped custom circuitry. Interfacing to the microcontroller and op-amp is provided through 24 wire-wrap pins.

The Zeta Z100 board comes with a disk of easily modifiable software and sells for $39.95. The board with a PIC16C84 sells for $79.95 and $89.95 with the PIC16C71 EPROM version. LCD displays and enclosures are available.

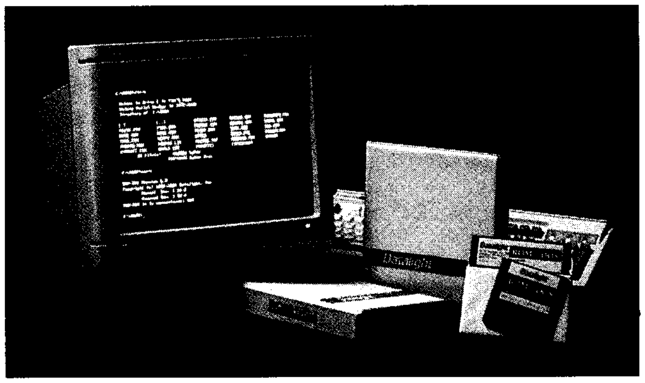


Zeta Electronic Design
7 Colby Ct., #4-193
Bedford, NH 03110
(603) 644-3239
Fax: (603) 644-3413

#503

NEW PRODUCT NEWS

SOFTWARE DEVELOPMENT KIT



Datalight has configured a special version of its ROM-DOS Software Developer's Kit for the Intel 386EX embedded processor. It includes a miniBIOS, ROM-DOS, and all of the software needed to get a DOS application up and running out of flash memory on the Intel 386EX processor evaluation board. The kit, which is being offered free for evaluation on the Intel 386EX processor, is a subset of Datalight's full ROM-DOS Software Developer's Kit which can be bought directly from Datalight for $495. An OEM license agreement is required to include software with any product.

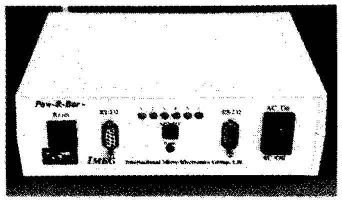
The kit includes ROM-DOS 6 (an MS-DOS 6 equivalent operating system), Datalight's miniBIOS (a no-royalty minimal BIOS), REMDISK (a remote disk utility), and ROM Disk Builder (used to load applications on the Intel 386EX processor board). The ROM-DOS files can be quickly transferred to flash memory on the board.

A 4-KB miniBIOS brings the system from power on through the initialization process and allows ROM-DOS to take control. The BIOS supports memory disks and a remote console. Full source code to the miniBIOS is included.

ROM-DOS 6 supports Windows 3.1 and local area networks. It has advanced power management for low-power systems which have APM BIOS support, and an XMS memory manager (HIMEM.SYS) for handling extended memory.

The kit can be downloaded from Datalight's BBS (206) 435-8734 or obtained by contacting Datalight.

Datalight, Inc.
307 N. Olympic Ave., Ste. 200 • Arlington, WA 98223 • (206) 435-8086 • Fax: (206) 435-0253 **#504**



Pow-R-Bar™ $149" (msrp)

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

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

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

IMEG International Micro Electronics Group, Ltd.

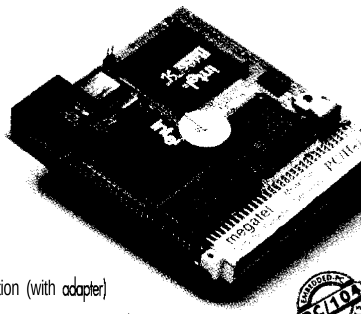
155 W. Tiverton Lexington, Kentucky 40503
P.O. Box 25007 Lexington, Kentucky 40524
800-274-8699 606-271-0017 Fax: 606-245-1798

104

WORLD'S SMALLEST

486™ DX Embedded PC with floating Point, Ethernet & Super VGA Only 4"x4"

The PC/II +*i* includes:



- 486DX™ CPU at 25MHz or 33MHz clock frequency
- Full 8K Cache with Floating Point
- Ethernet local Area Network
- Local Bus Super VGA Video/LCD
- Up to 2MBytes Flash™ with TFFS
- 4 or 16MBytes User DRAM
- PC/104 or ISA Bus compatible option (with adapter)
- 4"x 4" Format; 6 watts power consumption at +5 volt only

486DX and Flash are registered trademarks of Intel Corp as are PC, AT of IBM, megatel of Megatel Computer (1986) Corp.

(416) 245-2953

125 Wendell Ave. • Weston, Ont. • M9N 3K9 • Fax: (416) 245.6505 **megatel®**

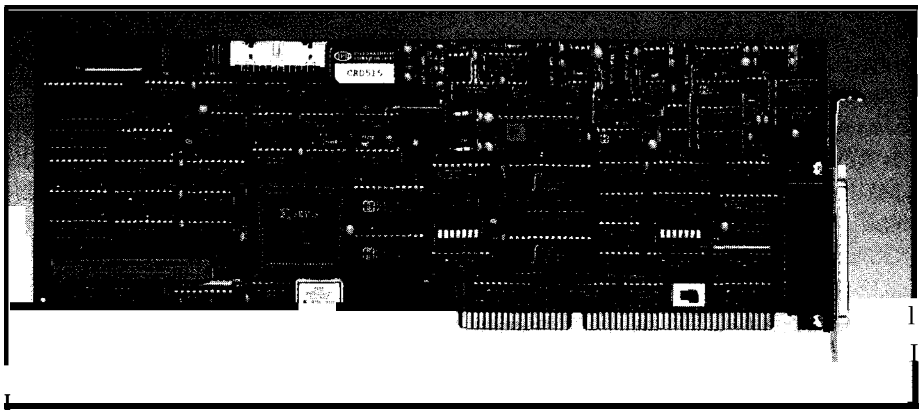
#105

NEW PRODUCT NEWS

HIGH-SPEED ANALOG INPUT CARD

A high-speed, 8-channel, 1 -MHz analog-input card has been announced by ComputerBoards. The **CIO-DAS16/M1** supports 1 MHz of continuous input to a MEGA-FIFO accessory card through DTCONNECT, pre- and post-trigger buffers of unlimited size, a 256-step programmable channel and gain queue, 24 lines of digital I/O, and three 16-bit counters.

Augmenting the speed of the card is its ability to transfer huge blocks of data when using the companion MEGA-FIFO board. By attaching a fully loaded MEGA-FIFO, up to 128 megasamples of data can be stored at top speed. The DT-CONNECT feature enables such large blocks of data to be transferred to the buffer card at the full speed of the board. Windows application software experiences no delays in concurrent programs during sample transfers.



A precise, 12-bit analog-to-digital converter transforms high-speed analog signals into their digital representation. This 1 -MHz, 0.8-µs A/D converter resolves the data to an accuracy of 1 part in 4095. Without the MEGA-FIFO, acquisition speeds of 800 kHz to system memory are attainable through use of the REPINSW command and the onboard 1 -KB FIFO.

Analog-input ranges are software programmable. The unique programming flexibility of the 256-byte channel-gain queue lets analog inputs be sampled at different ranges on different channels. Ranges may be set for bipolar and unipolar values.

The I/O lines and counters are accessible through a 50-pin, dual inline connector on the rear of the card. The I/O lines may be used for relay closure to turn on motors and valves or to latch signals into an analog-input channel of the card. The precision counters support sample pacing and event counting.

InstaCal, supplied free, is a software package which installs, calibrates, and tests all ComputerBoards' cards. For programmers, the Universal Library is an easy-to-use software library supporting all board languages in DOS and Windows.

The CIO-DAS16/M1 card sells for $999 and the Universal Library sells for $49.

Computer-Boards, Inc.
125 High St., Ste. 6
Mansfield, MA 02048
(508) 261-1123
Fax: (508) 261-1094

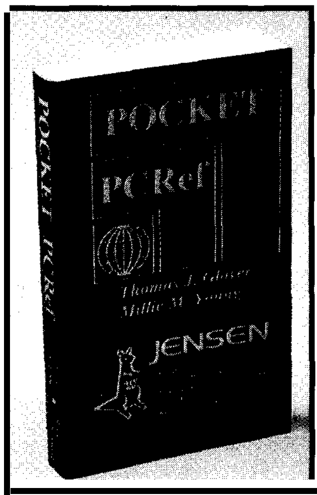
#505

POCKET REFERENCE BOOKS

Jensen Tools Inc. offers two pocket-size handbooks complete with reference information on a broad range of technical subjects. The books are about the size of a 3" x 5" postcard and less than ½" thick. The books tuck easily into a pocket, glove box, or briefcase.

The **Pocket PC Ref** (939B010) is a reference of computer information, especially PC hardware. It presents 320 pages of tables and charts on video standards, keyboard scan codes, floppy-drive and hard-drive specifications, printer codes, and CPU processor types, ASCII codes, trademarks, DOS 5.0 commands, and more.

The Pocket PC Ref sells for $14.95. A free Jensen catalog is available.

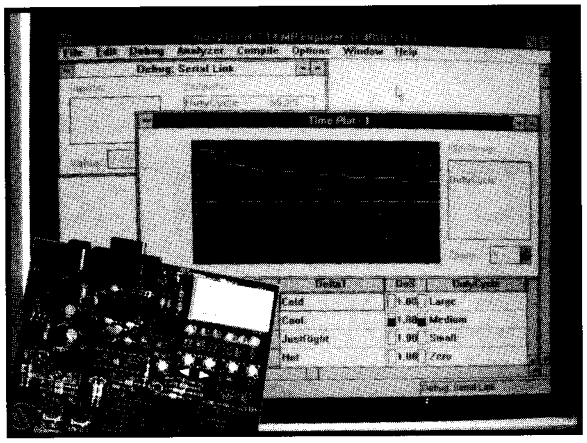


Jensen Tools, Inc.
7815 S. 46th St. • Phoenix, AZ 85044 • (602) 968-6231 • Fax: (602) **438-1690**

#506

NEW PRODUCT NEWS

FUZZY LOGIC DEVELOPMENT TOOL



Microchip Technology has introduced **fuzzyTECH-MP,** two suites of advanced PC-based development tools for creating, analyzing, simulating, and debugging fuzzy-logic applications for its PIC 16 and **17** microcontrollers. The new tool suites, developed by Inform Software Corporation, are the first fuzzy-logic development tools to include a fully functional, fuzzy-logic hardware-demonstration board. The board offers hands-on experience with fuzzy-logic system implementation and speeds the overall development process.

fuzzyTECH-MP is available in two versions: the MP Explorer for designers who need a working knowledge of fuzzy-logic system design and the MP Edition for engineers who implement more complex systems. Both versions run in Windows and feature simple point-and-click commands with powerful graphical editors.

In addition to fuzzy-logic system specification, the suites provide the graphical tools needed to simulate fuzzy solutions in real time or in response to recorded or simulated process data. Six different debug modes are included for comprehensive program test and verification. Graphical support is also provided for fuzzy system optimization, including "what if" analyses for efficiently handling large designs, transfer plots for identifying redundant rules or regions of instability, and a comprehensive data analyzer for signal processing and visualization. Once the system design is complete, the fuzzyTECH-MP suites generate assembly code compatible with MPASM, Microchip's Universal Assembler, which can be integrated into a PIC **16** or **17** application.

The MP Explorer includes all the graphics editors and tools to guide designers through the fuzzy-system design process for systems with up to two input and one output variable. The MP Edition includes this tool set with expanded flexibility for handling more complex systems of up to eight-input and four-output variables. Both suites support 8- and 16-bit resolution for input and output variables, and 16-bit computational resolution.

The fuzzyTECH-MP Explorer sells for $295; the fuzzyTECH-MP Edition sells for $995.

Microchip Technology, Inc.
2355 West Chandler Blvd. • Chandler, AZ 85224-6199
(602) 786-7200 • Fax: (602) 899-9210 **#507**

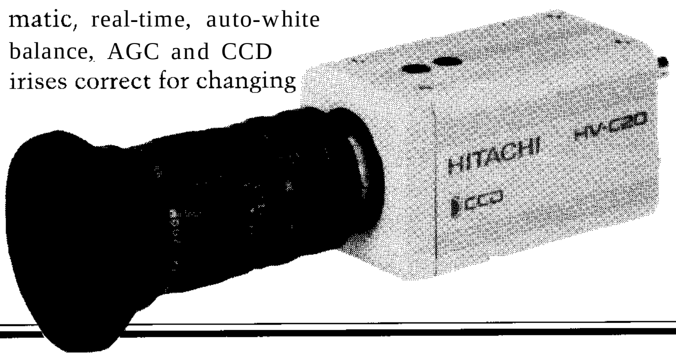
3-CCD COLOR CAMERA

A ½″ 3-CCD color camera with 410,000 pixels and a micro lens has been introduced by Hitachi. The HV-C20 is designed for the industrial user and is ideal for video conferencing, image processing, and microscope applications.

New C-mount prism optics provide a much smaller and less expensive package than previously available with 3-chip cameras. An automatic correction circuit eliminates shading errors that sometimes occur with C-mount lenses. The camera measures only 65 mm x 65 mm x 125 mm.

The HV-C20 features 700-line TV resolution, sensitivity of f8.0 at 2000 lux, a signal-to-noise ratio of 60 dB, and vertical contour correction. For fully automatic, real-time, auto-white balance, AGC and CCD irises correct for changing color temperatures and illumination levels. Field-On-Demand offers immediate output of a single field and immediate reset. A long-exposure mode is provided for those applications that require extreme sensitivity.

Protocol documentation is available for computer programming control by customers through the 2-way, RS-232C interface. Composite, Y/C, and RGB outputs on the camera provide a video interface for a wide variety of external equipment, and a gen-lock input provides synchronization of multiple cameras.



Hitachi **Denshi** America, Inc.
150 Crossways Park Dr.
Woodbury, NY 11797
(516) 921-7200
Fax: (516) 496-3718

#508

FEATURES

12 Designing Real-Time Embedded Software Using State-Machine Concepts

20 All Together Now

30 Make the Most of Your DSP-Based Sound Card

40 Design Contest

44 A RISC Designer's New Right ARM

FEATURE ARTICLE

David Tweed

Designing Real-time Embedded Software Using State-machine Concepts

Dave takes us back to the fundamentals of state-machine programming. His point: viewing real-time programming in the same way as state-machine programming gives greater clarity to the design process.

Did you know that all software is really a state-machine specification?

Think about it. All a programmer knows about the statements he or she writes is that they will be executed in a certain order. Each statement specifies the change in the computer's state between when the CPU begins and when it finishes executing the statement (state information includes the CPU's program counter, other registers, main memory, and I/O registers). This is true whether the programmer is writing machine code (1s and Os), assembly language (mnemonics), or a higher-level language like C.

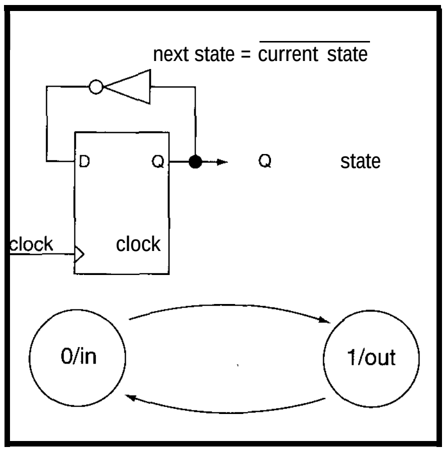


Figure I--The simplest state machine has one bit, *two* states, and no inputs.

Understanding this fundamental concept can bring a new clarity to the design process for real-time systems.

INTRODUCTION TO STATE MACHINES

The concept of a state machine goes back to the theoretical underpinnings of computing--the Turing machine, which consists of a state machine and a memory tape. In this article, a state machine includes any hardware that can be in a finite number of discrete states and the rules about how it makes a transition from one state to another.

The simplest state machine has two states: the on/off of an electronic flip-flop or the in/out of a retractable ballpoint pen. Each process has a single input that we call clock, and each time the clock is activated, the machine changes to the opposite state. The electrical output of the flip-flop or the physical position of the pen tip is the memory of the current state, and this single "bit" of storage can keep track of two states. Figure 1 represents either the flip-flop or the pen.

In general, a state machine can have no more than 2^N states, where N is the number of bits of memory available to store the state. Most state machines will have significantly fewer states, because many bit combinations will be either meaningless or redundant.

For example, a BCD counter has 4 bits of storage, but only 10 of the **16** possible states are meaningful. This isn't to say that the other six states don't exist. In fact, the designer must make sure that, if the counter happens to be in an incorrect state (say, at powerup), it can be switched to a correct one.

This is what reset pins are for, but sometimes it is sufficient to make sure that the illegal states eventually make transitions to legal states. Figure 2 shows a typical implementation of such a counter. As long as it remains in the states 0000-1001, it cannot make transitions to the states 1010–1111. However, if it should happen to powerup into one of the latter states, it will get back into the correct sequence within two transitions.

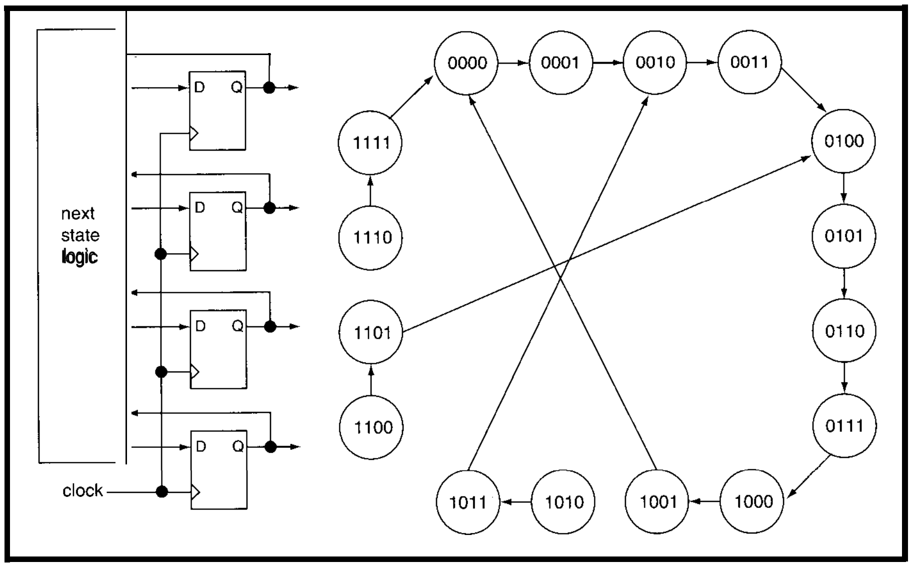


Figure P-*These diagrams show the states as* circles *and the transitions from one state to* another *as* arrows. *Each state is labeled* with the combination *of state* variables (bits) *that represents that state.*

State machines that have only one exit from each state are useful as counters, but not much else. In general, a state machine will have external inputs that can modify its behavior. For example, we might want a 2-bit (4-state) counter that has reset and enable inputs. Figure 3 shows one possible state diagram for such a counter.

Now that there is more than one possible transition out of each state, each arrow is labeled with the input conditions that cause a particular transition to be selected over another. Care must be taken to ensure that it isn't possible for two arrows to be selected by the same set of input conditions.

The clock input to the state machine is implicit: the machine makes one transition per clock pulse or edge. Sometimes an arrow leads back to the state from which it starts. This simply means that for such a combination of inputs, the machine remains in the same state.

A state machine can also be described by a table in which each line of the table describes an arrow in the diagram. There are columns for the start state, input conditions, and next state. Table 1 offers a table layout to the state machine of Figure 3.

So far, there has been no mention of output from a state machine. In the simple machines we've talked about so far, the state variables themselves are used as the output. This represents one kind of machine in which the output states are directly associated with the internal states of the machine.

In more complex state machines, often the output is in the form of *events* that are associated with the transitions from state to state rather than the states themselves. In this kind of machine, there may even be more than one transition from one state to another, differing only in the input conditions that select them and the output events that they generate. When I get to specific examples, you will see this kind of state machine.

We can continue to extrapolate this concept until we get to a microprocessor, which along with all of its internal and external memory, forms a

Current State	input Conditions	Next State
00	reset + $\overline{\text{enable}}$	00
00	reset & enable	01
01	reset	00
01	$\overline{\text{reset}}$ & $\overline{\text{enable}}$	01
01	reset & enable	10
10	reset	00
10	$\overline{\text{reset \& enable}}$	10
10	reset & enable	11
11	$\overline{\text{reset \& enable}}$	11
11	reset + enable	00

Table 1-A state-machine description in *tabular form makes it easy to verify that transition conditions are mutually exclusive.*

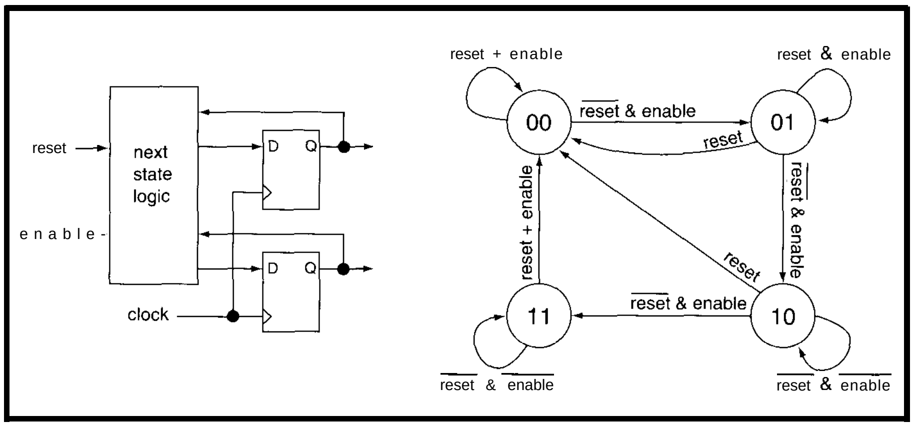


Figure 3-A *diagram for a state machine with inputs must have labeled transitions.*

complex state machine with a large number of states. A large (but slightly smaller) number of those states is completely meaningless-you can imagine filling memory over and over again with random numbers until you create a program that plays solitaire. But, this isn't an efficient way to program.

With a microprocessor, we actually have a kind of hierarchy of state machines. At the lowest level, there is the microcode that specifies a state machine with a few hundred states. This state machine, as it operates, causes registers to be updated and external memory to be read and written.

At the next level up, we see the concepts of a program counter, address register, and data register. These are really nothing more than convenient abstractions which make it easier to write machine-language programs for the processor.

At the highest level, we have languages like BASIC and C. These languages provide yet another level of abstraction. On the one hand, variables and data structures are the state-machine memory. On the other, each statement may represent dozens or even hundreds of machine-language instructions, yet each can be thought of as single state-machine transition.

STATE MACHINES IN REAL-TIME SYSTEMS

In real-time programming, the operation of the computer or state machine must be synchronized with events occurring outside the system. Often, there is more than one possible event that might occur at a given moment, and the software must make an appropriate response to the next event. The overall structure of the software should be set up to make programming responses to external events as simple as possible, while meeting the real-time constraints of those responses.

There are two common ways of accomplishing this: polling loops and interrupts. Let's consider each one in terms of state machines.

The polling loop is usually considered to be the simpler of the two, and consists of having the state machine test external inputs one at a time. The loop moves from a current state to one of two possible next states based on the value of that input. This might take the form of a conditional jump, conditional subroutine call, or conditional return statement. Such branching makes the state machine more complicated, but it still can be in only one state at a time.

Unfortunately, with the polling loop, while the state machine is testing or responding to one external input, it is ignoring all of its other inputs. Interrupts, on the other hand, get the attention of their state machine right away (assuming the priorities are arranged appropriately).

Interrupts are a way of multiplexing the hardware among two or more state machines. The main-line code specifies one state machine, while each interrupt-service routine (ISR) specifies another. When an interrupt occurs, the operation of the current machine (the main line or a lower-priority ISR) is suspended, its state is saved, and the ISR associated with that interrupt begins to operate.

The drawback of the interrupt-based system is the complexity of multiplexing the hardware among state machines. This complexity is mitigated by the availability of real-time executive kernels that handle low-level details and allow the programmer to get on with the application's real work. Even if you rolls your own kernel, the results can often be used over and over again in other applications.

The decision of whether to use polled I/O or interrupts in an embedded real-time computer is complex and doesn't always have a clear-cut answer. Next, I will explore some of the issues affecting this decision.

THE SYSTEM'S STRUCTURE

All nontrivial systems have inputs and outputs that allow them to inter-

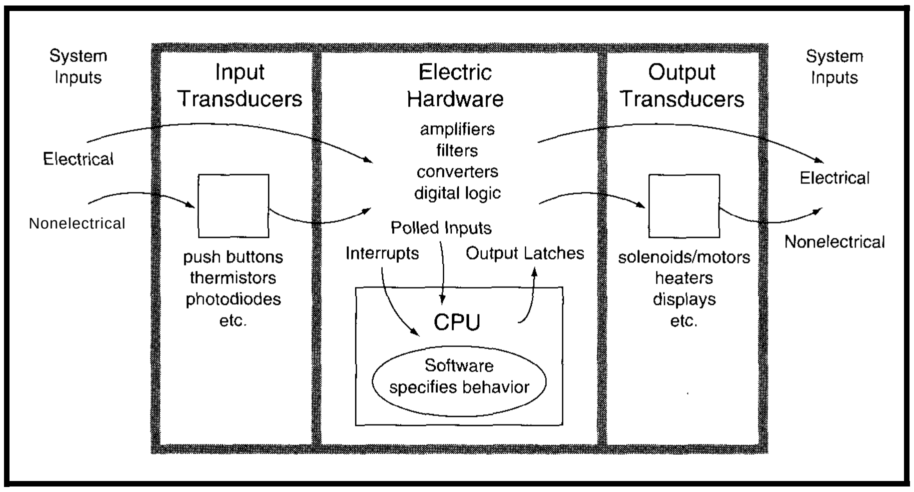


Figure 4—*Real-time systems interact with their environment in various electrical and nonelectrical ways.*

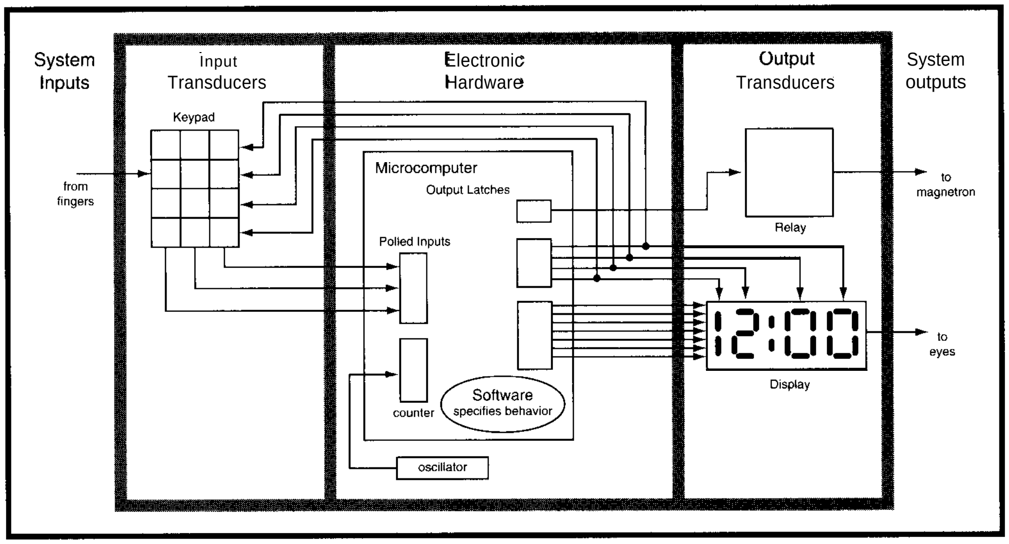


Figure 5—*The* microwave oven interacts with the *user* via keypad and display, and *controls* a magnetron to cook food.

act with their environment. In the case of electronic systems-computer-based systems in particular--the interactions must be either electrical or converted into electrical form. Figure 4 shows a generic hardware and software system with inputs and outputs.

The electronic hardware performs conversions of electrical signals (amplification, analog-to-digital, digital-to-analog, etc.) and implements state machines of various kinds, including the one on which the software runs. The software, in turn, determines the behavior of the CPU state machine. Input transducers can be mechanical (switches), temperature (thermistors), and light (photodiodes), among many others. Similarly, output transducers include solenoids and motors, heating elements, and displays of various kinds.

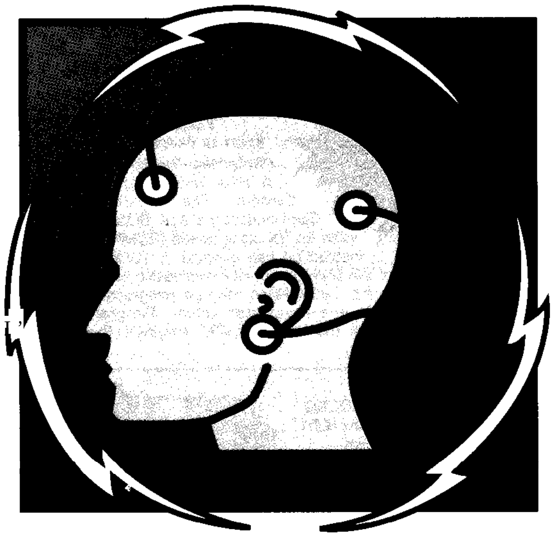
So far, my description has been very abstract. Let's bring things down to earth by considering some real systems.

One common real-time computer that many people use daily is the controller in their microwave oven. When we put it into the context of our generic system, it has input transducers (a keypad), output transducers (a display and magnetron on/off control), and most likely a single-chip microcontroller driven by a crystal (see Figure 5). Since it also needs a way to measure the passage of time, we add a hardware counter that is advanced by the oscillator (most microcontrollers have such a counter built-in).

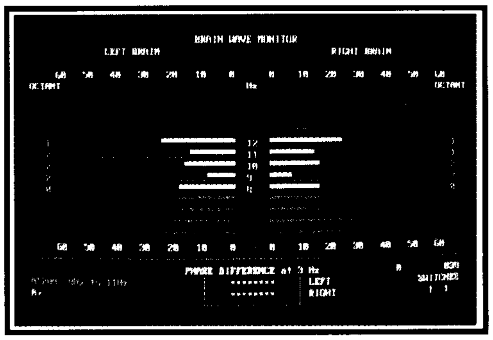
The first step in designing the software for this system is understand-

HAL-4

EEG Biofeedback Brainwave Analyzer



The HAL-4 kit is a comnlete battery-operated 4-channel electroencephalograph (EEG) which measures a mere 6" x 7". HAL is sensitive enough to even distinguish different conscious states-between concentrated mental activity and pleasant daydreaming. HAL gathers all 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 straightforward. It samples four channels of analog brainwave data 64 times aer second and transmits this digitized data serially to a PC at 4800 bps. There, using a Fast Fourier Transform to determine frequncy, amplitude, and phase components, the results are graphically displayed in real time for each side of the brain..



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

to order the HAL-4 Kit or to receive a catalog,
CALL: **(203) 875-2751** OR FAX: **(203) 875-2204**

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

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

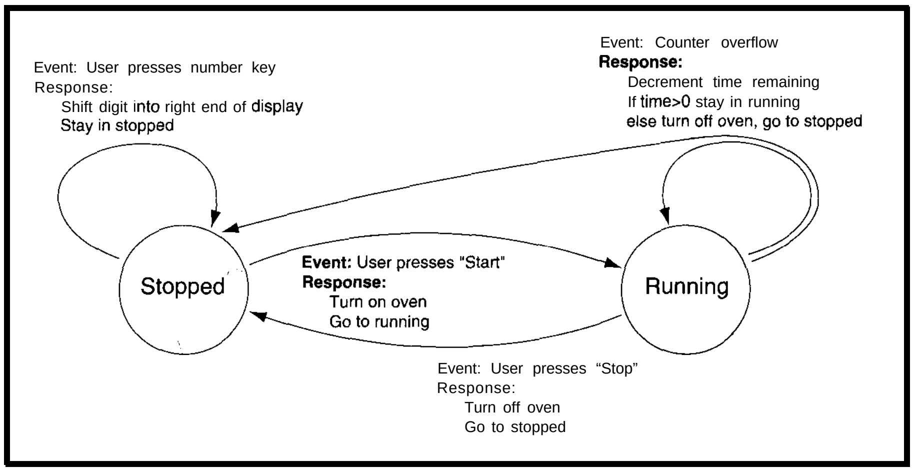

Event: User presses number key
Response:
Shift digit into right end of display
Stay in stopped
Event: Counter overflow
Response:
Decrement time remaining
If time>0 stay in running
else turn off oven, go to stopped
Stopped
Running
Event: User presses "Start"
Response:
Turn on oven
Go to running
Event: User presses "Stop"
Response:
Turn off oven
Go to stopped

Figure 6—*A* state-machine diagram, in which *the* outputs are events, *ties the* responses *to the* fransifions *rather than to the* states.

ing what the overall system must do. We want the user to be able to enter a time period, start the oven, and have it stop by itself when the specified time has elapsed. (We'll leave niceties like a time-of-day clock and temperature sensing for the competition to worry about for now.)

The user enters a sequence of digits on the keypad, sees them in the display, and then presses "start." The oven runs while the display counts down and stops when the display reaches zero. The user can press "stop" while the oven is running to immediately stop it and freeze the count at its current value. Such high-level system behavior can be easily described with a state diagram (see Figure 6).

Since the hardware is so simple, the computer clearly has to implement these behaviors directly in software, and it must do so in terms of the details of operating the keyboard, counter, display, and relay. It must constantly scan the keyboard to receive input from the user. When the oven is on, it tracks the state of the counter and must constantly scan the display to show the user what's happening. Finally, it must turn the magnetron on and off at the appropriate times.

FUNCTIONAL SPECIFICATION

What we have been doing so far is coming up with a functional specification for the software (and not a line of code has been written yet!). As the process continues, we divide each high-level behavior into its subtasks until the subtasks are the lowest-level primitive operations of the system. You may have heard of this before— it's called top-down design.

The lowest-level primitives in our microwave oven are the following:

- Activate a row on the keyboard and check whether any of the switches in that row are closed
- Activate a digit on the display and turn on the correct segments
- When the oven is on, check whether the counter has overflowed

We need to come up with some time limits on the operation of these primitives. For example, a user expects the oven to respond more or less immediately when he presses a button, so this puts a lower limit on the keyboard-scanning speed.

Also, we need to worry about things like debouncing the mechanical switch, which in turn puts an upper limit on the speed. Generally, a good compromise between apparent response speed and the typical characteristics of the switch is 10–20 ms. We want to scan the entire keyboard at least once during this time, and preferably a few times. Since our keyboard has four rows, let's scan one row per millisecond.

Similarly, our J-digit display needs to be refreshed at a rate that's high enough to make it seem to be continuously lit. Unless it uses technology that has optical persistence (like a

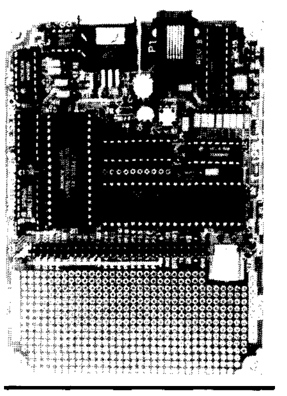
TURBO-128
THE *NEXT GENERATION* EMBEDDED CONTROLLER

*** * NO DEVELOPMENT TOOLS REQUIRED * ***
READY TO PROGRAM IN BASIC OR ASSEMBLY

Photronics Research introduces the T-128: A True Single Board BASIC Development System The T-128 is based on Dallas Semiconductor's new 8051-compatible DS80C320 With its 2X clock speed (25MHz) and 3X cycle efficiency, an instruction can execute in 160ns: an 8051 equivalent speed of 62.5MHz!!! Equally impressive is the T-l 28's high-speed NVRAM interface. Any of the 128K RAM may be programmed directly from a PC file through the console: eliminating EPROMs and associated tools. Program Development has never bee" faster or more convenient, even with the finest EPROM emulator.The T-128 features PORT 0 bias and EA-select for DS87C520 upgrade.

PROCESSOR
•Dallas Semiconductor's DS80C320
•300% more efficient than the 8051
*Three 1 6-bit Timer/Counters
•13 Interrupts (6 Ext, 7 Int)
.A second 16-bit Data Pointer
•384 Bytes of Internal RAM
*Programmable Watchdog
•Brownout Protection
•Power-Fail Reset/Interrupt
•Power-On Reset
-Fully supported by ***Franklin C51***

MEMORY
• Entire 128K Memory Map populated with *fast* NVRAM (64K DATA+64K CODE]
•All memory programmed on-board
•Partitionable as CODE/DATA/OVERLAID
•Code Space is Write-Protectable
•State-of-the-Art Data Protection



Only 5" x 3-3/4" • 500-hole Proto Area • Console/Power connected by a single 4-conductor telephone wire (very convenient)

BASIC520
Modified BASIC52 Interpreter (BASIC-520]
Now Fast Enough for New Applications
•Stack BASIC Programs and Autorun
•CALL ASM Routines for Maximum Speed

I/O
•Three 8-bit Parallel Ports
*Two Full-Duplex RS232 Serial Ports
•Decoded Device I/O Stmbes
•50-Pin Bus connector

UPGRADE
•DS87C520 processor (33MHz)
•Instruction cycle: 121 ns
•8.25 MIPS
•8051 equivalent 82.5 MHz
•Internal 16K ROM/1K SRAM

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

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

#106

Task	How Often?	How lona?	CPU utilization
Scan keyboard row	every 1 ms	100 µs	10%
Scan display digit	every 1 ms	50 µs	5%
Check counter	every 1 ms	70 µs	7%
Total:		**220 µs**	**22%**

Table 2—*A* microwave *oven* indicates that a polling loop would be most appropriate.

CRT), a refresh rate of a few hundred hertz is required to eliminate flickering and other annoying artifacts. If we can display one digit per millisecond, the overall refresh rate will be 250 Hz, which should be fine. Do you see a trend developing here?

Finally, our CPU runs at 1 .OOO MHz, and our hardware counter divides the CPU cycles by 1000, which means it's going to overflow once per millisecond (aha!). It would seem at first blush that it might be appropriate to use the counter overflows at all times to synchronize the operation of the other primitives. But, we only have half the story so far. We also need to know how much CPU processing is associated with each of these primitive tasks.

When a key is pressed, we are going to either shift a digit into the display or start/stop the oven. We will also have to update some internal state information associated with debouncing the key. Call it 100 instructions for the first cut.

To refresh a display digit, we need to update our digit counter, generate a digit strobe, look up the digit in a table to generate its 7-segment pattern, and send that pattern to the display. Call it 50 instructions for now, although this is probably generous for most real microprocessors.

Finally, when the counter overflows (and the oven is running), we need to increment an internal millisecond counter and decrement the display every time it reaches 1000 (and possibly turn off the.oven). Let's say 70 instructions.

When we put all of this information into a table, several things immediately become obvious (see Table 2). First, all the tasks need to run with the same 1-ms period. Second, even if all three tasks require their maximum execution time, they can be completed within the 1-ms period. These are good indications that a simple polling loop is a good overall architecture for the firmware of the oven. The control flow might look something like Figure 7.

A SECOND EXAMPLE

Another example of a real-time system is an audio-spectrum-analyzer display that might be used in a high-end home-audio system. In some ways, it is simpler than the microwave oven, yet it brings to light some problems which require a different approach.

The hardware consists merely of an analog-to-digital converter (ADC), a DSP chip, and a 32 x 128-LED display. We need a DSP to compute the Fast Fourier Transform, but you can think of it as a fast microprocessor. Figure 8 shows the analyzer system in its generic format.

The system has three tasks: collect values from the ADC, compute a 256-point FFT, and display the results on the LED display. The ADC produces 44,100 values each second that must be saved in groups of 256. When a group is complete, the second task computes the FFT, and when the results are available, the third task displays them one at a time on the multiplexed display. About 172 groups ($^{44,100}/_{256}$) are processed each second (one every 5.8 ms).

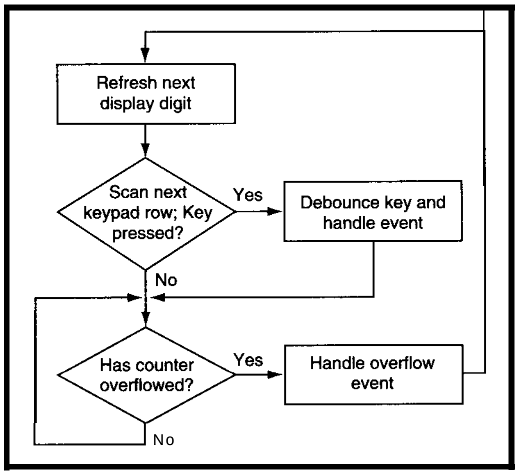
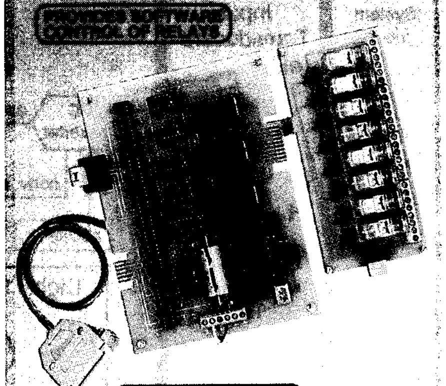

Refresh next display digit
Scan next keypad row; Key pressed?
Yes
Debounce key and handle event
No
Has counter overflowed?
Yes
Handle overflow event
No

Figure 7—*The top-level flowchart for the* microwave *polling loop shows each task as a box.*

RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

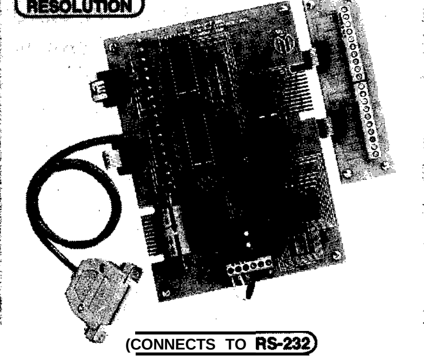


(CONNECTS TO RS-232)

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

ANALOG TO DIGITAL

8, 10 & 12 BIT RESOLUTION



(CONNECTS TO RS-232)

ADC-16 A/D CONVERTER' (16 **channel/8 bit)..$** 96.96
ADC-8G A/B **CONVERTER*** (6 **channel/10 bit).$124.90**
Input voltage, amperage, pressure, energy usage, **joysticks** and a wide **variety** of other types of analog **signals. RS-422/RS-485 available** (lengths to **4,000').** Call for info on other A/D configurations and 12 bit converters **(terminal** block and cable **sold** separately).
ADC-8E TEMPERATURE INTERFACE' (6 **ch)..$ 139.95**
Includes term. block **&** 6 temp. sensors (-40' to 146' **F).**
STA-6 **DIGITAL** INTERFACE'(6 channel) . . . . **$ 99.95**
Input on/off status of relays, switches, **HVAC** equipment, security devices, smoke detectors, and other devices.
STA-8D TOUCH TONE INTERFACE'............... **$** 134.90
Allows callers to **select** control functions from any phone.
PB-4 **PORT** SELECTOR (4 channels **RS-422)....$ 79.95**
Converts an **RS-232 port** into 4 selectable **RS-422** ports.
CO-466 **(RS-232** to **RS-422/RS-485 converter)** $44.95

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

FULL TECHNICAL **SUPPORT...provided** over the telephone by our staff. Technical reference 8 disk including test software **&** programming examples in Basic, C and assembly are provided with each order.

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

CONNECTS TO RS-232, RS-422 or **RS-485...use** with IBM and compatibles, Mac and most computers. All standard baud rates and protocols (50 to 19.200 baud).

Use our **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 43215-5438

#107

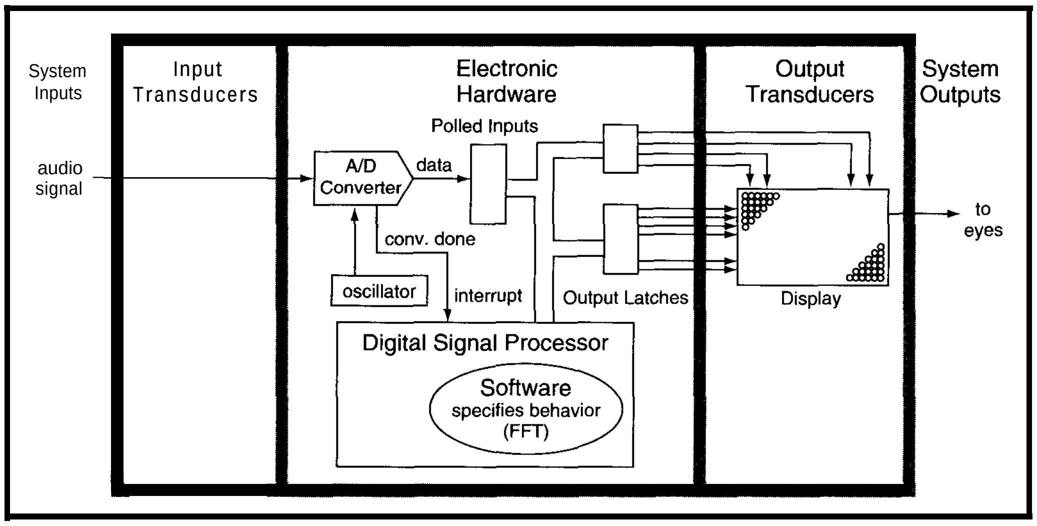


Figure 8—*The spectrum analyzer takes an audio signal as input anáproduces a dot-matrix display for the user's eyes*

The data-flow diagram also helps bring out potential problems such as having two processes that update the same data structure (two arrows pointing to the same box). In this situation, care must be taken to ensure that only one of the processes can make its update at a time. If one process interrupts the other, then the structure can be left in an inconsistent state with some data from one process and some data from the other.

The display consists of 128 columns, and it would be most convenient to tie the column-refresh rate to that of the input rate. Since there are 256 input values per group, the display will actually get refreshed twice per group, for an overall rate of 344 Hz, which should look fine.

Now we hit a snag. It takes the DSP 5 ms to calculate our 256-point FFT, yet we need to take in an ADC value and refresh a display column every 1/44,100 s or 22.7 µs. While we could rewrite our FFT to pause at the appropriate points to take care of these activities, the resulting code would violate all the principles of modular programming and be next to impossible to debug or modify.

This is where interrupts come to our rescue. We take advantage of the hardware's ability to multiplex itself between two software programs-one program does nothing but compute FFTs and another handles I/O. Obviously, these two programs need to communicate with each other, and the trick to using interrupts effectively is understanding just how much (or how little) communication is required. A data-flow diagram is an incredibly useful tool for this.

THE DATA-FLOW DIAGRAM

The data-flow diagram is somewhat analogous to the flowchart or control-flow diagram, but rather than showing how the execution proceeds in time, it shows the flow of information among processes.

The processes are shown as circles and the information or data structures are shown as boxes. Circles are connected only to boxes and never directly to other circles. An arrow drawn from a circle to a box indicates that the process writes or updates that data structure. An arrow from a box to a circle indicates that the process reads the data structure.

The processes are assumed to be running in parallel, although in most real systems, they are really just multiple tasks running on a single CPU. The important thing is that the data-flow diagram does not specify the order in which things happen. Hence, it helps to make clear what communication among the processes is required so that things happen in a particular order when needed.

Figure 9 shows the data-flow diagram for the audio-spectrum analyzer. There are three processes, two double buffers, and a l-bit flag that controls which set of buffers is being used by each process at a given time.

The input routine collects values from the ADC into input buffer #1. When that buffer fills, it sets the flag and begins filling input buffer #2. When the second buffer fills, it clears the flag and goes back to buffer #1. Similarly, the output routine sends the data from output buffer #l to the LED display when the flag is cleared and sends data from buffer #2 when it is set.

The FFT routine has four states:

- wait for the flag to be set

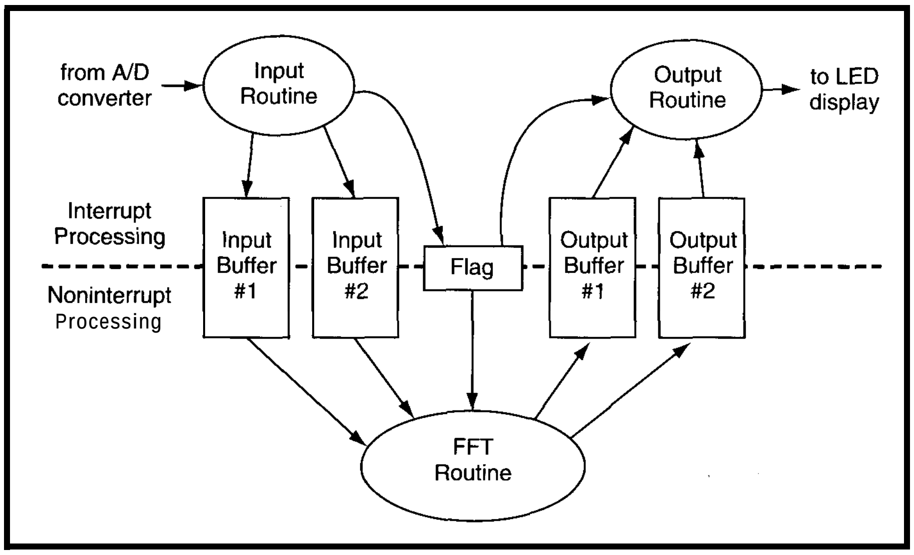


Figure 9—*The data flow diagram for the spectrum analyzer shows how to partition the individual activities.*

- calculate the FFT over input buffer #1, placing the result into output buffer #1
- wait for the flae to be cleared
- calculate the FFT over input buffer #2, placing the result into output buffer #2, then go back to state 1

You can see that this routine and its programmer can be essentially unaware of the actual I/O activity.

The hardware switches between the two software programs controlled by the interrupt, which is generated every time the ADC has a new value. The activities of the FFT are suspended, the new ADC value is stored, and new row and column data are output to the display. As long as the total CPU time, including the context switch, doesn't exceed 100%, then the analyzer will work well (see Table 3).

This may seem like high utilization to someone used to working with general-purpose microprocessors. But, since there is no variability in the execution times of any of the tasks, there's no real need for any padding.

Task	How often?	How lona?	CPU utilization
Store ADC value	every 23 µs	0.5 µs	2.21%
Scan display column	every 23 µs	1.0 µs	4.41%
Compute FFT	every 5.8 ms	5.0 ms	86.13%
Total:			92.75%

Table 3—*The task list for the analyzer shows why we need to use an interrupt and verifies that the CPU is not overloaded.*

GENERAL GUIDELINES

Let's take what we've seen here and try to formulate some general guidelines about polling versus interrupts. Of course, there are no hard-and-fast rules, and very often a hybrid approach is called for.

- If all of the tasks in the system can be made to run equally often, and the total execution time of all the tasks is less than that period, then polling will generally produce a simpler system.
- If the tasks run with widely varying periods, or the periods cannot be synchronized with each other, interrupts are strongly preferred.
- If any one task takes longer to run than the minimum periodicity of any other task, then interrupts are almost certainly required.

The real trick comes in not using more interrupts than you really need. For example, in a complex system, there might be a group of tasks which fall under the first guideline when taken out of the context of the system. It might be wise to tie the entire group to a single-timer interrupt whose ISR executes them in sequence. You can then go back and evaluate the remaining tasks in light of the resulting simpler system context.

CONCLUSION

By considering the embedded computer and its firmware as a state machine and seeing how the firmware specifies state-machine behavior, we get a better feel for how the software and hardware interact. Data-flow and state diagrams help make the structure of the system and constraints on its implementation clear.

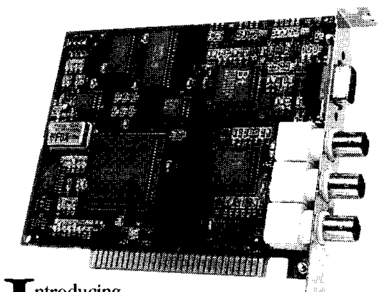
We have seen how to apply top-down design methodology to real-time systems, starting with overall system behavior and ending with the lowest-level primitive operations that are available. In addition, we have developed some guidelines that help to guide the decision about whether to use polled I/O versus interrupts in system design.

Dave Tweed has been developing real-time software for microprocessors for more than 18 years, starting with the 8008 in 1976. His system design experience covers the gamut from supercomputers and workstations to microcomputers and DSPs. He may be reached at dave.tweed@circellar.com.

IRS

401 Very Useful
402 Moderately Useful
403 Not Useful

PRECISION FRAME GRABBER FOR ONLY $495*



Introducing the CX100 precision video frame grabber for OEM, industrial and scientific applications. With sampling jitter of only ±3 nS and video noise less than one LSB, ImageNation breaks new ground in imaging price/performance. The CX100 is a rugged, low power, ISA board featuring rock solid, crystal controlled timing and all digital video synchronization. Software developers will appreciate the simple software interface, extensive C library and clear documentation. The CX100 is a software compatible, drop-in replacement for our very popular Cortex I frame grabber. Call today for complete specifications and volume pricing

ImageNation Corporation
Vision Requires Imagination
800-366-9131

— CX100 FEATURES —
- Crystal Controlled Image Accuracy
- Memory Mapped, Dual-Ported Video RAM
- Programmable Offset and Gain
- Input, Output and Overlay LUTs
- 512x486 or Four Images of 256x243 (CCIR 512x512 & 256x256)
- Monochrome, 8 Bit, Real Time Frame Grabs
- Graphics Overlay on Live or Still Images**
- External Trigger Input
- RGB or B&W, 30 Hz Interlaced Display
- NTSC/PAL Auto Detect, Auto Switch
- VCR and Resettable Camera Compatible
- Power Down Capability
- BNC or RCA Connectors
- Built-In Software Protection**
- 63 Function C Library with Source Code
- Text & Graphic Library with Source Code
- Windows DLL, Examples and Utilities
- Software also available free on our BBS
- Image File Formats: GIF, TIFF, BMP, PIC, PCX,TGA and WPG

** THESE OPTIONS AVAILABLE AT EXTRA COST
*SINGLE UNIT PRICE

P.O. BOX 276 BEAVERTON, OR 97075 USA PHONE (503) 641-7408 FAX (503) 643-2458 BBS (503) 626-7763

#108

All Together Now

Writing Multiuser Application Sofware for ACCESS.bus

FEATURE ARTICLE

David Rodgers & Peretz Tzarnotzky

With ACCESS.bus, up to 125 users can simultaneously interact on one host computer. A close look at hardware and software architectures prepares you for the ultimate challenge: create your own game!

ACCESS.bus is a serial communication protocol between a computer host and its peripheral devices. It provides a simple, uniform, and inexpensive way to connect peripheral devices such as keyboards, mice, joysticks, modems, monitors, and printers to a single computer port.

ACCESS.bus (a bus for connecting ACCESSory devices to a host system) was defined and developed by Digital Equipment Corporation. DEC offered it to the computer industry as an open standard, enabling any vendor to implement it on host systems or in peripheral devices without fees or royalties.

Although ACCESS.bus protocol architecture is an open industry standard, it is controlled and maintained by the ACCESS.bus Industry Group (ABIG). Significant corporate and individual members, representing all facets of the computer industry, participate in the evolution, definition, and proliferation of ACCESS.bus technology.

ACCESS.bus is a system for connecting up to 125 I/O devices to a single port on a host computer. Interface adapters to ISA-based PC systems and SPARCstation SBus systems facilitate easy connection to common hardware installations. ACCESS.bus peripheral I/O devices (keyboards, mice, trackballs, etc.) are already available from leading manufacturers such as Logitech, Key Tronic, and Lexmark, to name a few. Full peripheral and software application development tools are also available for easy ACCESS.bus product development. System-board-level adaptations of ACCESS.bus are currently being evaluated by several personal computer systems manufacturers.

As well, the ACCESS.bus protocol is general enough to accommodate a wide range of specialized vertical applications including point of sale, education, embedded control, virtual reality, and PC-based games. See Figure 1 for the typical ACCESS.bus system.

ACCESS.bus data-transmission technology (clock, data, power, and ground) provides microcontroller integration on various open and

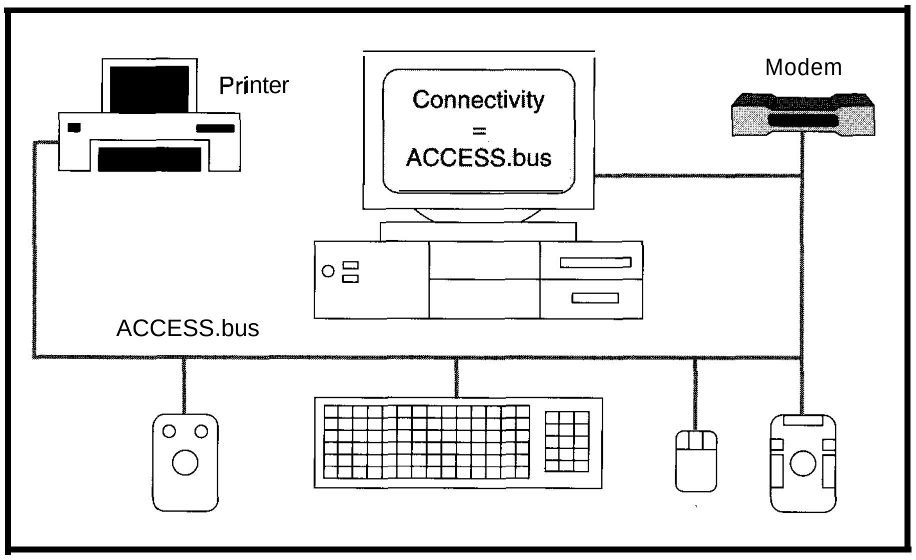


Figure 1—*In a typical ACCESS.bus desktop configuration, the usual nest of cables required to connect external peripherals is replaced by a single four-conductor cable.*

proprietary operating platforms and peripherals. From medical imaging and training systems to multiplayer, multimedia computer games, from point-of-sale systems integration and data collection to set-top cable converters, ACCESS.bus has evolved into a versatile and powerful tool.

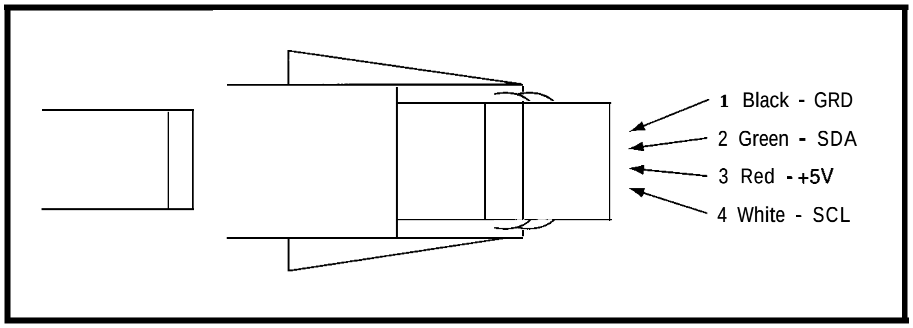


Figure **2**—*ACCESS.bus* uses a *four-conductor,* shielded, modular connector similar to *a telephone* RJ-11.

MULTIPLAYER GAMES

With the ACCESS.bus hardware interface defined and stable, software application opportunities are now emerging. Three key market areas have initial momentum in application development: multiplayer PC games, training and simulation, and education.

Computer Access Technology Corporation (CATC) developed two multiplayer games to show the features and benefits of the ACCESS. bus. The Windows versions of blackjack and Chinese checkers host up to six simultaneous players. These games highlight the simultaneous and independent interaction of multiple players, each with his or her own I/O devices. Both these games use multiple mice as their primary input device. Joysticks, trackballs, tablets, or arrow keys can also serve as the device-of choice.

These games demonstrate ACCESS.bus technology, offering insight into additional game possibilities. Interactive video games, in which combatants participate seated adjacent to each other and not through a network link, offer participants the opportunity to heighten reactive and cognitive-interactive computer skills while also experiencing the emotional intensity of shoulder-to-shoulder competition and collaboration.

EDUCATIONAL SOFTWARE

Though most people queried believe that computer games are more recreational than educational, early indications of the educational benefits of games using interactive computer skills show that games enhance learning.

ABIG hosted a Software Creators Contest over Internet. Several abstracts for ACCESS.bus software applications were submitted for review. ACCESS.bus technology games were predominant. Of the submissions, four finalists were picked. Finalists were chosen according to the perceived functionality and benefit of their ideas. Notably, the proposals' underlying theme involved interactive learning tools disguised as games.

Imagine the benefits of having the entire classroom connected to a single CPU. Schools would only need to provide individual keyboards and displays, not an entire CPU, for each student. Interactive teaching, on-line tutorials, and instant feedback on lesson plans are only some of the potential benefits. It would be easier to monitor the students' learning process. Student-specific course curricula could be administered and monitored with greater expediency and accuracy. Most importantly, the budget required to supply a single school with computers could be spread across the entire school district.

With ACCESS.bus, educational applications for text and document storage and retrieval now offer a viable alternative to the hard-bound book. Standardized achievement tests can be distributed, graded, and the results tabulated with greater simplicity and less cost. While the technologist and entrepreneur might find this new market opportunity appealing, so does the taxpayer who endorses the applications because it makes better use of the public coffers and increases the potential educational benefit and knowledge retention of the student.

TRAINING AND SIMULATION SOFTWARE

ACCESS.bus is also useful for more advanced forms of training and technical orientation. The first flight simulator available to the desktop computer was a derivative of an actual military training tool. Unfortunately for the simulation program, the "student" was only able to play against the computer and not other humans. No amount of "fuzzy logic" is yet able to adequately imitate the dynamic of human behavior. Virtual reality does offer significant promise in allowing many people to interact with each other in simulated training or game conditions.

ACCESS.bus offers the virtual developer a complete hardware, software, and application layer interface with an easy and cost-effective cabling and connector interface. The programmer can realize the full potential of a given creation without prohibitive implementation costs.

ACCESS.bus LAYERING

The ACCESS.bus standard includes a physical layer as well as several software layers. The physical layer defines the transmission medium and connectors for the bus (electric signals, cables, and connectors) and the basic message format (Start, Stop, Acknowledge, Arbitration, etc.). In addition, ACCESS.bus specifications describe base and application protocols in communications between peripheral devices and the code on a host computer.

THE PHYSICAL LAYER

At the hardware level, ACCESS. bus physical layer is based on the well-established Inter-Integrated Circuit (I^2C) serial bus developed by Philips/ Signetics. The serial bus architecture features a single data line which

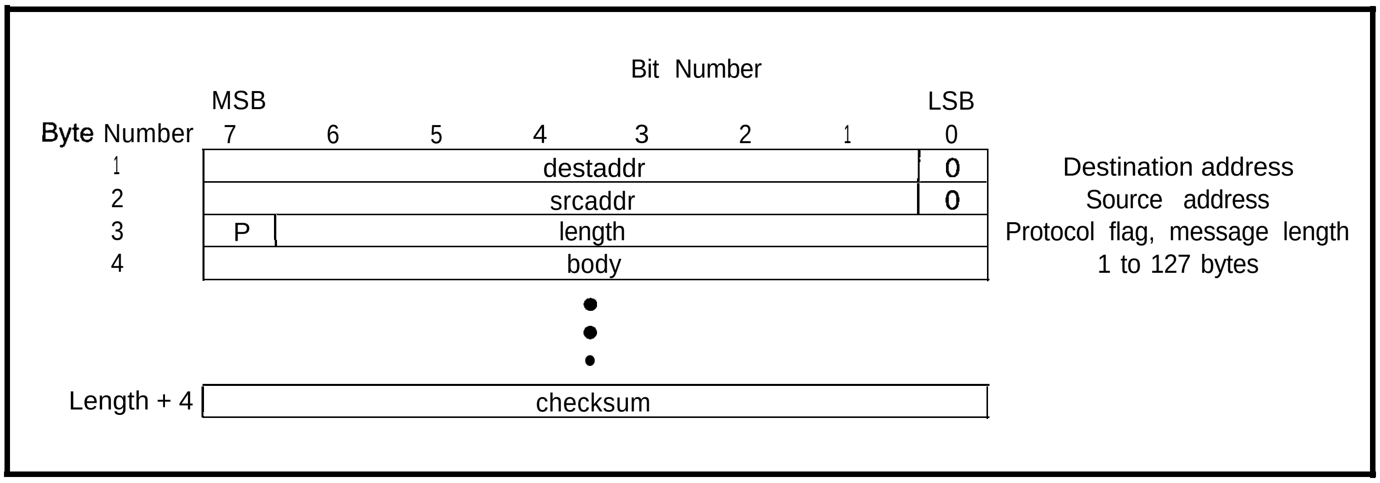

Bit Number
MSB
LSB
Byte Number 7 6 5 4 3 2 1 0
1 destaddr 0 Destination address
2 srcaddr 0 Source address
3 P length Protocol flag, message length
4 body 1 to 127 bytes
Length + 4 checksum

Figure 3—*The ACCESS.bus message packet includes a standard header, a variable-length body, and a checksum.*

carries one bit of information at a time. This, of course, results in lower costs for cabling, connectors, and controller circuitry.

The simple and efficient I²C protocol defines a symmetric, multi-master bus on which arbitration among contending masters is effected without losing data. I²C cooperatively synchronizes the serial clock for exchange of data between bus partners with different maximum clock rates. Addressing, framing of bits into bytes, and byte-acknowledgment by the receiver are all defined by a bus-transaction scheme within the I²C protocol. I²C microcontrollers handle the logical requirements of bit-level handshaking.

Pictured in Figure 2 is the ACCESS.bus physical connection. The shielded cable contains four wires: serial data (SDA), serial clock (SCL), power (+5 V), and ground (GND). It uses standard, shielded, modular connectors available from AMP and Molex. This shielding of the cables and connectors enables ACCESS.bus to meet FCC radiation and ESD requirements.

A typical ACCESS.bus device has two connectors so that devices may be chained on the single bus. Hand-held devices may have a captive cable joined to the bus trunk with a T connector. The serial-data (SDA) and serial-clock (SCL) lines work together to define the information carried on the bus.

The physical layer can support clock rates up to 100 kbps.

BASE PROTOCOL

The base protocol establishes the nature of ACCESS.bus as an asymmetrical interconnect between a host computer and a number of peripheral devices. The host plays a special role as a manager of the ACCESS.bus. Data communication is always between host and peripheral device and never between two peripherals. Although the I²C protocol provides for mastery by either the sender or the receiver of a bus transaction, the ACCESS.bus protocol defines masters as exclusively senders and slaves as exclusively receivers. Of course, the host and all the devices are both master senders and slave receivers at different times.

The ACCESS.bus base protocol defines the format of an ACCESS.bus message envelope, which is an I²C bus transaction with additional semantics, including checksum reliability control. The base protocol defines a set of control and status message types, which are used in the configuration process.

Figure 3 gives the ACCESS.bus message format. The first byte in the message is the receiver's unique address. The second byte contains the transmitter's unique address. The third byte of a message comprises two fields. Bits 2-7 provide a byte count for the body of the message. Thus, a message body can have O-127 bytes and is followed by a checksum byte for error control. The checksum is the bitwise X 0 R of all the preceding bytes of the message.

The high-order bit of the third byte is a protocol flag (P), which distinguishes between data stream messages (P = 0) and control and status messages (P = 1). Datastream messages carry the application information exchanged between the device and the host. The control and status messages manage the ACCESS.bus protocol. As well, the base protocol defines a set of nine control and status message types used in the configuration process (see Table 1).

Two of the unique features of this configuration process include *autoaddressing* and *hot plugging.* With autoaddressing, devices are assigned unique bus addresses in the configuration process without the need for setting jumpers or switches on the devices. Hot plugging enables devices to be attached or detached while the system is running.

APPLICATION PROTOCOL

The application protocol is the highest level of the ACCESS.bus protocol. It defines message semantics that are specific to particular functional types of devices. Different device types require different application protocols.

So far, application protocols have been defined for three device classes: keyboards, locators, and text devices. Each of these predefined classes is designed to be broad. The keyboard device protocol, for instance, defines standard messages for reporting keystrokes and controlling keyboard peripherals. The protocol attempts to

a) Computer-to-device Messages	Message Purpose
`Reset()`	Force device to power-up state and default ACCESS.bus address.
`Identification Request0`	Ask device for its identification string.
`Assign Address(IDstrng, new addr)`	Tell device with matching identification string to change its address to new address.
`Capabilities Request(offset)`	Ask device to send the fragment of its capabilities information that starts at offset.
`Enable Application Report`	Enable or disable a device to send application reports to the host computer.
`Presence Check`	Check if the device is present on the bus at the specific address.

b) Device-to-computer Messages	Message Purpose
`Attention`	Inform computer that a device has finished its powerup or reset test and needs to be configured.
`Identification Reply(ID string)`	Reply to identification request with device's unique identification string.
`Capabilities Reply(offset, data frag)`	Reply to capabilities request with data fragment which includes a fragment of the device's capabilities string. The computer uses offset to reassemble fragments.

Table 1—*The ACCESS.bus base procotol defines nine control* and status messages *used in the* configuration process.

define the simplest set of functions from which industry-standard keyboard interfaces can be built.

The locator device protocol defines a set of standard messages for reporting locator movement and key-switch activation for mice, tablets, and other positioning devices. Complex locator devices can be modeled as a combination of basic devices or as their own device driver.

The text device protocol provides a simple way to transmit character or binary data to and from stream-oriented devices such as a printer, bar-code reader, or modem. The sequential-character-stream model also serves as a common denominator for connecting RS-232 interface devices.

Designing devices that conform to the general device semantics is a major advantage. Device-specific software, both in the device-resident firmware and the driver software needed for the host operating system, can be accessed by the new device.

In the future, it is anticipated that more device-specific application protocols will be defined under the aegis of the ABIG. As well, any device vendor may implement a special device protocol within the general message envelope defined by the base protocol.

Participation in all three of the protocol levels requires understanding of the device level. The lower levels of this firmware are likely to be common to many devices. Higher levels of the firmware are expected to be more specific to the device and the application.

SOFTWARE ARCHITECTURE

Figure 4 depicts the host software structure of the ACCESS.bus. An ACCESS.bus peripheral requires software at both ends of the bus transaction for managing all levels of the peripheral interaction-the physical layer, base protocol, and application protocol. As well, the peripheral device requires software to support communication between the device microcontroller and the application-specific I/O transducer circuitry.

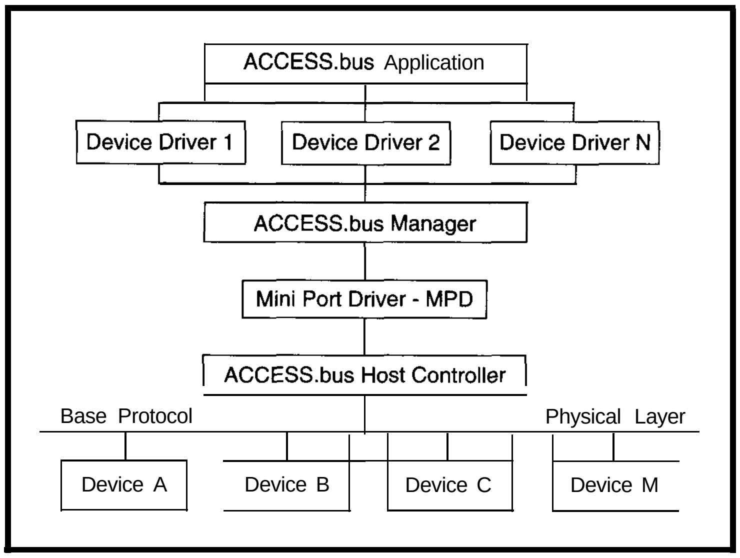


Figure 4—*The application software running on the host is shielded from the ACCESS.bus controller* by several layers of software.

The host-system operating software must provide interfaces so that application programs can access both ACCESS.bus devices and the ACCESS.bus itself. Since the lower levels of the interaction are common to diverse device types, they can be supported by the same or similar software modules.

The Mini Port Driver (MPD), pictured in Figure 5, is a communication software layer that separates the ACCESS. bus specific hardware from the ACCESS.bus manager.

The manager is a central software driver that controls the operation of all ACCESS.bus devices attached to the bus. It communicates with the Mini Port Driver on the one side and with the ACCESS.bus device drivers on the other.

The manager initializes and controls the ACCESS.bus, recognizing newly inserted or removed devices. It links device drivers and applications with specific ACCESS.bus devices, validates incoming messages, and serves as a bidirectional data switch routing messages to and from the appropriate device(s).

SOFTWARE DEVICE DRIVERS

The software device drivers serve as a bidirectional interface between application programs and a specific type of device or devices (mouse driver, keyboard driver, etc.]. Obviously, the appropriate driver depends on the device type. However, there may be further parameters that characterize the device and affect the choice of driver or specific arguments for a selected driver.

It is important to remember that the application program may also need to be informed of these device parameters. The device-capabilities-information feature of the ACCESS.bus protocol gives a measure of device independence in the selection of drivers and informs the host software of device characteristics.

APPLICATION LAYER

All application programs communicate with ACCESS.bus devices via an ACCESS.bus device driver or directly with the bus manager. Applications can use many devices of the same type (multiple mice, multiple keyboards).

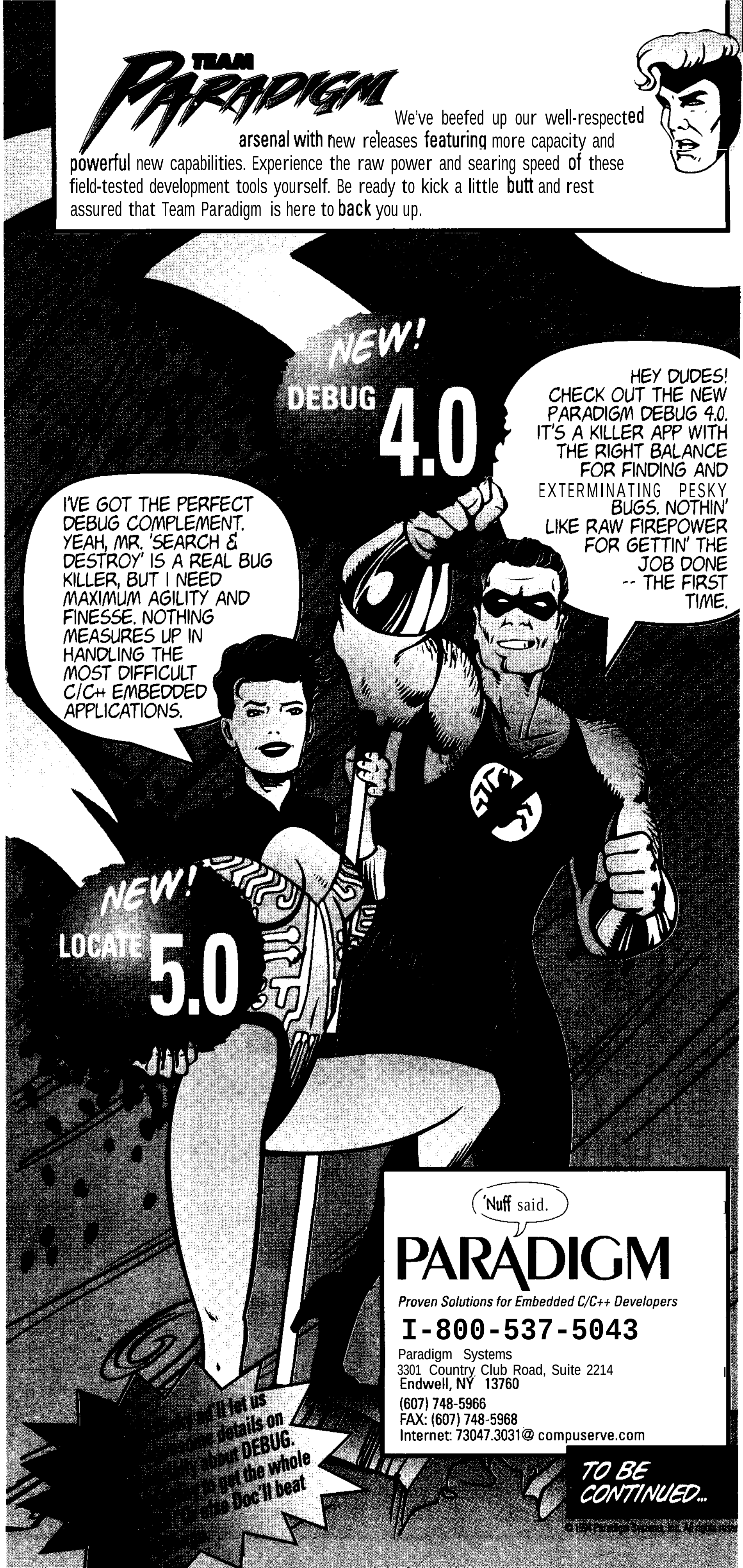
TRANSFERRING DATA TO AND FROM APPLICATIONS

For a device to be accessible to application programs running on the host, it must be connected to an appropriate software driver. Establishing this association is the last phase of configuration.

Device-capabilities information explicitly states a device's functional characteristics. Although these characteristics are implicit in the device-type designation contained in the ID string, there are sometimes variances among individual devices of a given type. For example, the capabilities information might include the national alphabet for a keyboard or resolution and units for a locator.

The capabilities information is contained in an ASCII-encoded text string stored on the device ROM. The base protocol defines a simple and compact grammar for building the capabilities string.

The semantics are carried by keywords. The base protocol defines


TEAM PARADIGM
NEW! DEBUG 4.0
HEY DUDES! CHECK OUT THE NEW PARADIGM DEBUG 4.0. IT'S A KILLER APP WITH THE RIGHT BALANCE FOR FINDING AND EXTERMINATING PESKY BUGS. NOTHIN' LIKE RAW FIREPOWER FOR GETTIN' THE JOB DONE -- THE FIRST TIME.
I'VE GOT THE PERFECT DEBUG COMPLEMENT. YEAH, MR. 'SEARCH & DESTROY' IS A REAL BUG KILLER, BUT I NEED MAXIMUM AGILITY AND FINESSE. NOTHING MEASURES UP IN HANDLING THE MOST DIFFICULT C/C++ EMBEDDED APPLICATIONS.
NEW! LOCATE 5.0
'Nuff said.
...'ll let us ... details on ... about DEBUG. ... to get the whole ... else Doc'll beat
TO BE CONTINUED...

We've beefed up our well-respected arsenal with new releases featuring more capacity and powerful new capabilities. Experience the raw power and searing speed of these field-tested development tools yourself. Be ready to kick a little butt and rest assured that Team Paradigm is here to back you up.

PARADIGM

Proven Solutions for Embedded C/C++ Developers

I-800-537-5043

Paradigm Systems
3301 Country Club Road, Suite 2214
Endwell, NY 13760
(607) 748-5966
FAX: (607) 748-5968
Internet: 73047.3031@ compuserve.com

© 1994 Paradigm Systems, Inc. All rights reserved keywords which apply to all device types. The application protocol defines the keywords specific to certain device types. To date, the application protocols define semantics for the capabilities information for generic keyboards, locators, and general text devices. The grammar allows for easy extension of the capabilities-information specification.

THE MOUSE DRIVER

The ACCESS.bus Multiple Mouse Driver (ABMSWIN.DLL) is a driver developed by CATC and offered as part of their Windows application development kit.

The driver connects to as many devices as specified in the [AbMsWin] section under the maxDev entry. The application connects to this driver by supplying the target queue (HWND) for mice movements and button state. Interrupts are converted to Windows messages and placed in the application-input queue. These messages are in raw format and cannot be used by the application.

To convert the raw message into a more useful format, the application returns the messages to the driver, which in turn processes the information and sends additional messages to the application message queue. The messages include information such as moves, button state, connect, disconnect, and so on. The processed messages are in the following format:

wParam:
  LOBYTE is the mouse ID
  HIBYTE is the button state
lParam:
  LOWORD is the x position in pixels
  HIWORD is the y position in pixels

There are certain restrictions that the programmer must plan for:

- The driver blocks information from the application once a raw message is sent. To continue to receive information, the application must return the raw message to the driver. This feature prevents overflow of the message queue.
- The driver is capable of handling one window at a time. Mouse bitmaps are displayed in the client area of the connected window.
- Application information is saved and restored when the mouse is in motion, but there is no protection to the mouse image. The application must explicitly turn mice off prior to screen update, and then turn them on again.

MOUSE DRIVER SERVICES

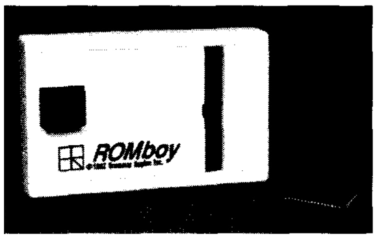
Upon interrupt, the application is notified with a raw message, "type **MICE-EVENT**" which is defined as WM_USER+128. The application must then send this information to the driverusingthe AbMouseMoveMouse service. The driver processes the raw information and finally replies with one of the messages listed in Table 2.

Specific service messages are also necessary for complete application compatibility. Many of the service messages and their meanings are included in Table 3.

'The BEST in ROM emulation technology:

Now 4 Mbit Available

ROMboy!



❑ **1 Mbit**
❑ **100ns**
❑ **Price $295**

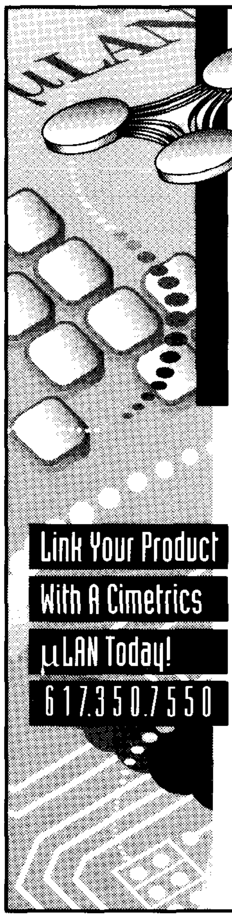
ROMboy includes a **10** day, no-risk money back guarantee!

Call Today — **800-776-6423**

Grammar Engine Inc.

921 Eastwind Dr., Suite 122
Westerville, OH 43081
614/899-7878
Fax 614/899-7888

#110



Cimetrics
TECHNOLOGY

Linking Microcontrollers. Breaking Boundaries.

Link Your Product With A Cimetrics µLAN Today!
617.350.7550

***The O-Bit* Solution**

The Cimetrics Technology 9-Bit Solution is a complete microcontroller network (µLAN) that supports the 805 1, 68HC11,80C186EB/EC, and many other popular processors. The O-Bit Solution takes full advantage of microprocessor modes built in to microcontollers. The 9-Bit Solution allows simple and inexpensive development of master/slave multidrop embedded controller networks.

- 8051, 68HC11,80C186EB/EC compatible
- A full range of other processors supported
- Up to 250 nodes
- 16 Bit CRC error checking with sequence numbers
- Complete source code included.

55 Temple Place • Boston, MA 02111-1300
Ph 617.350.7550 • Fx 617.350.7552

#111

THE KEYBOARD DRIVER

The ACCESS.bus Multiple-KeyboardDriver(ABKBWIN.DLL), which is included in CATC's application development kit, connects to as many devices as specified in the ma x De v entry of the [AbKbWi n] section. The application connects to this driver by supplying the target queue (HW N D) for keyboard events. Interrupts and the resulting messages are processed in a manner similar to that described for mouse drivers..

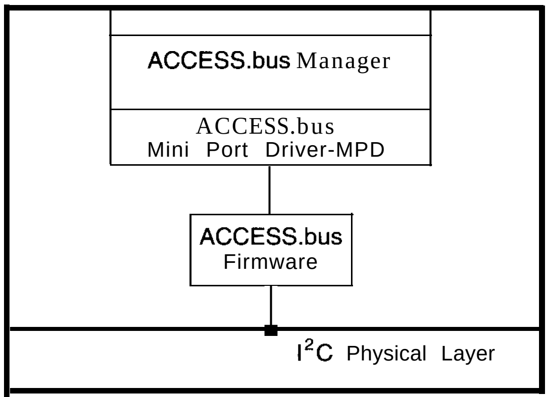


Figure 5-The *ACCESS.bus* Mini *Port Driver is the ACCESS.bus* firmware-specific interface. The *bus* manager *layer* remains *constant* despite *host* firmware adaptions

CHINESE CHECKERS

Chinese checkers is a native Windows game application. Similar to the board game, the application can be played by two to six players. Ten marbles are "placed" in the player's section of a six-pointed star. Each group of marbles is a color unique from the rest of the groups and the player of a group is represented by a same-color pointing device (i.e., only the player with the yellow cursor can move the yellow marbles). The number of players can be changed, and one or more groups can be assigned to the computer using the Game Setup command. (Previous to the ACCESS .bus implementation, Chinese checkers was a single-cursor game in which one cursor could control and move any and all marbles.)

The setup window is simple and self explanatory. A group of marbles can be added or removed, assigned to an individual or the computer, allotted a turn sequence, and changed color. The addition or deletion of a mouse (or another locating device) automatically adds or deletes a cursor and group of marbles for the associated player.

Each one of the active mice moves a colored cursor while it is inside the game window. You can turn the colored cursor into a Windows system cursor by moving it out of the game window from the top side of the window (the menu side). Once the cursor is out of the game window, it becomes a standard Windows mouse cursor. When you move it back into the application window, it automatically reverts to a colored game cursor.

BLACKJACK

The ACCESS.bus version of blackjack was created specifically to show the simultaneous, multiplayer capacity of the ACCESS.bus. The game window gives a top-down view of a blackjack table in a casino with stations for six players, each having their own betting chips and cursors.

Unlike Chinese checkers, which operates in serial mode, each player playing in turn, blackjack is a parallel game-participants play simultaneously. Each participant can concurrently place a wager, buy more chips from the cashier, and indicate a "hit" or "stand" from the dealer.

All setup and operational characteristics associated with the Chinese checkers apply to blackjack. The game includes on-line help, configuration of mice and colors, rules of the game, changing active mouse buttons, and alteration of card design.

DESIGNING A GAME

Not only is the Chinese checkers an adaptation of a classic board game, it is an example of converting an existing game into an ACCESS. bus game. When David Williams originally created the Chinese checkers computer game, he intended it to be a single-input-device game. Whether play was against the computer or other humans, only one interface device could be used by participants. As one move completed, the next person took over the mouse and made his or her move.

The opportunity for each participant to "own" his or her pointing device (mouse, trackball, or joystick) and the ability of the software to accept and accommodate actions of several devices brings a whole new look and feel to the game. No longer do participants need to clamor for their turn at the mouse. Players can even use the pointing device of their choice.

MICROCODE DEVELOPMENT

The microcontroller in the device provides the intelligence for managing the device's participation in all levels of the ACCESS.bus protocol. Use of components with I^{2}C interface functionality simplifies development of the lowest level of the interface and protocol. Since the protocol concerns only the bus communication methods common to all sorts of peripheral devices, they may be implemented by reusing software previously developed for some of these components. Devices conforming to the semantics of the predefined standard application protocols may also benefit from the availability of some off-the-shelf code at the top level.

```
MICE_LDOWN        (MICE_EVENT+1)    /* left button pressed */
MICE_LDBLCLK      (MICE_EVENT+2)    /* left button dblclk */
MICE_LUP          (MICE_EVENT+3)    /* left button released */
MICE_MDOWN        (MICE_EVENT+4)    /* middle button pressed */
MICE_MDBLCLK      (MICE_EVENT+5)    /* middle button dblclk */
MICE_MUP          (MICE_EVENT+6)    /* middle button releases */
MICE_RDOWN        (MICE_EVENT+7)    /* right button pressed */
MICE_RDBLCLK      (MICE_EVENT+8)    /* right button dblclk */
MICE_RUP          (MICE_EVENT+9)    /* right button released */
MICE-MOVE         (MICE_EVENT+10)   /* mouse move */
MICE-CONNECT      (MICE_EVENT+11)   /* mouse connected */
MICE_DISCONNECT   (MICE_EVENT+12)   /* mouse disconnected */
```

Table *2-Upon a* mouse interrupt, the application is notified *of a mouse event with* a *raw message. Once that message is passed to the driver, the driver responds with one of these messages.*

```
void far pascal AbMouseDriverOn(HWND hwndApp, HDC hdc)
```
Invoked at **initialization** to indicate the target window for mice events. Zero is returned for success, and a negative number if the driver is already busy with another application.

```
void far pascal AbMouseDriverOff(void)
```
Disconnects the window from the driver.

```
int far pascal AbMouseActiveDevs(void)
```
Returns the number of connected devices.

```
int far pascal AbMouseGetPos(int id, int far *btn, int far *x, int far *y)
```
Returns the mouse position and button state. If the mouse is disconnected, then the service returns zero. If the mouse is connected, then the service returns a nonzero value. *btn*, *x*, and y are set with the button state and the mouse position. Buttons are represented by bits: D0 = left button, D1 = right button, and D2 = middle button.

```
void far pascal AbMouseXorMouse(HDC hdc, int id)
```
XORs the mouse bitmap at the current position.

```
void far pascal AbMouseMoveMouse(HDC hdc, WPARAM wParam, LPARAM event)
```
Must be called for each raw message from the driver. The **wParam** and **lParam** parameters should be identical to the values passed via the `MICE_EVENT` message.

```
void far pascal AbMouseClientArea(RECT far *rect)
```
Informs the driver of the client area size. The mice images are restricted to the client area of the connected window. Invoke this service upon the `WM_SIZE` message.

```
void far pascal AbMouseSetPos(int id, int x, int y)
```
Sets a particular mouse position. At initialization, the position is set to x = 0 and y = 0. The application may use this service to center the mouse in the client area. If a device was connected while the application was running, the driver assigns the position as the center of the client area.

```
int far pascal AbMouseGetSysMouse(void)
```
Instructs the system mouse driver (`ABMOUSE.DLL`) to release one of its devices and it releases the last active device. The multiple device driver (`ABMSWIN.DLL`) then attempts to claim that mouse. Applications can use this routine to convert a system mouse into an application mouse. The returned value is negative if no slots are available in `ABMSWIN.DLL driver`. In this case, you can increase the number of the devices in `maxDev` of the `[AbMsWin]` section of your ACCBUS.INI. However, the normal return is zero which the service returns even if no mouse is found (this occurs when no system mouse is available). An application may test for this condition using the `AbMouseActiveDevs()` services.

```
int far pascal AbMouseRelSysMouse(int id)
```
Causes ABMSWIN.DLL to release a device and ABMOUSE.DLL to claim it. Applications may use this service to convert application mice to system mice. Zero is returned for success, and a negative value is returned if the device identification is invalid or not connected.

```
void far pascal AbMouseSetBitmap(HDC hdc, int id, HBITMAP hBmp)
```
Sets the mouse image. The mouse image is a 16 ¥ 16 bitmap that was loaded using Windows' `LoadBitmap()`. Note that the driver will not delete the bitmap upon termination. It is the responsibility of the application to call Windows' `DeleteObject()`. The returned value is 0 if the function is successful and -1 if the device identification is invalid.

```
int far pascal AbMouseRestoreBitmap(HDC hdc, int id)
```
Restores the default mouse image. Zero is returned for success, and -1 for an invalid device identification.

```
int far pascal AbMouseShortCapabilities(int id. DevProt far *devProt)
```
Fills the given structure **devProt** with the device's prot, type, and model strings (see ACCESS.bus specification for the meaning of these parameters). Zero is returned for success and -1, for invalid device identification. This service may be invoked to determine the specific model of a device

Table 3-*Under the ACCESS.bus protocol, service messages* are defined *to* handle *all* aspects of *the* operation of attached *devices. This is just a sampling of some of the messages available to do mouse support.*

HOST SOFTWARE DEVELOPMENT

Vendors of host systems supporting ACCESS.bus must supply drivers and other operating systems modules necessary for access to the ACCESS.bus port both for application program clients and for other system software such as the interactive I/O handlers of the window system. Here again, many ABIG member companies offer a wide variety of products to support ACCESS.bus functionality, including a manager, mini port and device drivers for various operating systems, software development kits, and source-code modules.

OTHER GAMES

What fun is a game that cannot be played with other people?

Chinese checkers and blackjack are small examples of the feature characteristics that ACCESS.bus offers the game and entertainment industry. Skill games especially, like chess or checkers, can be played by humans in front of a single video display without the need to either share an input device or be relegated to competition with a computer.

Other classic casino games—poker, roulette, and craps-are natural candidates for ACCESS.bus support as these games typically have several participants involved at one time. As in blackjack, each player needs individual control over the amounts and placement of their betting chips. Simultaneous placement of wagers, cashier interaction, and execution of play would be easy and effective in the ACCESS.bus environment.

Chinese checkers offers insight into other board games such as Monopoly, Life, or Clue. Although some of these are available for computer, they don't offer player interactivity. The games would gain new life with control devices for each player. Players would gain more control over piece movement and game interaction.

The largest game-growth sector, however, is the action-game market. Imagine SuperMario with two or more plumbers or a multiplayer Wolfenstein. The interactive opportunity of these types of games would open new markets to the manufacturers of both the games and peripherals.

With ACCESS.bus, the software company would no longer be confined to building games with the idea that only one mouse, joystick, keyboard, or trackball can be attached to the system. In fact, the game company could bundle new peripherals with the game at attractive prices and be assured that the new devices would be compatible with the system.

BEYOND GAMES

ACCESS.bus is an evolution of peripheral connection technology for the personal computer industry. The ability to add or delete devices from a system is only the first step. Intelligent components of the entire systems may soon have plug-and-play support characteristics. The ability to plug in a memory module or coprocessor, turn on the system, and have the new components recognized with the power-on self test is a near reality.

The Video Exchange Standards Association (VESA) has adopted ACCESS.bus for use in display devices. The display data channel (DDC) specification incorporates ACCESS.bus as a control interface for autoconfiguration of a compliant display device to a host computer system. The DDC also facilitates control of the display adjustments-refresh rate, vertical and horizontal scan, resolutions, color temperatures, screen position, brightness, contrast, and other characteristics-and lets users create software files that load user-defined or preset configurations to the monitor. Video adapter vendors may offer ACCESS. bus host interfaces in their products or pass the ACCESS.bus information to another device such as an add-on card, adapter, or system motherboard.

Eventually, it may be possible to hide the computer under or behind the desk. Peripherals will be connected to an ACCESS.bus port on the front of the monitor. The need to run several cables out the back of the system onto the desk will be eliminated.

OTHER APPLICATIONS

While games are attractive and splashy, many other applications appear to benefit from the use of ACCESS.bus. At this time, the most pressing need is educational software.

In the classroom, where computer literacy is as much a part of the course curriculum as math and language, the opportunity to place a computer system at every student's desk is not feasible due to cost, configuration, and connectivity. Even in today's market of low-cost, preconfigured, off-the-shelf systems, the typical Windows-compatible personal computer is about $1,000. If you multiply that by 500–1000 students per school, the overall cost could approach $1,000,000 per school! Add in the cost of maintenance and support-the computing classroom becomes untenable.

Conversely, educational software programs that serve an entire class from one computer system makes the possibility of the computing classroom much more feasible. A keyboard with a small LCD panel and an appropriate software application would let the teacher engage the class in a host of on-line activities.

Computer systems manufacturers are developing these hardware solutions today. What's needed the most is a software base!

THE FUTURE OF ACCESS.BUS

ACCESS.bus is already a proven, stable technology with a strong base in embedded control and vertical applications. Even without the opportunities of the game and educational markets, the possibility of attaching multiple peripherals to the power of the latest wave of high-speed CPUs is attractive. The system board and CPU certainly have enough bandwidth and power to support multiple concurrent tasks.

Why limit multitasking to four COM ports and a couple of parallel ports? ACCESS.bus not only provides system connectivity, but may in fact obviate COM ports and conventional peripheral connectivity arrangements.

The fact that ACCESS.bus is an open industry standard providing an easy and effective connection to multiple devices ensures its use and growth in scope and appeal.

In fact, now is a good time to initiate a ride on the ACCESS.bus wave since the field is not crowded. The hardware cost benefits are such that new software and applications will arguably achieve greater profit margins and quicker market acceptance than new nonACCESS.bus applications.

David Rodgers is the director of sales for Computer Access Technology. After starting his career as an engineering technician in 1983, he has worked as a national sales and marketing director for several large companies.

Peretz Tzarnotzky is vice-president of engineering at Computer Access Technology and has 22 years experience as an electronics and systems engineer.

Either author can be reached at catc@netcom.com.

CONTACTS

For full details about the Software Creators Contest:

ACCESS.bus Industry Group
370 Altair Way, Ste. 215
Sunnyvale, CA 94086
(408) 991-3517
Fax: (408) 991-3773
ABIG@netcom.com

For complete systems solutions including development tools for hardware, software, and firmware.

Computer Access Technology Corp.
3375 Scott Blvd., Ste. 410
Santa Clara, CA 95054
(408) 727-6600
Fax: (408) 727-6622
catc@netcom.com

For educational software:

Emerald City Education
Lowry Building, Ste. 103
700 North Egan Rd.
Madison, SD 57042
(605) 256-5681

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful

Make the Most of Your DSP-based Sound Card

FEATURE ARTICLE

Bob Fine

If you have an add-on sound board for your PC, you probably simply plugged it in and forgot about it. Now it's time to dig into the innards of that board. Find out how to write your own signal processing code for it.

Have you ever used a sound card with your personal computer only to grow tired of using the standard applications software? Have you ever wished you could get into the "guts" of the sound card, to go beyond the standard voice processing and sound generation, to make it do what you wanted! Or, have you wished there were a convenient platform for developing signal-processing software and controlling it with your PC, thereby giving your PC access to real-world signals?

In this article, I will show you how to access the DSP resource on the newest sound cards, develop your own algorithm code, download it to the RAM on the card, and control the software via a Windows program.

For the context of this article, I'll assume you have some experience writing code for a DSP either in assembly language or C. It would also be useful to become familiar with Windows programming (if you aren't already) before following the programming tips. In the Windows environment, a good user interface can provide a powerful, yet easy-to-use, mechanism for interacting with the DSP-based sound card. This article will show a simplified Windows program.

A BIT OF HISTORY

As the popularity of the personal computer grew in the 1980s so did the number of programmers writing entertainment software. This software evolved from basic games to more sophisticated programs that took advantage of animation and color. Sound was added to these games by using the computer's built-in speaker to produce "blips" and "beeps" sounds. There were no standards; everyone just did their own thing.

The popularity of video game devices that connected to a standard television spawned the desire for computer-based games that could produce sound effects and music. This led to the introduction of the first ISA-bus sound card which featured sound capabilities beyond those available through the small speaker in the PC.

Creative Labs introduced one of the first PC audio cards, the Sound Blaster, and soon many software developers began working on new games and a number of musical and MIDI applications (the applications took advantage of the Yamaha 0PL2 synthesizer chip on the card). Decent sound effects and music were possible for the first time. The Sound-Blaster standard was born out of the large library of software titles that was available.

In 1990, a number of hardware and software companies formed the Multimedia PC Marketing Council (MPC). Part of the MPC specification for minimum PC hardware requirements included a sound board with 8-bit MIDI sampling in and out (11.025 kHz and 22.05 kHz sampling rates). This formed the beginning of an industry movement to offer developers access to a standard platform and meant that software written for one MPC-compatible PC card could also run on any other. However, there still existed some incompatibility, and the functions of the PC sound cards were fairly fixed.

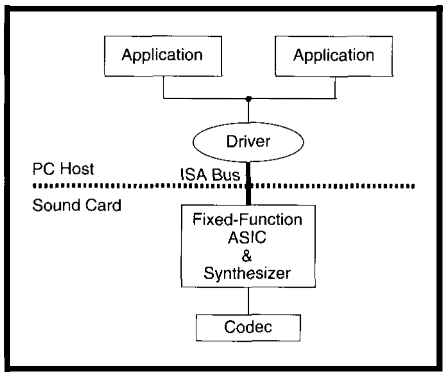


Figure 1—First-generation sound cards had most of their funtionality fixed in the hardware and had no upgrade path.

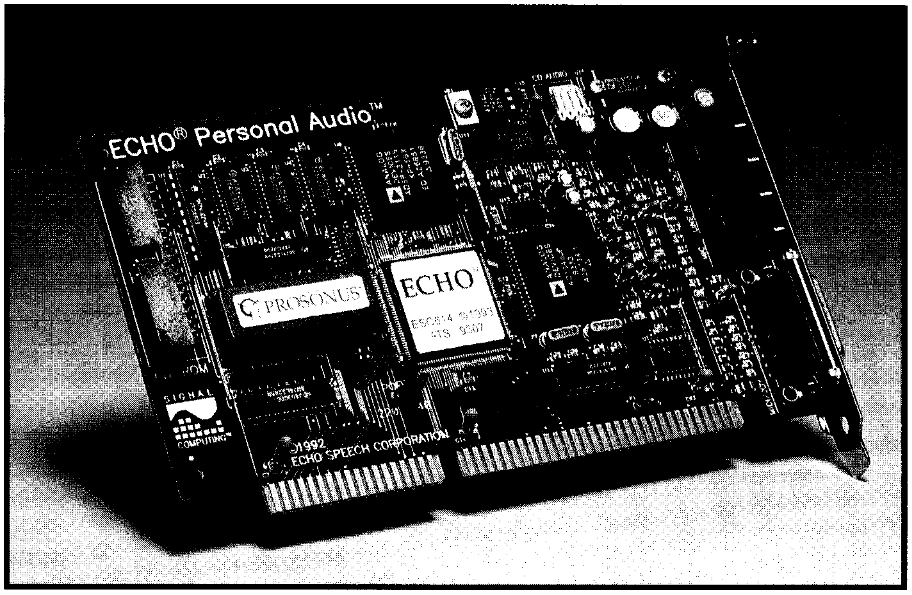


Photo I--The *Personal Sound System card, part of Analog Devices' ADMK-610 manufacturing kit, offers high-end audio, voice, and speech for personal computers.*

Industry acceptance of the Windows operating system led to the eventual release of Windows multimedia extensions, which provided further capabilities for processing sound and MIDI files and more programming tools for multimedia capabilities. This movement helped overcome some of the compatibility obstacles, but didn't do much to overcome the limited, fixed-function capabilities of first-generation sound cards.

FIRST- AND SECOND-GENERATION SOUND CARDS

A typical first-generation sound card consisted of an ADC and DAC, ASIC, FM-synthesis chip, jumpers, and connectors. The functions of the card were fixed in hardware. The programmer merely selected one of the fixed functions by writing commands to control registers in the ASIC.

Figure 1 shows the software partitioning of the first-generation sound card. The host applications accessed the sound-card functions through a driver. The driver passed the proper commands to the ASIC registers where fixed functions were controlled. Ease of programming and existence of a standard ushered in the considerable amount of game software developed for these cards and their popularity. The card's functions, however, were limited and fixed by the hardware. The hardware limitations forced consumers to buy a new sound card to take advantage of newly available features.

Second-generation sound cards use a programmable Digital Signal Processor (DSP) which flexibly processes the signal fed to the card. Software mimics the fixed functions found in first-generation cards and provides functions never before implemented. Software upgrades replace hardware upgrades-only the programmer's imagination and the speed of the DSP cause a limitation. For the first time, you can program the DSP on the sound card for whatever function you desire.

Photo 1 shows the Echo Personal Sound System, a second-generation, DSP-based sound card. Other similar sound cards include:

- Cardinal Digital Sound Pro 16
- Orchid Soundwave 32
- Wearnes Beethoven ADSP16
- Western Digital Paradise 16-DSP
- Adaptec AME-157x

Every major DSP manufacturer offers a chipset solution with a software development toolkit which can be used to build a DSP-based sound card. Analog Devices, AT&T, IBM, Motorola, and Texas Instruments are involved in the specification of an industry-standard DSP Application Programming Interface (API) which enables Windows programmers to write generic applications software for just about any sound card.

The generic design for a DSP-based sound card consists of a DSP, ISA bus interface, and analog I/O circuitry. The common architecture used by sound card manufacturers such as those listed above is based on an Analog Devices ADSP-2115 DSP shown in Figure 2. The circuit has three basic components: a DSP with associated memory, a 16-bit analog interface with programmable sampling rate, and an ISA-bus-interface ASIC.

I won't go into the details of the hardware since you can easily buy a sound card. Instead, I will focus on the software development process and some of the underlying hardware which deals with the communication over the ISA bus.

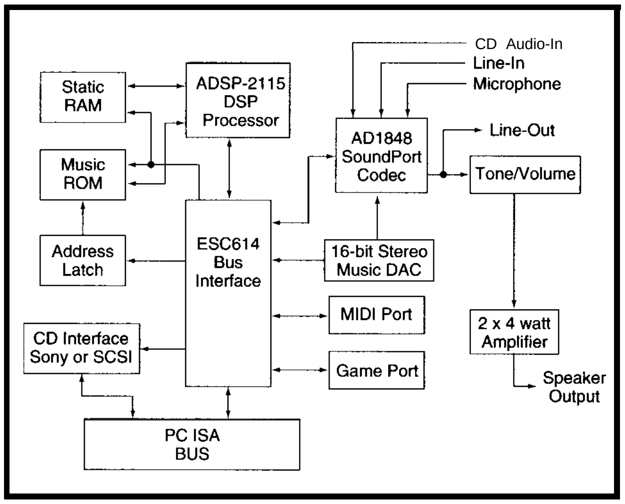


Figure 2—*The* latest *round of sound cards are based on a DSP, so offer* much improved functionality *and* flexibility. *The* operation of the board *can be* completely changed *simply by loading new software onto* if.

GETTING AT THE DSP

Contrary to popular belief, second-generation DSP-based sound cards are RAM-based and are programmable. I will guide you through the step-by-step creation of a DSP program which runs the DSP of the sound card and of a Windows program which controls the DSP-program execution. To re-create or modify the code, you'll need a Windows C compiler (I used Borland C 4.0) and a software development kit from a DSP manufacturer (I used Analog Devices' kit).

Once you write your algorithm software, you still need to get the DSP code downloaded to the processor. Then, you need to control the execution of the software interactively via some host software. At this step, you will need the help of some software utilities or tools.

Figure 3 illustrates the communication between the host PC's software and the DSP software. The Windows application makes API (or function) calls Behind the scenes, the device driver passes the information along the ISA bus to registers inside the ASIC on the sound card. These registers are accessed by the DSP under control of DSP code. The key software elements controlling this communication are the DSP manager and the DSP shell.

If you could look inside the DSP API and the driver software, you would see that the ASIC controls all the information passed between the host and the DSP. The ASIC contains a number of data and control registers which are mapped into the PC I/O space and the DSP memory space. Under program control, both the host PC and the DSP can read and write the ASIC's registers. The ASIC also has some additional hardware control lines that can signal the host PC and reset and interrupt the DSP. Figure 4 provides a more detailed diagram of the connections between the ISA bus and ASIC, and between the ASIC and ADSP-2115.

To load DSP code into DSP's program memory, the host first loads an ASIC register with the first instruction byte for the DSP. The host then writes a control word to the ASIC which, in turn, resets the DSP. The DSP begins its boot sequence, reading the ASIC's data register. The ASIC detects the DSP read operation and suspends the DSP by requesting its bus. At this point, the DSP has read one byte and is essentially frozen.

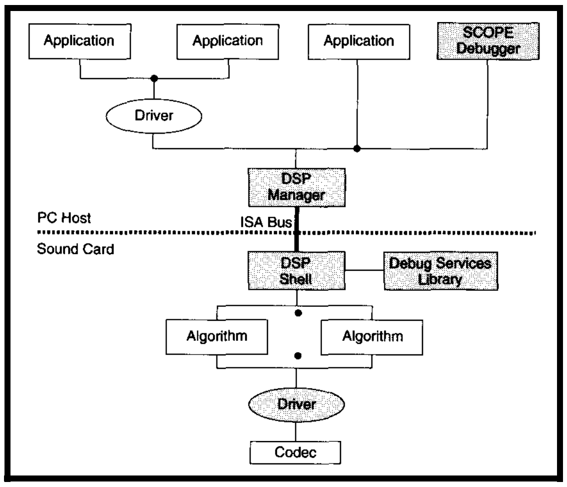


Figure 3—*Support for code development tools is located not only on the host PC but also on the sound board itself.*

The ASIC detects the DSP going into its bus-request mode and sends a message back to the host requesting another data word. While the DSP is suspended, the host writes another instruction byte and then a control word to the ASIC. The DSP is taken out of bus request and is allowed to read the next byte. The DSP read operation is detected by the ASIC, and the DSP's bus is requested again, suspending the DSP.

This process of writing data words to the ASIC, having the ASIC momentarily request and release the bus of the DSP, and letting the DSP read another instruction word is repeated until the DSP's program memory is filled with code.

Once the complete DSP program is loaded, program execution starts. This sequence of events is illustrated in the flowchart in Figure 5. The host can communicate to the DSP by writing data to the ASIC register and having the DSP read this register.

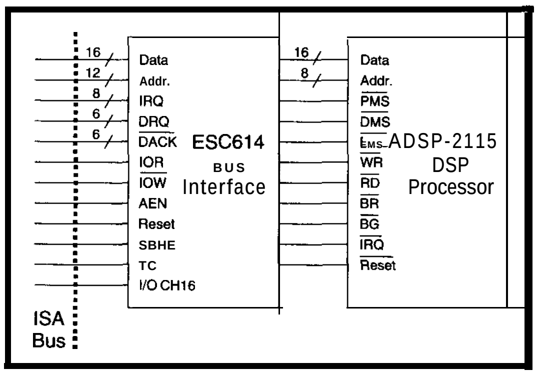


Figure 4-*On DSP-based sound boards, an ASIC handles all the interface details between the DSP and the ISA bus.*

Communication from the DSP to the host is just the reverse.

This process may sound very complex and, in fact, can be complicated for the programmer. But, this is why software development tools are available. Such tools simplify things by supplying a DSP manager and shell.

The DSP manager is a program (actually, a library of functions) which runs under Windows and provides overall control of the DSP-based subsystem. The DSP manager lets you load and unload algorithms on the DSP and provides a communication system between the host and the DSP. The underlying communication is done for you, so you don't have to worry about explicitly writing to the ASIC.

The manager is implemented as a Windows Dynamically Linked Library (DLL) and also contains a driver for specific hardware platforms. The DSP manager coordinates the resources of the DSP subsystem and is not dependent on the specifics of hardware implementations.

In fact, the DSP manager will adhere to the proposed API standards being set by the Interactive Multimedia Association (IMA). Therefore, you should be able to make DSP-API calls in your Windows program and have the function work with different manufacturers' DSPs.

The DSP shell consists of a set of the ADSP-2100 family functions which are written in assembly language and can be called using C. The functions are combined in a library. The shell provides services for a DSP algorithm such as initialization of DSP hardware, real-time interrupt servicing, algorithm and application communication, audio input and output device driver control, and miscellaneous system and debug services.

Again, you are freed from worry about the details.

SENDING AND RECEIVING MESSAGES

The operation of the host application and the DSP program on the sound card is based on a messaging scheme between the two programs. The DSP manager and shell each use a messaging scheme similar to that of Windows. If you have experience in programming for Windows, this messaging scheme should be easily followed.

Message values are usually checked in a case statement in your program and appropriate action is taken for each message. Messages are sent and received via API function calls. Function calls are made by the host application to the DSP manager and are also made by the manager to the host application. The same relationship holds true for the DSP algorithm and shell.

The arguments of the function describe the message. These functions are used to carry out specific tasks such as loading a program onto the DSP or filling a data buffer with samples from the A/D converter. The messages are used to inform the various program modules of the status of operations. Messages can also be associated with a buffer for functions that will either provide or require data. For instance, a message could be used to let you know that the buffer you requested to be filled with data from the ADC is full.

The generic message consists of a DSP-algorithm identification number (or handle), a message identification number, and the body (data words) of the message. The DSP program will typically request a message from the host application which sits in a queue until a message is ready. The host application will do the same. These function calls and messages will become clearer when I show code examples.

Now we can look at developing a DSP program that can be linked with the DSP shell of the software development kit. For this example, I have selected a FIR filter with a Windows interface that lets you control the operation of the filter by pointing and clicking the mouse. You can use this technique to create just about any signal-processing function.

For simplicity, the example program does not perform any error checking. Most function calls return an error code if the function was not successful. To create a robust program, you should always check for any returned error codes and take the appropriate action.

THE CODE DEVELOPMENT PROCESS

There are actually three layers of DSP code that will eventually be linked together to form an executable program which can be downloaded to the DSP. At the lowest level are the DSP algorithm subroutines. These routines are called from a main module which communicates to the DSP shell. The shell handles all the interrupt servicing, I/O, and communications to the host PC via the ASIC.

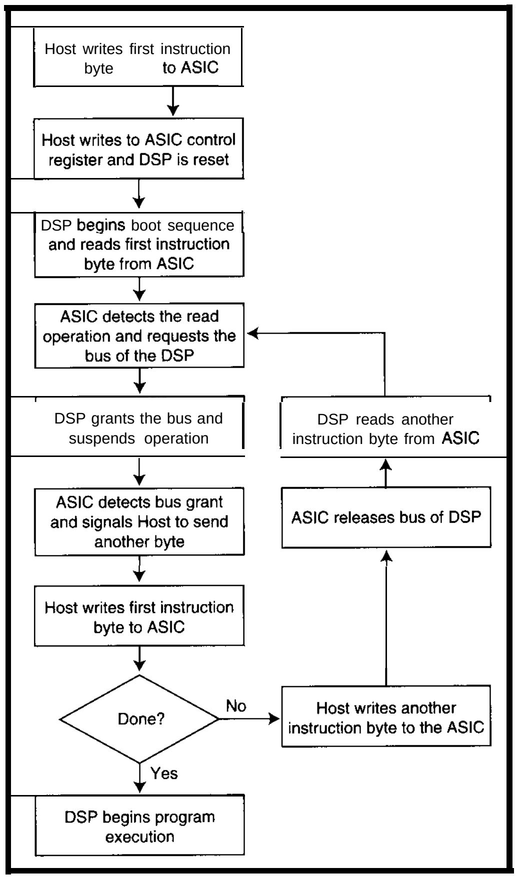

Host writes first instruction byte to ASIC
Host writes to ASIC control register and DSP is reset
DSP begins boot sequence and reads first instruction byte from ASIC
ASIC detects the read operation and requests the bus of the DSP
DSP grants the bus and suspends operation
DSP reads another instruction byte from ASIC
ASIC detects bus grant and signals Host to send another byte
ASIC releases bus of DSP
Host writes first instruction byte to ASIC
Done?
No
Yes
Host writes another instruction byte to the ASIC
DSP begins program execution

Figure 5—*The procedure for downloading code to the sound card DSP involves some nontrivial code, but isn't particularly hard to understand.*

WRITING DSP CODE

If you have experience writing DSP code for an embedded system, you will note some differences in the process of writing DSP code for a sound card installed in a PC. Programmers of embedded DSP systems code their DSP algorithm, then take the assembled-and-linked DSP code and either program it into an EPROM (plugged into a socket on the board) or use a communication connection (usually via the RS-232 port) to download the code to the DSP.

Even though the algorithm code for the embedded system looks similar to that for the DSP on the sound card, other software components are also required. The technique of getting the DSP code onto the DSP is also different. I will explore these issues later on in this article.

The DSP assembly code for a FIR filter is shown in Listing 1. The code for a DSP-based sound card is basically the same as for an embedded system. The key differences lie in the manner in which the executable is created and the way in which the program is downloaded to the processor.

As you can see in Listing 1, data values and filter coefficients are fetched from memory. These values are multiplied together and the product is accumulated with previous products. This process repeats N times, where N is the number of taps of the FIR filter.

For this program to run on the sound card and process information in real time, several things have to happen. First, the program needs to be loaded from the memory (or disk) of the PC into the local memory of the DSP. Filter coefficients also need to be loaded. The processor must be initialized along with the sampling characteristics of the A/D and D/A converters (i.e., sampling rate). After a buffer is filled with data from the ADC, the data is processed by the filter routine and a result buffer is filled. Data values from the result buffer are sent to the DAC.

From a user perspective, the filtering is performed in real time and no data is lost. However, there is a small delay resulting from the storage of data samples in the buffer. Tradeoffs between the size of the data buffers and the amount of interrupt-servicing overhead can be made by the programmer. The smaller the delay and the less data stored, the more frequently the DSP shell must vector to the filter routine. A larger buffer results in less vectoring, but a bigger delay.

A data-memory delay-line buffer called da t . a is declared. Two coefficient buffers are also declared and initialized with values from coefficient files. For more flexibility, you could also send the coefficient values to the subroutine interactively. In the sample program, a decision is made based on a mode value sent from the host application. The program then implements either a high- or a low-pass filter depending on the value of the mode.

Two functions, Re g S a v e and RegRestore, are referenced in the filter program. These functions perform a save-and-restore of all processor registers, so DSP shell data is not lost. This is important since many other ADSP-21xx routines may be in the system. By not paying attention to the housekeeping of registers and data values, both your program and others may not work as expected.

The filter code shown contains two subroutines. One initializes the delay line buffer (i n i t F i 1 t e r_) and the second performs the filter calculations (sFilter_). When the filter routine is called by the main DSP program, two parameters are passed to the filter routine. The input data value from the A/D converter is passed in the AR register and the filter mode value is passed in the AY1 register. The result of the filter is left in the AR register to be passed back to the main program.

CREATING A MAIN PROGRAM

As mentioned, a main program is used to call the DSP subroutines and to communicate with the host via the DSP shell. Since the main program is not performing time-critical calculations but acting as more of a controlling function, the main module can be written in C.

Writing a main DSP program in C that works with the DSP shell differs from writing regular C programs. Normally, a C program makes calls to the operating system or runtime libraries; the program controls the system. In the DSP shell system, the shell controls and calls your program when an event occurs. Listing 2 offers a sample main program.

Each algorithm has a d s pA 1 g - Main0 function. The dspAlgMain function is the entry point to your algorithm's main module, much like the ma i n function in C. The DSP shell calls dspAl gMai n with messages. The first message that an algorithm receives is an initialization message. After receiving this message, your algorithm may call DSP shell functions to enable I/O devices, host communications, or system service messages. Your d s pAl gMa i n function must return to the DSP shell after processing a message.

Listing 1—*This program implements a single-precision, direct-form, FIR-filter structure. The coefficients in this program are for a filter with a length of 81 taps. The filter can be programmed to select coefficients for a low or high-pass filter. The filters are designed via the Parks-McClellan algorithm and the coefficients are stored in the files HP_COEFF.DAT and LP_COEFF. DAT.*

```
.MODULE/RAM      FIR_LPF;

#define          taps 81
.VAR/CIRC        data[taps];
.VAR/DM/RAM      i0_pnt;
.VAR/PM/CIRC     lp_coeff[taps];
.VAR/PM/CIRC     hp_coeff[taps];
.INIT            lp_coeff: <lp_coeff.dat>;
.INIT            hp_coeff: <hp_coeff.dat>;

.external   RegSave;
.external   RegRestore;
.ENTRY      initFilter_;
.ENTRY      sFilter_;

initFilt
            call RegSave:        save all regs except AR and AY1}
            I0=^data;            start addr of delay line buffet-1
            M0=1;                use address modify of 1}
            L0=taps;             circular buffer size}
            CNTR=taps;
            DO zero UNTIL CE:
zero:           DM(I0,M0)=0;     clear the filter delay line buffer1
            dm(i0_pnt) = I0:     save address pointer in memory}
            call RegRestore:      restore registers }
            RTS;

sFilter_
            DIS  M-MODE;         perform two's complement math}
            call RegSave;        save all regs except AR and AY1}
            I0= dm(i0_pnt);      load addr reg with buffer pointer)
            M0=1:                use address modify of 1}
            L0=taps;             circular buffer size}
            M4=1;                use addr modify of 1 for coeff ptr}
            L4=taps;             coeff buffer size}
            DM(I0,M0)= AR:       store sample in data buffet-1
            I4= ^lp_coeff;       addr of coeff buffer1
            AR=PASS AY1;         check mode, hi pass or lo pass}
            if EQ jump not_hp; {if mode 0, use lo pass filter coeff1
            i4=^hp_coeff;      {if mode 1, use hi pass coeff1
not_hp:     CNTR=taps-1;
            MR=0, MX0=DM(I0,M0), MY0=PM(I4,M4); {1st coeff, data}
            DO conv UNTIL CE:
conv:           MR=MR+MX0*MY0(SS) MX0=DM(I0,M0), MY0=PM(I4,M4)
            MR=MR+MX0*MY0(RND);
            IF MV SAT MR;
            AR = MR1;          {pass result in AR register)
            dm(i0_pnt) = I0;
            call RegRestore;   {restore registers)
            ENA M_MODE;
            RTS;
.ENDMOD;
```

The DSP main program included in Listing 2 first defines the messages it will send to or receive from the host. I have created five messages that I would like to use with my Windows interface. I want to be able to turn the code on or off, switch between a filtered or unfiltered signal, and select between a high- or low-pass filter. A message structure is defined along with some message and I/O buffers. Two buffers of each type are declared, so that while the host is working with one (filling or emptying it), the DSP code can be working with the other. The host and the DSP exchange the buffers to transfer information. This way, it is guaranteed that each program, both host and DSP, has access to a buffer.

a)

Functions	Activity
dspGetMsg	Gives a buffer to the driver to use when a message comes in
dspSendMsg	Sends a message to the host
dspGetIo	Gives a buffer to the device driver to fill
dspSendIo	Sends a block of data to the I/O device
dspGetIoDevConf	Returns the configuration of an I/O device
dspSetIoDevConf	Sets the configuration of an I/O device
dspStartIo	Starts an I/O device driver
dspStopIo	Stops an I/O device driver
dspGetData	Gives a buffer to the driver to use when data comes in
dspSendData	Sends a block of data to the host

b)

Functions	Activity
dspLoadAlg	Loads and starts an algorithm on the DSP
dspGetMsg	Gets a message from the DSP algorithm
dspSendMsg	Sends a message to the DSP algorithm
dspGetData	Provides a buffer for data from the DSP algorithm
dspSendData	Sends a block of data to the DSP algorithm
dspReset	Resets the DSP device and returns it to a known state
dspUnloadAlg	Terminates and unloads the algorithm on the DSP device

Table 1—*The DSP shell functions (a) run on the sound board's DSP chip while the DSP manager functions (b) run on the host PC. Together they form a powerful interface.*

The dspAlgMain routine accepts four calling parameters from the DSP shell: a message ID and three related data words. A case statement is used to test the passed message ID when the DSP shell calls dspAlgMain. The DSPCB_INIT message is sent to tell dspAlgMain to perform its initialization. Here, the message queue is started by requesting two messages with the dspGetMsg function. When the DSP shell has information for the algorithm, it fills and returns these buffers. The initialization routine also sets the operating configuration for the code (ADC and DAC) with the dspSetIoDevConf function.

Other messages are checked in the case statements and are used to confirm that a buffer has been filled or emptied by the I/O device driver or that a message has been sent or

▶ Good Stuff ◀

Bar Code Sensor
Battery Controllers
Clock/Calendars
Digital Power Drivers
DTMF & Phone Interfaces
Firmware Furnace Widgets
HCS-II Hard-to-find Parts
I²C Bus ICs
IR LEDs & Photodiodes
IR Data Link Parts
IR Remote Control
Laser Diode Controllers
Linear Hall Effect Sensor
Multiplexers & Crosspoints
Power Op Amp
Remote Temperature Sensor
Stepper Motor Drivers
Watchdogs & Power Monitors
8051 Information
and more!

Use **a soldering iron? Get the parts!**

UPS: Ground/2nd day $6.50/9.00 to 46 US states, COD add $4.50. PO Boxes and Canadian addresses. $6 for USPS mail Check, MO, or COD only; no credit cards. no open POs. NC residents add 6% sales tax. Quantity discounts start at five parts Data sheets included with all parts

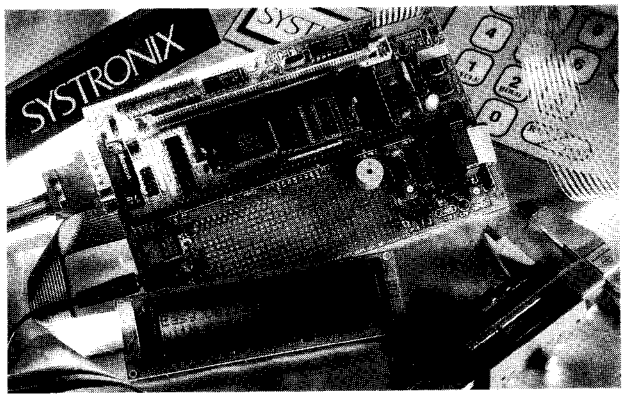
Call/write/FAX for serious/y tempting catalog...

Pure Unobtainium

▶ Your unusual part5 source◀

13109 Old Creedmoor Road Raleigh NC 27613-7421
FAX/voice (919) 676-4525

NEW! UNIVERSAL DALLAS DEVELOPMENT SYSTEM from **$199!**



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

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

#112

received by the host program. When a buffer has been filled with data by the I/O device driver, a DSPCB_GETIO message is sent by the DSP shell to dspAlgMain. The pointer to the buffer is saved and the buffer is sent back to the DSP shell with the dspGetIo function. When the data buffer has been emptied by the I/O device driver, a DSPCB_SENDIO message is sent by the DSP shell to dspAlgMain. A pointer to the buffer is saved and the filter subroutine is called. Filtered samples are put into the buffer, which is then sent to the I/O device driver with the dspSendIo function.

Table 1a lists some examples of DSP shell functions that can be called from your dspAlgMain routine. Most of these functions return an error code that should be checked.

LINKING THE DSP CODE TO THE DSP SHELL

After the DSP main routine and related DSP subroutines have been written and assembled or compiled, the object modules must be linked with the DSP shell. The command sequence to assemble, compile, and link is shown in Listing 3.

The -c switch tells the C compiler (g21) to produce only an object file and not to link. The link command (ld21) specifies a link file which contains the names of the object modules for the FIR subroutine and the main module. The library, lib-debug.a, contains the DSP shell code. The final command line invokes the splitter which properly formats the object code to create a filename.ld file. This file contains the executable code that is loaded onto the DSP.

WRITING THE WINDOWS HOST CODE

Programming in Windows offers key advantages: a "point and click" environment and the ability to run several programs simultaneously and share resources. Windows multimedia extension provides the added convenience of hooks for sound.

Using standard Windows programming techniques, a window can be created with a number of child-window controls which relate to the

Listing 2—*This program offers an example of a main DSP module for a FIR-filter demo program.*

```
#include "dspshell.h"

/****     Define messages that will be sent from the Host     ****/
/*        to the DSP.  Each message has an associated ID.        */

#define msg_CODEC_ON          500
#define msg_CODEC_OFF         501
#define msg_SET_LOWPASS       502
#define msg_SET_HIPASS        503
#define msg_FILTER_TOGGLE     504

/****        Define message parameters        *******************/
/*          (size of message and data buffers                    */

#define SIZE_DATAMSG    1     /* Size of message buffers   */
#define  SIZE_IO_BUFFER 64    /* Size of Codec IO buffers */

typedef struct {              /* Create message structure */
  WORD  hAlg;                 /* The DSP alg handle       */
  WORD  hApp;                 /* The Host app handle      */
  WORD  wMsgId;               /* The message ID           */
  WORD  wLen;                 /* The message data length */
  WORD  wData[SIZE_DATAMSG]; /* The message data         */
} MSG_STRUCT;

MSG_STRUCT msg1, msg2;        /* Use 2 message structures */
DSPIOCONF devConf;            /* IO configuration struct */
DSPIOCAPS devCaps;            /* IO capabilities struct  */

/***********      Declare global variables and arrays     ********/

int   InputDevice;            /* Handle to Codec input         */
int   OutputDevice;           /* Handle to Codec output        */
int   hAlg;                   /* Handle to DSP algorithm       */
int   hApp;                   /* Handle to Host app            */
int   TalkThru = 1;           /* Turn filter on (0) or off (1) */
int   FilterType=0;           /* 0 = low Pass, 1 = Hi Pass     */
int   buffer1[SIZE_IO_BUFFER]; /* Data buffer for DAC output    */
int   buffer2[SIZE_IO_BUFFER]; /* Data buffer for DAC output    */
int   buffer3[SIZE_IO_BUFFER]; /* Data buffer for A/D input     */
int   buffer4[SIZE_IO_BUFFER]; /* Data buffer for A/D input     */
int   *lastInputBuffer;       /* Pointer for input buffers     */

/*****        Declare external functions and data          *****/

extern void initFilter(void); /* Initializes filter */
extern int  sFilter(int, int);/* FIR filter routine */

/*********************************************************/
/*  void dspAlgMain(WORD wCbId, WORD wParam1,            */
/*                        WORD  wParam2, WORD wParam3)*/
/* Description    The dspAlgMain function is the entry     */
/*                 point to the algorithm. The DSP Shell   */
/*                 calls dspAlgMain with messages.         */
/* Parameters: WORD wCbId       callback message ID       */
/*             WORD wParam1     message dependant value */
/*             WORD wParam2     message dependant value */
/*             WORD wParam3     message dependant value */
/* Return Value: none                                      */
/*********************************************************/

void dspAlgMain(WORD wCbId,WORD wParam1,WORD wParam2,WORD wParam3)

int   *pBuf;                  /* Pointer to output data buffers */
int   wTemp;
int  i, wNumDevs;
MSG_STRUCT *pMsg;               /* ptr to a message structure */
```

(continued)

Listing 2-continued

```
switch(wCbId) {
  case DSPCB_INIT:                /* Sent by DSP Shell for init */
     hAlg = wParam1;              /* Save DSP algorithm handle */
     hApp = wParam2;              /* Save Host app handle      */
     msg1.wLen = SIZE_DATAMSG; /* init message data length */
     msg2.wLen = SIZE_DATAMSG:

     /* Request some messages from the DSP Shell          */
     dspGetMsg(hAlg, (int *)&msg1, sizeof(MSG_STRUCT));
     dspGetMsg(hAlg, (int *)&msg2, sizeof(MSG_STRUCT));

     initFilter();                /* initialize filter buffers */

    /**** find out how many I/O devices there are   ******/
     wNumDevs = dspGetNumIoDevs();   /* how many devices? */
     for  (i=0; i<wNumDevs; i++){
       dspGetIoDevCaps(i, &devCaps, sizeof(DSPIOCAPS));
       /* only want 1848 ports, ignore other DAC */
       if(strcmp (devCaps.szDeviceName,"AD1848") == 0){
          if (devCaps.dir == DSPIODIR_IN) InputDevice =i;
          if (devCaps.dir == DSPIODIR_OUT) OutputDevice = i;

     }
     /****   initialize the Codec for 8 kHz              ****/
     dspGetIoDevConf(InputDevice, &devConf, sizeof(DSPIOCONF));
     devConf.rate = 8000;
     devConf.mode = DSPIOMODE_MONO;
     dspSetIoDevConf(InputDevice, &devConf, sizeof(DSPIOCONF));
     break:                     /* end of DSPCB_INIT msg handling */
  case DSPCB_CLOSE:
     dspStopIo (hAlg, InputDevice);
     dspStopIo (hAlg, OutputDevice);
     break;
  case DSPCB_SENDIO: /* buffer was emptied by output I/O device */
     if  (wParam3==0){
       pBuf = (int *)wParam2;    /* save pointer to buffer */
       /* if talkthru mode, just copy in buffer to out buffer */
       /* if not talkthru, put filtered samples in out buffer*/
       for  (wTemp = 0; wTemp < SIZE_IO_BUFFER; wTemp++){
         if(TalkThru) pBuf[wTemp] = lastInputBuffer[wTemp];
         if(!TalkThru) pBuf[wTemp] = sFilter(
                              lastInputBuffer[wTemp], FilterType);

       /* requeue the output buffer */
       dspSendIo(hAlg, OutputDevice, pBuf, SIZE_IO_BUFFER);

     break:
  case DSPCB_GETIO:               /* Buffer was filled by I/O device*/
     if  (wParam3==0){
       lastInputBuffer = (int *)wParam2; /* pointer to buffer */
       /* requeue the input buffer */
       dspGetIo (hAlg,InputDevice,(int *)wParam2,SIZE_IO_BUFFER);

     break:
  case DSPCB_GETMSG:
     pMsg = (MSG_STRUCT *) wParam2;  /* save pointer to message *
     switch(pMsg->wMsgId) {
       case msg_CODEC_ON:
          /* initialize I) buffers, 2 for Xmit and 2 for Rcv */
          for  (wTemp=0; wTemp<SIZE_IO_BUFFER; wTemp++){
             buffer1[wTemp]=0;
             buffer2[wTemp]=0;
             buffer3[wTemp]=0;
             buffer4[wTemp]=0;

          /* send buffers to IO for input/output     */
          dspSendIo (hAlg, OutputDevice, buffer1,  SIZE_IO_BUFFER)
```

(continued)

execution of the DSP routines. In my example, I have created six controls:

- Turn Codec On
- Turn Codec Off
- Enable/Disable Filter
- Select High-Pass Filter
- Select Low-Pass Filer
- Exit Program

I've developed a sample host program, however there isn't room to list it here. It is posted on the Circuit Cellar BBS for you to refer to in the following discussion.

During the processing of the WM_CREATE message, the controls are created and the algorithm is loaded using the dspLoadAlg function. The dspLoadAlg function has the following syntax:

```
LONG dspLoadAlg(dwDeviceID,
  dwInstance, lphAlg,
  lpAlgSection, dwCallback,
  dwFlags);
```

where dwDeviceID is the DSP device number (the DSP manager can determine this for you), dwInstance represents application instance information, lphAlg is a far pointer to an algorithm handle, lpAlgSection is a far pointer to a null-terminated string identifying the algorithm to be loaded, dwCallback specifies the address of a call-back function where Windows processes messages from the DSP shell, and dwFlags is a callback flag.

Since I am only sending messages from the host to the DSP (no messages are being sent from the DSP to the host), I selected the CALLBACK-NULL flag. If messages are to be sent from the DSP to the host, a CALLBACK-FUNCTION or CALLBACK-WINDOW flag should be used to instruct the DSP manager to send the message to your WndProc routine or your own callback function.

The host program responds to the six controls by sending a message to the DSP shell via the DSP manager. Table 1b includes some examples of DSP manager functions that can be called from your host Windows program.

Listing *1-continued*

```
            dspSendIo (hAlg, OutputDevice, buffer2, SIZE_IO_BUFFER);
            dspGetIo (hAlg, InputDevice, buffer3, SIZE_IO_BUFFER);
            dspGetIo (hAlg, InputDevice, buffer4, SIZE_IO_BUFFER);
            dspStartIo (hAlg, OutputDevice);  /* start the Codec */
            dspStartIo (hAlg, InputDevice);
            break:
         case msg_CODEC_OFF:
            dspStopIo (hAlg, InputDevice);   /* stop the Codec */
            dspStopIo (hAlg, OutputDevice);
            break:
         case msg_SET_LOWPASS:
            FilterType=0;
            break:
         case msg_SET_HIPASS:
            FilterType=1;
            break;
         case msg_FILTER_TOGGLE
            if (TalkThru){
               TalkThru = 0:

            else {
               TalkThru = 1;

            break;

      dspGetMsg(hAlg,(int *)wParam2,sizeof(MSG_STRUCT));
      break:                    /* end of DSPCB_GETMSG */

   case DSPCB_SENDMSG:          /* unused messages */
   case DSPCB_ERROR:
   case DSPCB_GETDATA:
   case DSPCB_IDLE:
   case DSPCB_SENDDATA:
   case DSPCB_TIMER:
   break:
```

A case statement is used to check the ID of the control (button) which has been selected by the user. The dspSendMsg function is used to send the appropriate message to the DSP. In my example, a simple message is sent that has no additional data except a message ID. A NULL value is used instead of the required pointer to the message data and the data size is 0. If a message is sent with a buffer of data, a pointer to the array will be specified in the form (LPVOID)&msg1.wBody.

The example host program only sends messages. For more flexibility, you would want the host program to receive messages as well. The host program can receive messages from the DSP through WndProc or through a callback function that you specify. These options are specified in the dspLoadAlg function call.

Listing 3—*The commands to assemble, compile, and link the DSP code are best placed in a batch file.*

```
set sdk=..\..\..

g21 -a %sdk%\bin\psa\psa.ach -I%sdk%\include bobfir.c -c
g21 -a %sdk%\bin\psa\psa.ach -I%sdk%\include fir.dsp -c
g21 -a %sdk%\bin\psa\psa.ach -I%sdk%\include regutil.dsp -c
ld21 -i linkfile -a %sdk%\bin\psa.ach -e bobfir
     -user %sdk%\bin\psa\libdebug.a -lib -x -g
spl21 bobfir bobfir -ld

set sdk=
```

SUMMARY

If you want to make the most of your DSP-based sound card, customize it by writing your own code. For example, if you don't like the quality of the music synthesis on a DSP-based sound card, remember it's the software, not the hardware, you're listening to. Take a stab at writing your own code. With a large installed base of these sound cards, the task may even turn out to be a profitable venture. ■

Bob Fine is product support manager for Analog Devices' DSP *products. He has over 10 years of DSP system design experience and has published a number of articles on DSP design issues, He may be reached at* bob.fine@analog.com.

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.

REFERENCES

Microsoft Windows Software Development Kit (SDK), *Multimedia Programmer's Guide* and *Multimedia Programmer's Reference.*

Microsoft Windows Device Driver Kit (DDK), *Multimedia Device Adaptation Guide.*

Petzold, Charles. *Programming Windows: the Microsoft guide to writing applications for Windows 3.1,* 3d ed. 1992.

ADSP-2100 Family Assembler Tools & Simulator Manual, 1st ed. 1993.

ADSP-2100 Family C Tools Manual, 1st ed. 1993.

ADSP-2100 Family C Runtime Library Manual, 1st ed. 1993.

SCOPE Technical Reference, 2d ed. 1994.

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

6th ANNUAL CIRCUIT CELLAR DESIGN CONTEST

Compiled by Janice Marinelli

Congratulations!

1st Place

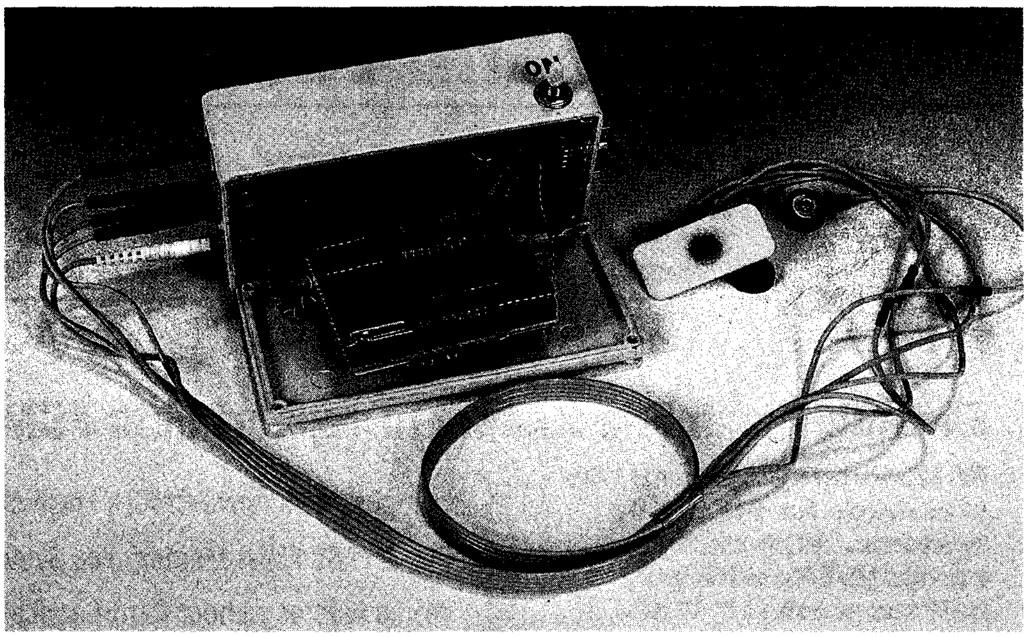
DESIGN CONTEST WINNER

Eric Wilson and Gregg Norris
The Eye Mouse

The eye mouse offers individuals with extreme disabilities the opportunity to communicate and control their environment.

For several years now, the electrooculogram has been classified as a bioelectric phenomena just as the electrocardiogram (ECG) and the electroencephalogram (EEG). The EOG measures the changes in potential which emanate from the eye orbit whenever the eye moves. This microvolt-signal passes throughout the extra cellular fluid in the head and is easily detected by scalp electrodes on the face.

Taking advantage of this technology, Eric designed an electrode interface which enables individuals to manipulate a common "mouse pointer" on a personal computer using their eyes—no bodily movement is necessary. The ionic currents on the surface of the skin are converted to the electronic currents necessary to drive the computer mouse. The ionic current is amplified, filtered, offset, and then run through a PIC16C71.



The bulk of the software, written as a mix of Forth and 68HC 11 assembler, was designed by creating a flowchart representing how the EOG data is analyzed and interpreted to create mouse commands. The left/right and up/down EOG voltages are directly mapped to the X-Y cursor coordinates on the computer screen. Position shift is initiated after the eyes have looked in a direction for half a second, and the cursor continues moving in that direction until it is stopped by the double blink of the eyes. Since it takes one second for most individuals to blink their eyes twice, mouse movement is limited to 1 cm/s.

Test results reveal that disabled persons were able to make 16 selections in any order in about 1.5 min. This marks a significant improvement in the quality of life for people who previously needed an interpreter to only communicate yes/no eye blinks.

2nd Place

DESIGN CONTEST WINNER

Herbert McKinney, Jr. XMAS in July

A talking Santa? No. An automated stocking stuffer? No. Remote-controlled Christmas tree lights? No. But, you're getting closer.

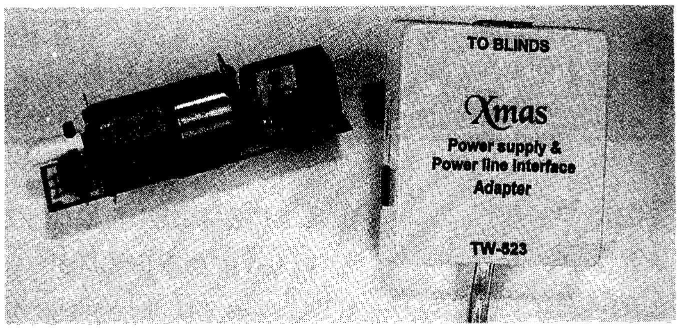
XMAS actually stands for an X-10 compatible Miniblind *Au*tomation System. Faced with needing to control some out-of-reach blinds and the options of either scaling custom-made, hand-rubbed, slate-topped speakers to get to them or buying a $300-per-blind commercial solution, Herbert chose to go back to the drawing board.

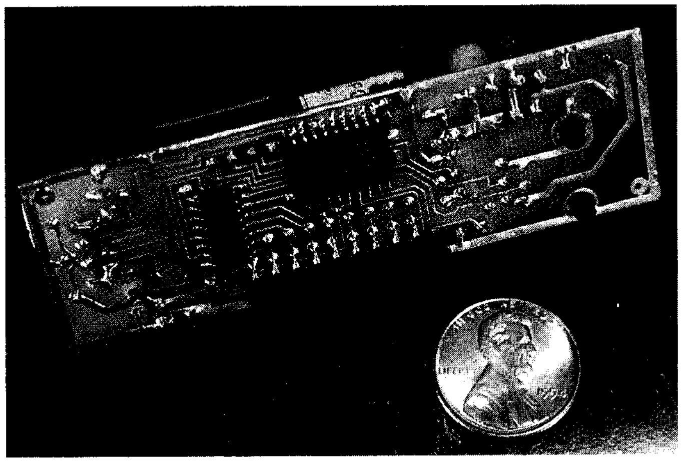
Instead, he designed control circuits and a drive motor which fit within the blind's header assembly. Anadapterunit connects to an X-10 interface module (TW523) and power supply. The modular cable may be concealed in a traditional telephone jack or run through the walls to outlets in the window sills for a more professional installation. Up to 256 units may be connected with each unit having a unique address or up to eight units may be grouped into one unique address.

XMAS interprets X-10 on, off, bright, and dim commands as open, close, up step, and down step, respectively. There are 16 stages between full up and full down. Control and programming of XMAS units can be supplied by many currently available home-control products.

Although this was Herbert's first PIC project, the code turned out to be the smaller battle. The challenge proved to be in placing a power supply, crystal, 3 flat packs, a motor and driver, optosensor, 2 limit switches, 15 pull-up resistors, and an RJ- 11 jack onto a 2.82-square inch, single-sided circuit board.

So, Herbert's speakers are no longer in danger, protected by a controller much less expensive than $300 per unit.





3rd Place

DESIGN CONTEST WINNER

Scott Heiserman and Clark Oden Remote-controlled Speaker Selector

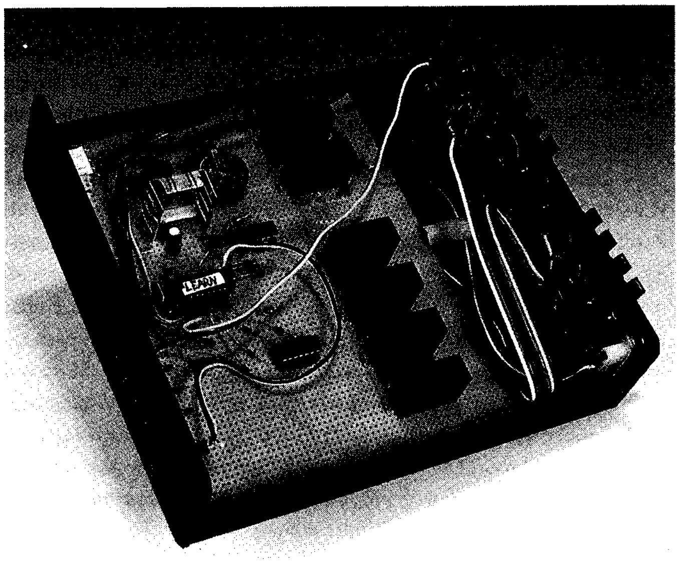
The RCSS is an add-on, home-stereo component designed for loudspeaker selection from any infrared (IR) remote controller. The remote can be used with off-the-shelf IR repeater systems.

Scott's primary concern for the remote was compatibility. But, since he couldn't find appropriate learning algorithms and specialized chips, he 'decided to develop his own. Finally, after hours of examining IR codes, Scott found a pattern that could be implemented in firmware.

It appears that all remote code sequences come in a preamble, space, code-information format. The real breakthrough came when Scott determined that the absolute difference between the low and high count following the preamble is significantly greater than 0 for a binary 1 and near 0 for a binary 0. This reduced the difficulty of programming by an order of magnitude.

Despite this breakthrough, the IR code learning and recognition as well as the complete, intuitive user interface still use most of the microcontroller's resources. **Every** I/O bit, all32 bytes of RAM, and all but 7 of the 504 bytes of ROM in Motorola's 68HC705 are used.

The electronic parts for this prototype unit were under $50.



Honorable Mentions

Tom Ward
An EPROM Emulator

Although a pessimist when it comes to the latest in microprocessor debugging and development tools, Tom recognizes the value of debuggers when it comes to multiple software modifications.

He needed an emulator which could provide both 16- and 32-bit EPROM arrays as well as a high-speed and menu-driven interface. Of course, to achieve high speed, the command-line interface has to be executed from within batch files.

The tricky part came in determining how to have multiple emulatorscontrolledfromasingle parallel port and cable. To solve this problem, Tom tied all emulators to a single parallel port bus. He had the emulators work out which bytes they should be storing.

And, it is compact. The whole system is built on a couple of PCBs, taking up no more real estate than a single 32-pin DIP. With a combination of standard and surface-mount components, the design stands no more than an inch tall as well. Since power is taken directly from the target socket, there is not even the need for a power supply.

Connection to the emulator is made via a 26-pin female IDC connector. This lets multiple emulators run from a single cable in a multidrop fashion and just fits lengthwise over a 32-pin device.

So, here you have it-one of the smallest, fastest, and cheapest emulators around.

Christopher Morris
RDS Prospector

Last year, the National Association of Broadcasting adopted a standard for the Radio Broadcast Data System, frequently referred to as RBDS. Based on the European RDS, it enables an ordinary car or home radio to display digital station symbols which are recovered from the inaudible 57-kHz band of an otherwise regular FM multiplex signal.

Chris's goal was to build an inexpensive add-on RDS decoder unit that would work with any existing FM tuner while also displaying full radio text messages and exploring various other types of transmissions.

Although several manufacturers have inexpensive RDS chips which include signal demodulator circuitry, he chose Philips's SAA6579T because it is readily available and well documented. He also picked Microchip's PIC16C84 for its fast Harvard architecture and EEPROM structure.

Predictably, the real trick was decoding the 57-kHz signal. RDS uses the concept of syndromes for synchronization and error detection. Data falls in 26-bit blocks and must be multiplied by a 26 x 10 matrix to give a 10-bit syndrome. You have to keep reading additional bits until you've achieved synchronization. The calculation has to be done repeatedly in less than 1 bit time which, at the rate of 1187.5 bps, is 842 µs.

Kenneth Pergola
Micro-bRISC Device Programmer

Necessity is the mother of invention. Or so it seems, for when Ken wanted to program the PIC16C64 and found that Microchip still had no design tools available for it, he set off to create his own.

The Micro-bRISC device programmer offers low-cost, high-speed support of the PIC16C64, '71, '74 and '84 microcontrollers. In fact, Micro-bRISC firmware and software were specifically written in an open-ended manner so that support of future PIC16C6*x*, '7x, '8x microcontrollers would not require a firmware upgrade.

The first step in creating the device came by successfully following Microchip's serial programming specifications rather than the parallel method used to program the '71 and '84 chips. Although these first accomplishments fell somewhat short, they did set him on the right path.

Using the PIC16C57 to coordinate everything, Ken achieves one-chip design with Micro-bRISC. To further streamline the development process, the programmer interfaces to any PC printer port for quick data transfer.

For More Information

The quality of this year's entrants tookagreat leap forward this year. For most entrants, the design and presentation was outstanding. We encourage all Design Contest winners and entrants to write complete articles about their projects. Design Contest articles willbehighlighted with the finish-line logo you see throughout this spread.

If you would like **more information about any of the projects, you will have to patiently wait for the full article to appear in an upcoming issue. We will** *not give* **out the addresses or phone numbers of any of our project designers.**

If you absolutely *must* **get in touch with any of them, you may send a letter to the designer in care of us. We will forward your letter to the designer. Send letters to Design Contest Winner,** *Computer Applications Journal,* 4 Park St., Vernon, CT 06066.

DESIGN CONTEST WINNER

FEATURE ARTICLE

A RISC Designer's New Right ARM

Development Boards and Design Specifics

Art Sobel

Without development tools, new processors just don't stand a chance. Last month we introduced you to the features of the ARM processor. This month, we take a look at how the hardware of an ARM development board works.

This is the second article in a three-part series about the ARM processor. In the first article (see *CAJ*51), I covered the background of the ARM, overviewing existing systems and briefly introducing ARM architecture. In this article, I will examine an ARM development board and describe its design. The third article, coming in the new year, discusses software development with ARM.

When I joined VLSI in 1991, the ARM600 was still on the drawing board. It was apparent to me that the chip would need a usable development board for engineers and programmers and a generous supply of RAM and ROM for potential applications. I wanted to ensure that other developers would have a useful example of a successful ARM design. Marketing, on the other hand, didn't want to build a workstation or personal computer with operating system headaches.

I also wanted the board to exist independent of a PC. This decision contrasted to the previous development board for the ARM2. The BlueStreak board plugged into an AT's ISA slot and operated as an ISA bus master for downloading code. As you can imagine, some "compatibles" did not want to operate with an ISA master and hung up.

An independent board meant that the development software could be located on UNIX workstations as well as PCs. Workstation availability broadened our market to include IC designers and high-end programmers. Also, a stand-alone board permitted the umbilical cord to be broken; a user application could exist without a host computer. The board was aptly dubbed ***PID,*** which stood for Processor Independent Development.

ARM Ltd. (in England) decided to build a small board for the ARM60 and call it ***PIE*** (Processor Independent Evaluation). In addition, they were writing cross-development software and a PIE monitor program called DEMON for DEbug MONitor. VLSI and ARM Ltd. worked together to ensure that DEMON could be ported to the PID board and that development

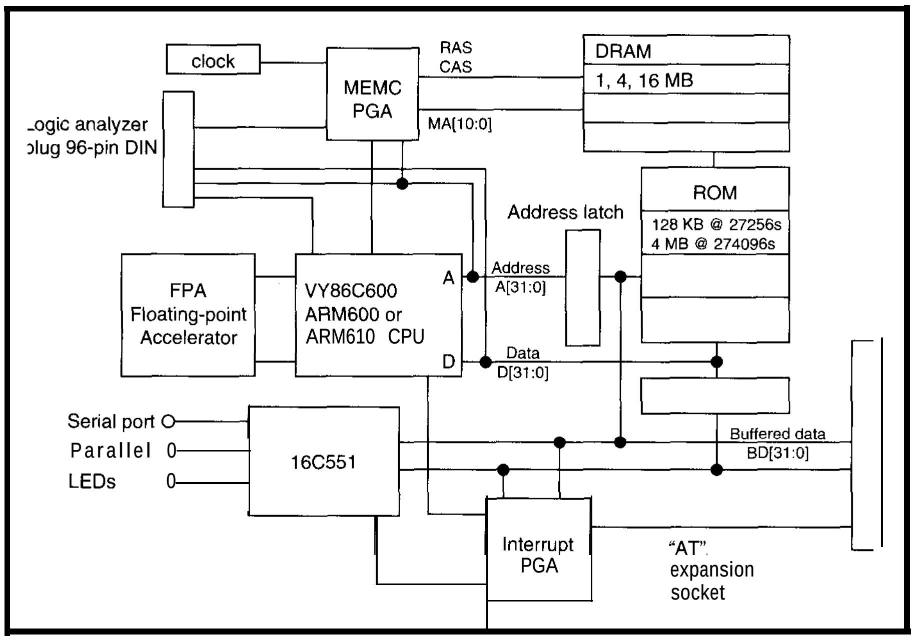


Figure 1--The Processor *Independent Development (PID) board simplifies ARM development*

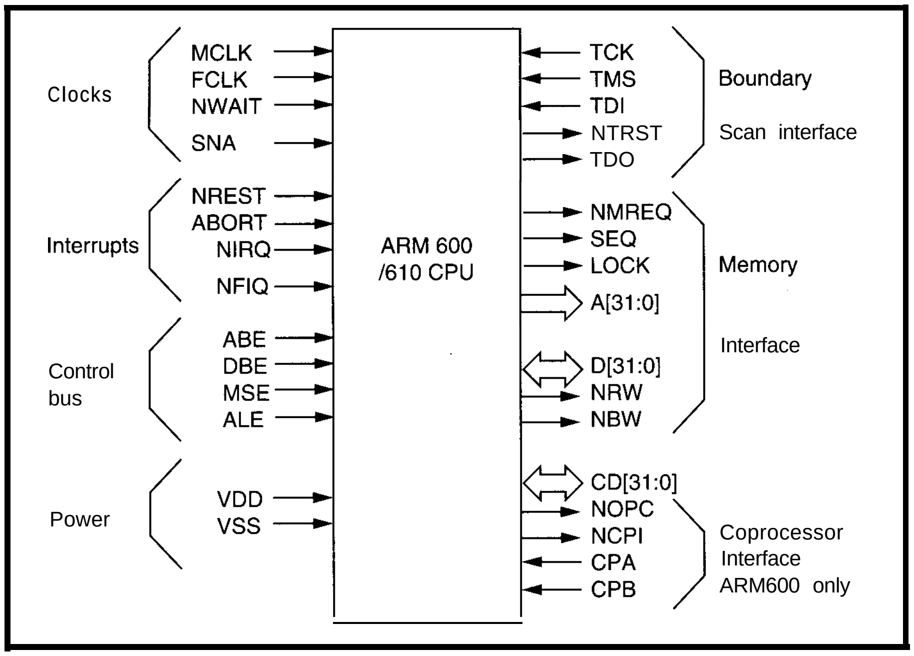


Figure **2**—*The ARM600/610 processor signals can be divided into seven groups.*

software would work the same on both boards.

However, at this time, the ARM600 is an orphan since it has been replaced by the ARM610, which is being mass produced for the Apple Newton. The ARM610 comes in the very small 144-pin Thin Quad Flat-Package and lacks a coprocessor port while the ARM600 is in a larger 160-pin Plastic Quad Flat-Package. But, as far as electrical and software specifications, they operate interchangeably.

In this article, I will be describing the PID board because

- it is more usable than the PIE series (the PIE has a very limited amount of SRAM)
- PID has proven more popular with our developers
- we have a greater quantity of PIDs available for development

OVERVIEW OF THE PID BOARD

Figure 1 shows the simple board architecture of the PID. The PID board uses the ARM600 as its CPU, which reflects the PID's ancient, 1992 origin. The ARM600 has an ARM6 CPU with 4 KB of cache, an MMU, and a write buffer. In addition, the ARM600 has a coprocessor port to be used with an optional floating-point coprocessor or other customer-invented coprocessors.

The PID features a large amount of DRAM (1-16 MB) using standard 80-ns, 30-pin, 8-bit SIMMs. Four EPROMs contain the C-DEMON and floating-point emulator code. Board logic is provided by QuickLogic PGA-based memory and interrupt controllers.

The board also has a 16C551 serial and parallel port chip. Communication to the host computer is accomplished through the serial port. The serial port is connected to a DE-9 male connector with PC pinout, and connection to the PC is achieved through a null modem cable. The parallel port is terminated at a DB-25 female connector with a pinout compatible with the PC. The 16C551 has an extra general-purpose port which is used for lighting the LEDs.

Once the application has been developed, it can be transported to the EPROM and live independently from the host. Expansion to a customer's application is provided by an AT-like slot which has a full 32-bit interface.

FAST CACHE AND FASTER SRAM

Because the FCLK for the ARM600 is going so fast (24 MHz, 40-ns cycle), external SRAM would need to have 20-ns performance or better to match the performance of the cache in an ARM60-based system. Otherwise, it would be unable to compensate for pad delays on and off chip.

To avoid these problems, internal memory is used. This turns out to be a better solution in many ways since internal memory is faster and has much better total power dissipation. When the cache is enabled, bus use drops to about 15%.

GETTING TO KNOW THE ARM

Before designing a board for the ARM600 or ARM610, it is wise to get acquainted with the CPU pins (see Figure 2). Grouped together are FCLK, MCLK, and NWAIT. The internal ARM6 CPU uses FCLK to access internal cache or to do internal cycles and uses MCLK to access external memory or I/O. FCLK and MCLK can be stopped to save power since the CPU is fully static.

A CPU bus cycle is defined as running from an active-negative edge of the clock to the next active-negative edge. The NWAIT signal is ANDed with the MCLK and is used to cancel the positive part of the cycle. So, a constant NWAIT (low active) signal will also cause the internal MCLK to stop.

Bus control pins ABE, ALE, DBE, MSE, and CBE are used to affect the way the ARM operates on a computer

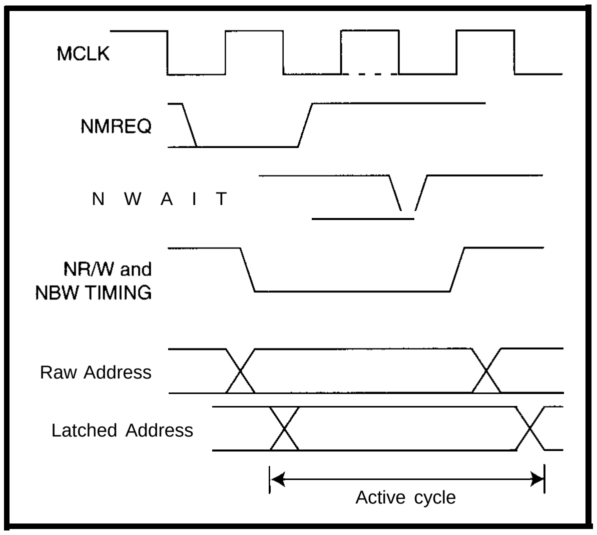


Figure J-Among *the ARM interface signals are those that support memory accesses.*

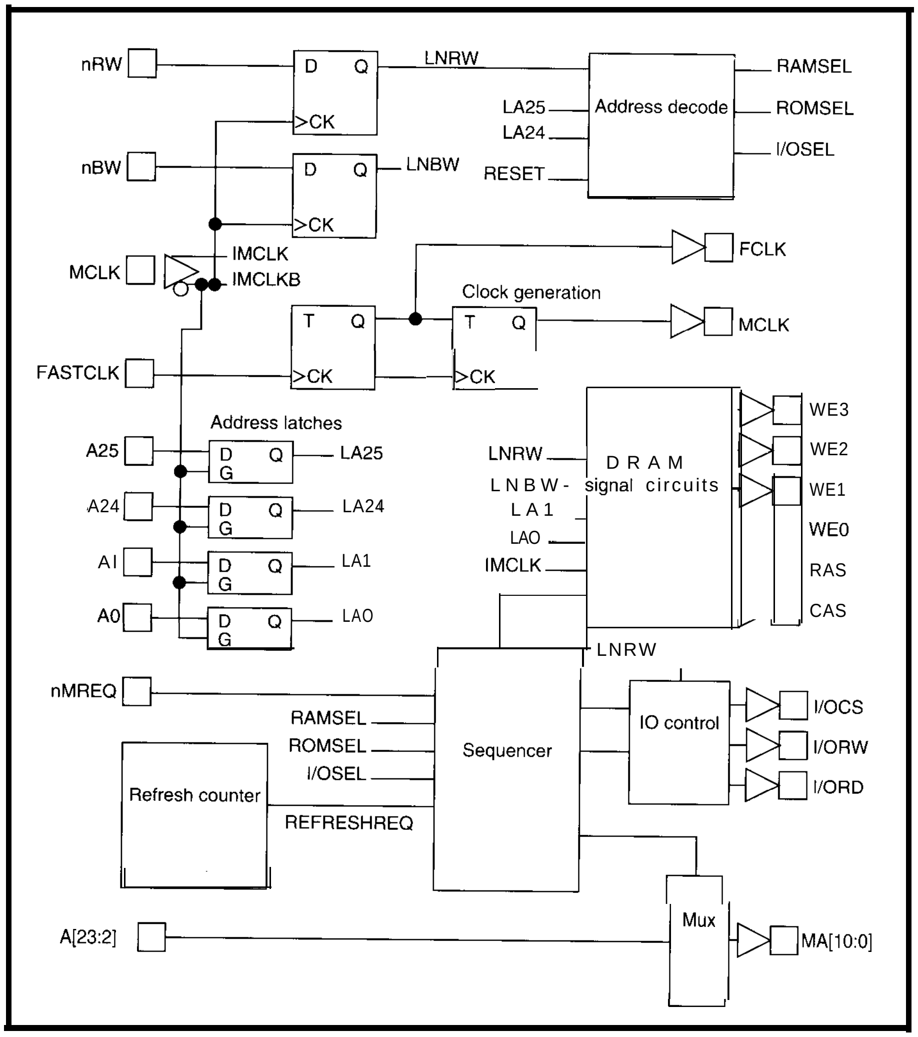


Figure 4-Control *of the PID memory is achieved with fhe MEMC3 QuickLogic PGA.*

bus. The ABE (address bus enable) signal enables the address pin drivers. If a DMA device is enabled on the same bus, the ARM's address lines can be turned off to allow for bus sharing. DBE (data bus enable) is used similarly to disable the ARM data bus. ALE (address latch enable) is used to delay the output of the address bus so it is valid throughout the bus cycle. MSE (Memory Sequential Enable) can be used to disable the request to the memory controller if two processors are sharing the bus.

ARM access to the bus is controlled by the NMREQ, SEQ, LOCK, NRW, and NBW pins. The NMREQ signal low indicates that the ARM will request the bus in the next cycle. SEQ indicates that the address will be N+4. In the ARM600, SEQ is always active when NMREQ is active because internal cycles turn MREQ off as the clock is switched to FCLK. In many memory systems (e.g., DRAM), sequential access can be made much faster than random access.

The NRW (not read / write) is used to indicate a read or write on the bus. The NBW (not byte/ word) tells the bus controller that the processor is accessing bytes or words. When bytes are written, the lowest significant byte in the string register is repeated four times on each byte lane. Only one byte lane can have an active write enable, which is decoded from NBW, NRW, and the lowest two bits of address.

LOCK tells the memory controller that a lock-swap instruction is being executed. This instruction is not interruptible to ensure that operating system constructs, like semaphores, work correctly.

In the ARM600, coprocessor pins are used when a floating-point unit or another attached instruction-set extender is added. A true ARM coprocessor monitors the instruction stream using the NOPC (not op-code fetch] signal. When it recognizes its own instruction, it waits for the ARM to generate the CPI (coprocessor instruction] signal before responding with a CPA (coprocessor available) signal.

If the coprocessor must take a number of cycles to complete the instruction, it generates a CPB (coprocessor busy) signal to stall the ARM. If there is no such coprocessor, the lack of the CPA signal causes the ARM to go to the undefined instruction trap, which is used, for instance, to emulate a floating-point unit and support the floating-point chip in the generation of transcendental functions, such as sine, log, and so on. The board, however, comes only with a socket for a floating-point coprocessor (the floating-point coprocessor is available from GEC Plessey Semiconductors if you need one).

The timing relationships of some of these signals are shown in Figure 3. To complicate life, the memory request (NMREQ) and sequential access (SEQ) signals appear one cycle before the active cycle. The address and the read / write (NRW) and byte / word (NBW) signals appear one-half cycle ahead of the active cycle.

In the PID design, the address signals are latched throughout the active cycle by using an external-

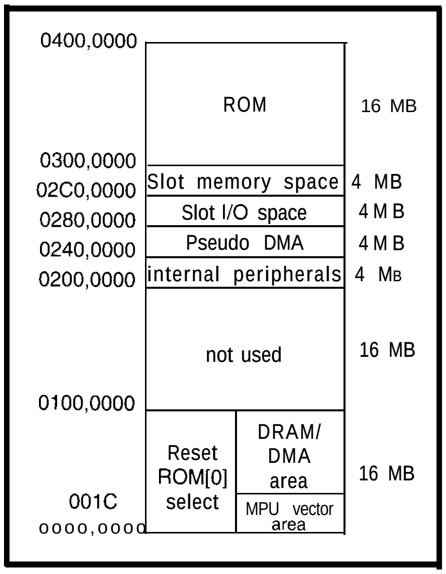


Figure 5—*The PID memory map includes a dual area in low memory used by both a boot ROM and DRAM/ DMA.*

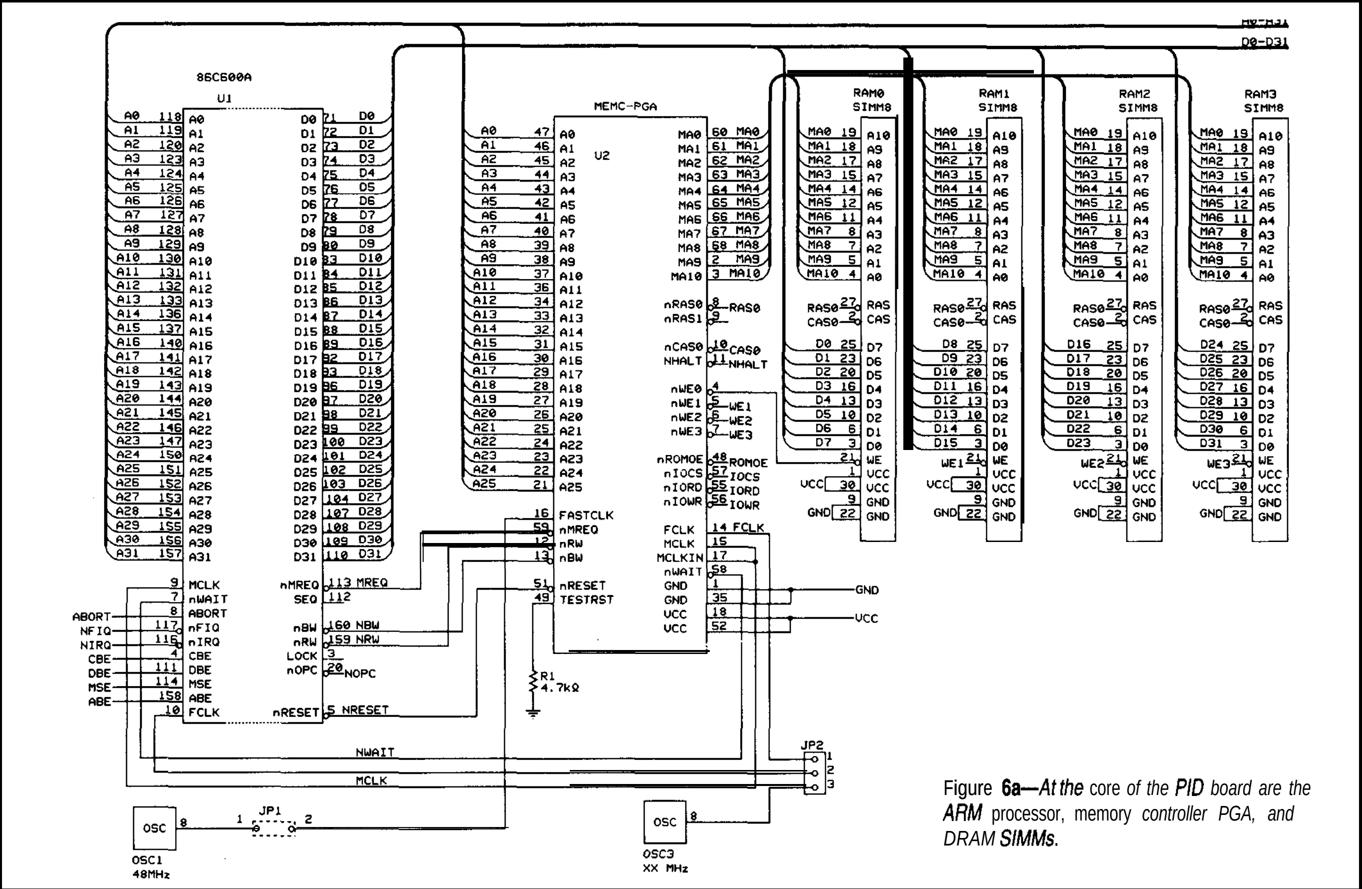
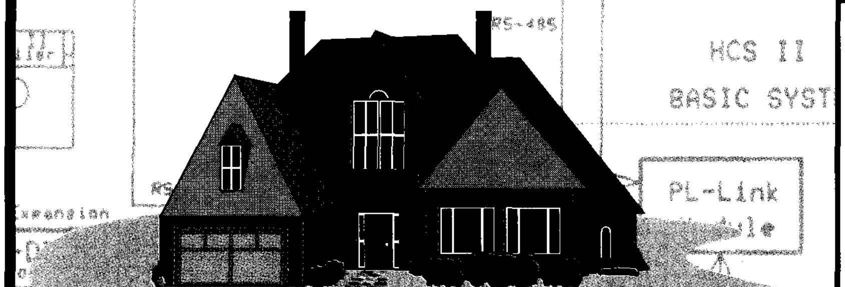


Figure 6a—*At the core of the PID board are the ARM processor, memory controller PGA, and DRAM SIMMs.*

HCS II
Home Control System



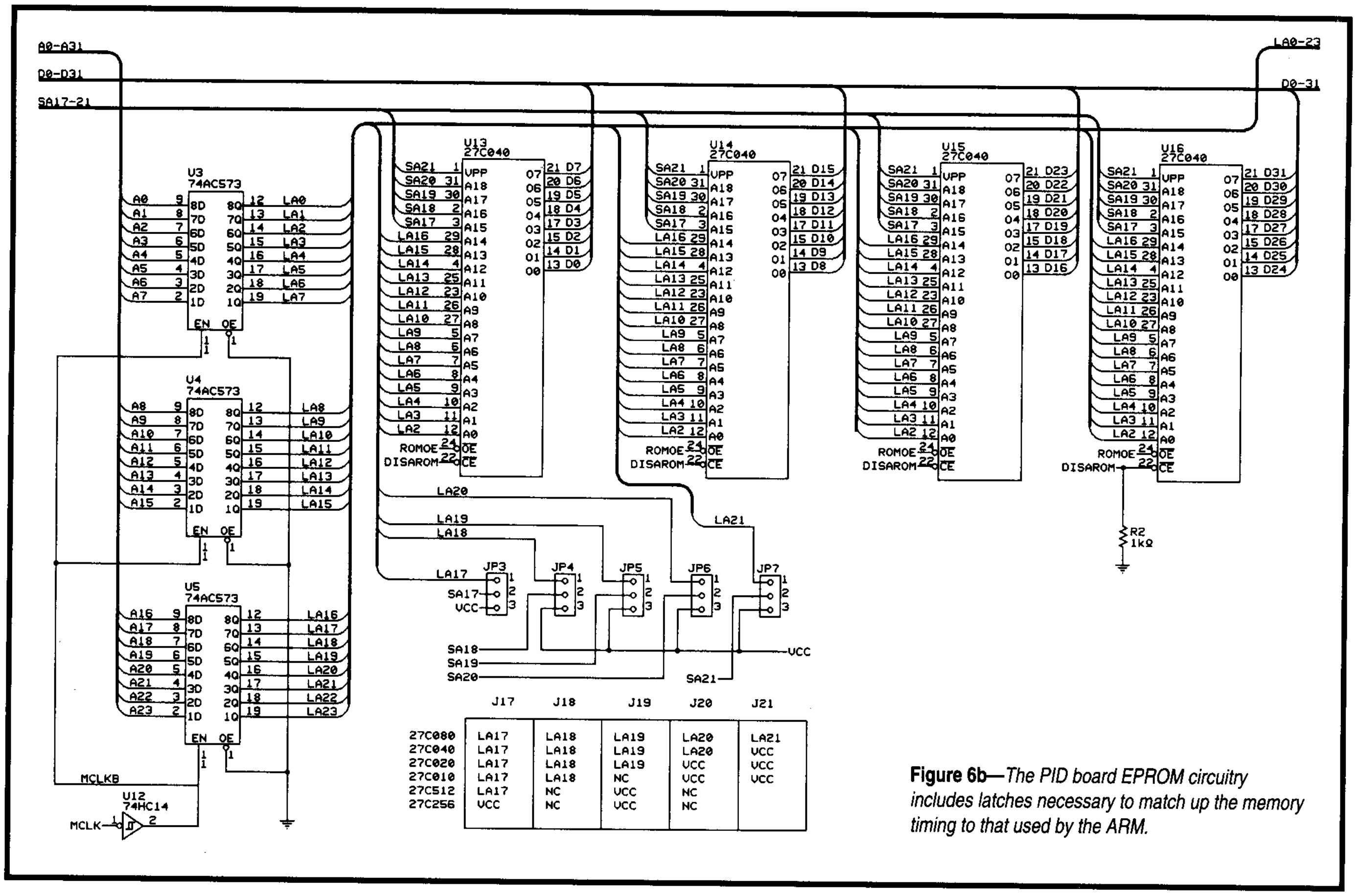
- **Energy Management**
- **Security and Alarm**
- **Coordinated Home Theater**
- **Coordinated Lighting**
- **Monitoring and Data** Collection

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

Circuit Cellar HCS

CIRCUIT CELLAR, INC.
4 Park Street, Suite 12 • Vernon, CT 06066
Tel: (203) 875-2751 • Fax: (203) 872-2204

R11b



	J17	J18	J19	J20	J21
27C080	LA17	LA18	LA19	LA20	LA21
27C040	LA17	LA18	LA19	LA20	VCC
27C020	LA17	LA18	LA19	VCC	VCC
27C010	LA17	LA18	NC	VCC	VCC
27C512	LA17	NC	VCC	NC	
27C256	VCC	NC	VCC	NC	

Figure 6b—*The PID board EPROM circuitry includes latches necessary to match up the memory timing to that used by the ARM.*

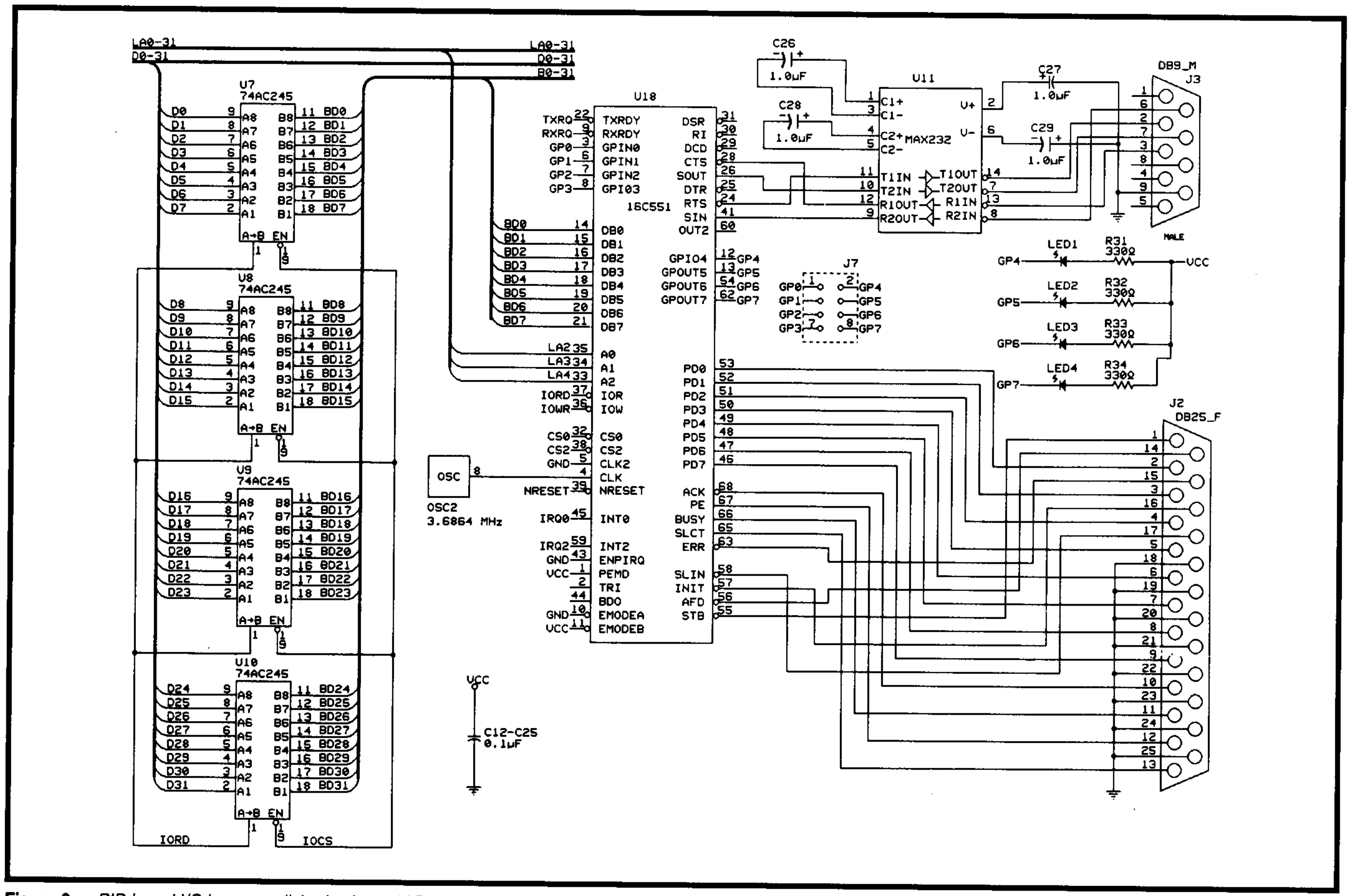


Figure 6c—*PID board I/O is accomplished using a 16C551 serial/parallel chip and companion receivers and drivers.*

address latch made from 74AC573s. The PID board uses statically addressed ROM and I/O. The states of the NBW and NRW signals are captured at the beginning of a bus cycle by the MEMC PGA and are used to generate the appropriate RAM, ROM, and I/O timing.

MEMORY CONTROLLER PGA

Control of the PID memory is achieved with the MEMC3 Quick-Logic PGA (see Figure 4). The memory map (see Figure 5) covers only 64 MB of total memory and is divided into three areas: DRAM, ROM, and I/O. The gate array generates a preset number of wait states for each cycle type.

MEMC3 controls the DRAM multiplexed-address bus, RAS, CAS, WE[3:0], ROMOE as well as the I/OCS, I/ORD, and I/OWR signals. A fixed counter produces a constant 15-µs refresh-request period which forces a CAS-before-RAS refresh sequence. The ARM writes single bytes by replicating the byte in each byte lane (four times total) and activating the proper byte write. When NRW is high and NBW low, the lowest two address bits select the proper WE pin. For word writes, all WE pins are driven low active.

The connections of the ARM600 to MEMC3 and the DRAM SIMMs are shown in the PID schematics (see Figure 6a). A 48-MHz oscillator is connected to the FASTCLK pin. FASTCLK is divided by two to make the 24-MHz FCLK signal and then divided by two again to make the 12-MHz MCLK signal. The memory state machine operates off inverted MCLK and responds to NMREQ and the addresses A[25:24] from the ARM600 to generate memory control signals and wait states.

As Figure 7 shows, the sequencer is in the IDLE state when there is no MREQ from the ARM600 or RFREQ from the internal refresh counter. When there is an MREQ, the logic checks the address range and responds with DRAM and IR (I/O or ROM) cycles. After reset, the ARM processors start execution at address 0. After some initializing of ARM registers, the code jumps to the real ROM location

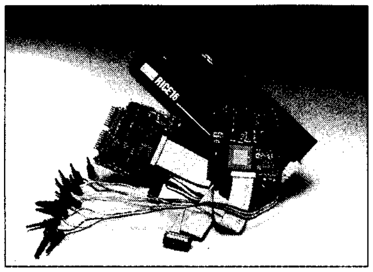
PIC16C5x/16Cxx Real-time Emulators

Introducing RICE16 and RICExx-Juniors, real-time in-circuit emulators for the PIC16C5x and PIC16Cxx family microcontrollers: affordable, feature-filled development systems from **$599** *

* Suggested Retail for U.S. only

RICE16 Features:

- Real-time Emulation to 20MHz for 16C5x and 10MHz for 16Cxx
- PC-Hosted via Parallel Port
- Support all oscillator type5
- 8K Program Memory
- 8K by 24-bit real-time Trace Buffer
- Source Level Debugging
- Unlimited Breakpoints
- External Trigger Break with either "AND/OK" with Breakpoints
- Trigger Outputs on any Address Range
- I2 External Logic Probes
- User-Selectable Internal Clock from 40 frequencies or External Clock
- Single Step, Multiple Step, To Cursor, Step over Call, Return to Caller, etc.
- On-line Assembler for patch instruction
- Easy-to-use windowed software



Emulators for **16C71/84/64** available now!

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

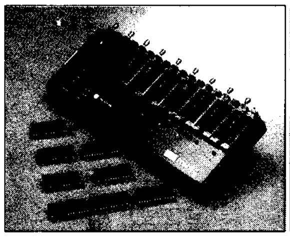
RICE-xx Junior series

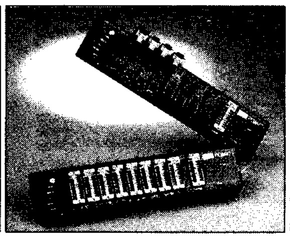
RICE-xx "Junior" series emulators support PIC16C5x family, PIC16C71, PIC16C84 or PIC16C64. They offer the same real-time features of RICE16 with the respective probe cards less real-time trace capture. Price starts at $599.

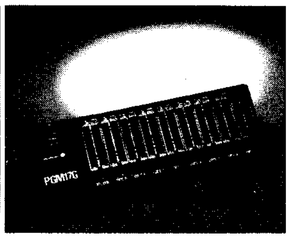
PIC Gang Programmers

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

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


PGM16G: for 16C5x family


PGM47: for 16C71/84



PGM17G: for 17C42

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

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

TransdAtA Advanced Transdata Corporation Tel (214) **980-2960**
14330 Midway Road. Suite 128. Dallas, Texas 75244 **Fax** (214) 980-2937

#117

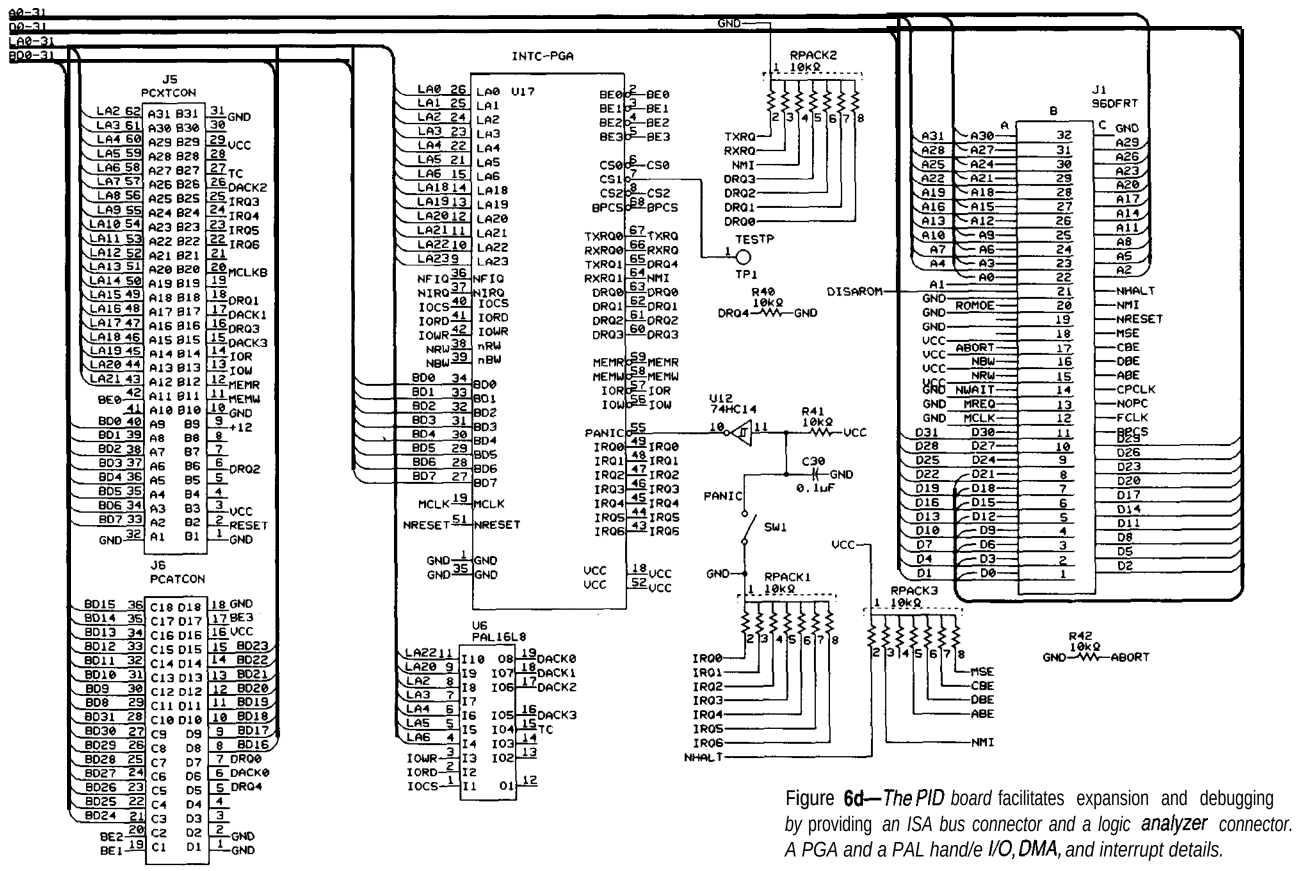


Figure 6d—*The PID board* facilitates expansion and debugging *by providing an ISA bus connector and a logic analyzer connector. A PGA and a PAL hand/e I/O, DMA, and interrupt details.*

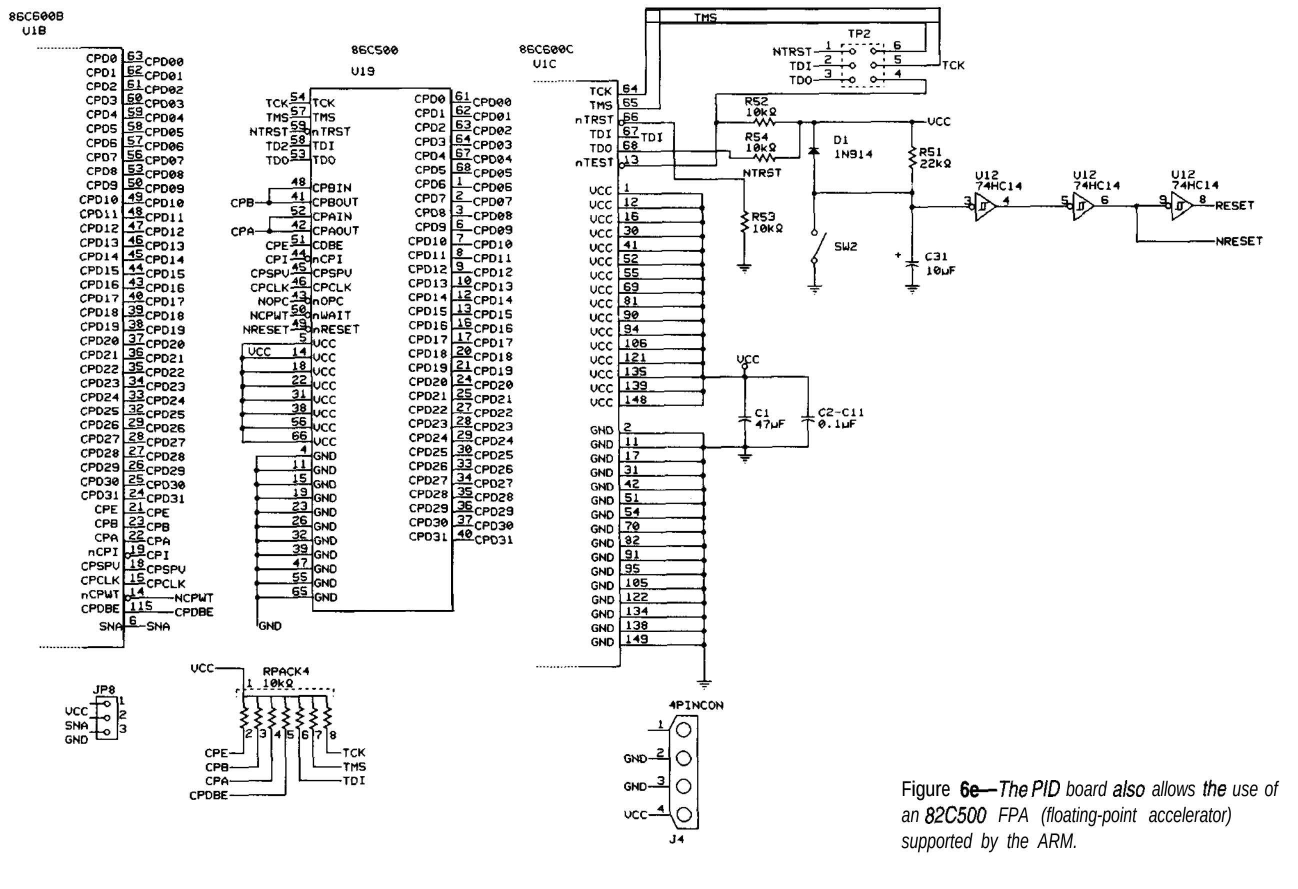


Figure 6e—*The PID board also allows the use of an 82C500 FPA (floating-point accelerator) supported by the ARM.*

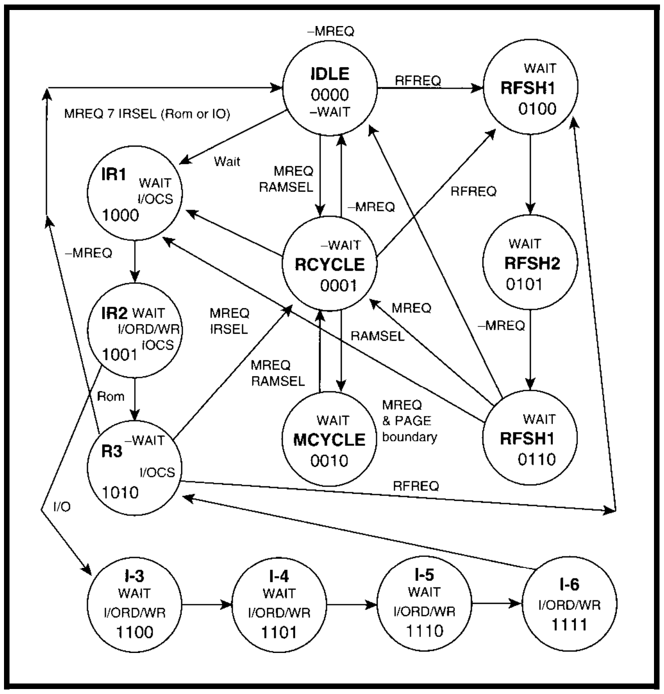


Figure 7-The *PID memory* state machine operates off *the inverted MCLK signal and responds to NMREQ and the addresses A[25:24] from the ARM.*

(at address x3xx xxxx). The address map is then set to normal operation which enables DRAM accesses to low address space.

DRAM accesses are shown in Figure 8. When MREQ is received and DRAM is selected, the RAS signal is asserted in the middle of the first MCLK cycle while the sequencer is still in IDLE state. CAS is asserted during the last half of the next MCLK cycle while the sequencer is in RCYCLE state. If a sequence of MREQ accesses is presented to the memory controller, RAS is held low and CAS cycles with a new column address on each cycle. The effective access time for nonsequential cycles is then two cycles (160 ns), and for sequential cycles, it is one MCLK cycle (80 ns) for a peak of **12** MIPS. The ARM600/610 will not go over a page boundary without getting off the bus for at least one cycle. This eliminates the need for page detect logic, even though it has been included.

The ROM circuitry is shown in Figure 6b of the PID schematics. The ROM data pins are tied to the memory data bus, but the addresses are routed through a set of 74AC573 latches. Because the address bus of the ARM-600 changes half an MCLK early in the access cycle, the addresses must be latched and held to the end of the cycle by the 74AC573s. These addresses are also used for I/O accesses. ROM access is straight-forward and lasts for three MCLK cycles, asserting NMWAIT for two cycles for a total time of 240 ns. So, when operating out of ROM without cache on, the CPU has an effective band-width of about 4 MIPS peak. See Figure 9 for a graphic depiction of ROM accesses.

I/O connection to the 16C551 is shown in Figure 6c of the PID sche-matics. Data is buffered through a set of 74AC245s before going to the I/O sub-system. Since the I/O data is also placed on the expansion slot, it is wise to eliminate the unknown loading that might occur. I/O accesses (shown in Figure 10) are stretched to seven MCLK cycles (480 ns), asserting NMWAIT for six cycles. The state machine starts at the IR1 state and proceeds from IR2 to 16, and then back to R3 before going back to IDLE or to IR1 for another I/O access.

The I/OCS, I/OWR, and I/ORD signals are provided by the MEMC3, but are further decoded by the INTWT PGA into I/OR or I/OW and MEMR or MEMW signals. This decoding is done in order to support the split I/O and memory space on the connector. The I/O signals are soooo long because many AT-compatible boards will not work any faster than the leisurely AT-bus signals. In fact, that is why there is an MEMC3. The MEMC2, a much faster board, produced a 240-ns I/O cycle with 160-ns I/OWR and I/ORD. This speed caused so much incompat-ibility problems that we created the MEMC3 to meet the need for a slower AT-bus signal.

INTWT PGA DESCRIPTION

The expansion-bus connector, the logic-analyzer connector, and the INTWT QuickLogic PGA are shown in Figure 6d of the PID schematics. The INTWT (interrupt with timer), shown in Figure 11, provides the BE[3-0] (I/O Byte selects) and the I/O control

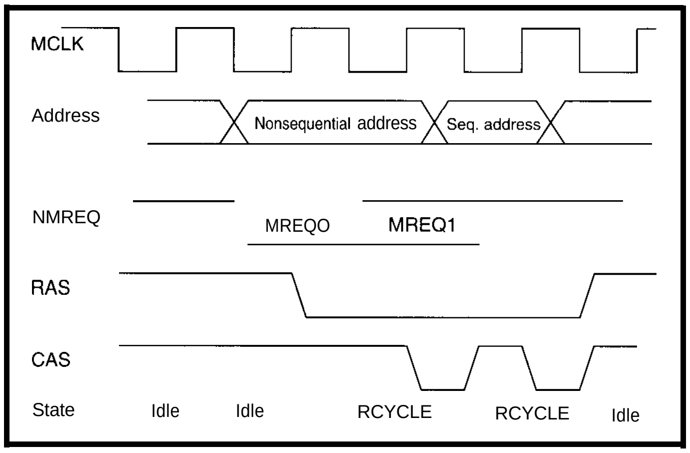


Figure &The *DRAM control signals perform normal RAS-before-CAS memory accesses.*

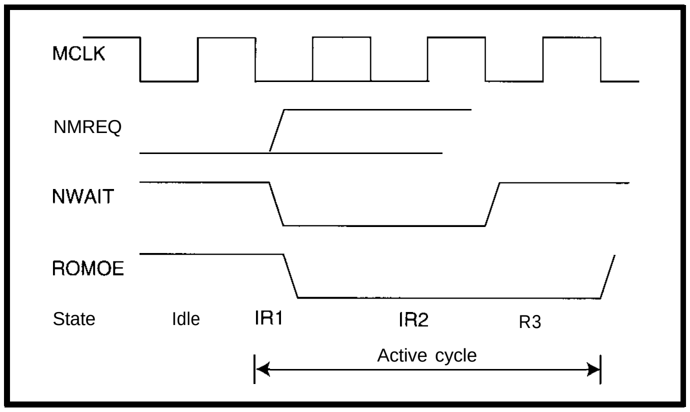


Figure 9—*ROM accesses slow down processor throughput considerably.*

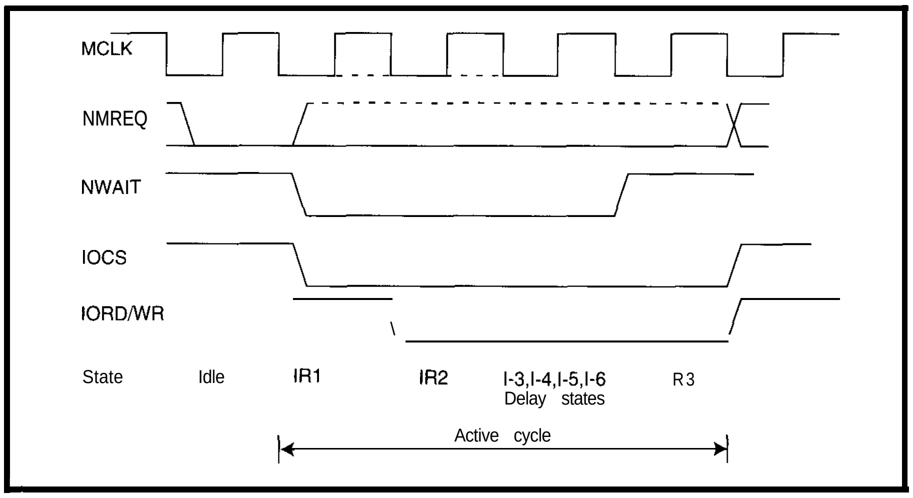


Figure 10—The I/O access timing is slowed down a great deal to be compatible with AT-type expansion boards

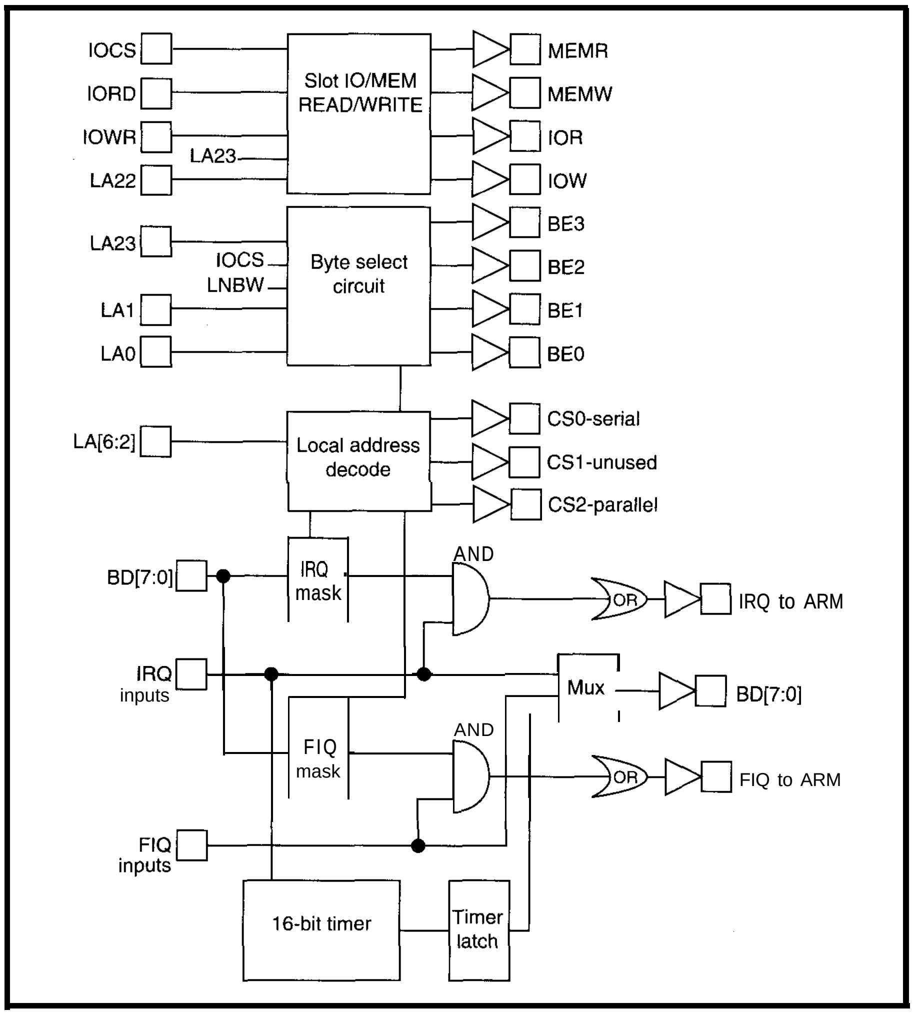


Figure 11 --The INTWT (interrupt with timer) PGA handles the ISA-bus I/O and interrupt details.

signals (MEMR, MEMW, I/OR, and I/OW) that go to the AT bus. The interrupts are divided into two classes: IRQ and FIQ (fast interrupts).

The AT bus has both I/O and memory space. Therefore the huge raw I/O space (16 MB) is split into a local area, a slot DMA area, a slot I/O area, and a slot memory area. Each slot area has 4 MB A[21:0] of addressable space. The AT slot's normal interrupts are routed to the INTWT and are ANDed with a mask register and ORed together to form the IRQ (a lower priority interrupt to the ARM60).

The DRQ signals are likewise ANDed and ORed to form the FIQ. An additional PAL16L8 is used to generate DACKs and the TC signals, which operate peripherals using the DMA channels. The DACKs are memory mapped and are operated by the processor. A constant timer is used to generated a periodic interrupt on IRQ1 every 10 ms. The INTWT internal timer can be used as an interrupt source only, or the counter contents can be latched and read to get fine-timing information. Functions such as interrupt vectors and priority are handled by the interrupt software.

EXPANSION CONNECTOR

The standard AT pinout has been changed to include all 32 bits of interface on the connector [see Table 1). It was probably a mistake to make any modifications to the AT pins, but the slot connector still has been very useful for test and development and is sufficiently compatible to allow the SMC Ethernet board and an old-time AT hard-disk controller to run.

8- AND 16-BIT ACCESSES ON AN AT BUS

Normally, an AT bus has a mechanism to pack word accesses into two byte accesses. I voted to jettison this capability for hardware simplicity. If the user wants this function, it can be done in software. Each potential application would be understood by the programmer and designer well enough to figure out how the interface would operate. Listing I has examples of data packing and unpacking.

FLOATING-POINT SUPPORT

Figure 6e of the PID schematics shows the 82C500 FPA (Floating-Point Accelerator). The ARM600 and ARM700 have a coprocessor port that can be used on the PID2 board with the FPA. Currently, the FPA is supplied by GEC Plessey only. The ARM700, also supplied by Plessey, has twice as much cache (8 KB) as the ARM600 and can be carefully placed on the same board (you have to be careful because it is surface mounted).

Future versions of PID will have the CPU mounted on a daughtercard to make processor replacement something that even I could do.

ARM-BASED MICROCONTROLLERS

In addition to designing the PID board, one my first jobs at VLSI Technology was overseeing the development of glue logic parts for an AM29000-based laser printer controller. It bothered me that we were using this chip since its instruction bus and data bus shared only one address bus, and this function could easily be done by the ARM. In addition, it was clear to me that we could and should include both the processor and the laser printer peripherals on one chip.

AMD had the same idea and soon came out with the AM29200, which added to the CPU core most of the logic for the laser printer. This move placed the chip set recently developed by VLSI for AMD customers into the discard pile. However, as Tom Cantrell pointed out (*CAJ* 33), the AMD29200 has very high power usage, which effectively limits the operating frequency without forced-air cooling.

When VLSI began making plans for an ARM-based microcontroller, I was given the responsibility of designing the laser printer chip. After talking with several customers and gathering some great logic designers together, we developed the 86C650.

Unlike the AM29200, the 86C650 uses only 500 mW with a 50-MHz main clock and develops 9500 Dhrystones for a sustained 5 MIPS. The 86C650 has built-in memory control and important peripherals such as synchronous and asynchronous serial ports, parallel port, laser video port, 4-channel DMA, and an external peripheral expansion bus. And, since every chip needs a development and test board, the 86C650 microcontroller chip was placed onto the 86LX650 laser printer controller board.

VLSI has developed another ARM microcontroller aimed at portable communicators based on the PCMCIA electrical specifications and form factor. This little guy, called Ruby, has a peripheral PCMCIA interface, 8530 synchronous serial port, 8250 asynchronous serial port, timer, interrupt controller, I/O extension bus, and 8255 parallel port. Currently, there is a PC-based development board for it, but it is specialized for wireless PCMCIA developers.

PIN	A (IC)	B (circuit)	C (IC)	D (circuit)
1	GND (I/O CHK)	GND	BE1 * (SBHE)	GND (MEM16*)
2	DO7	RESET	BE2* (LA23)	GND (I/O16*)
3	DO6	+5 v	D24 (LA22)	NC (IRQ10)
4	DO5	NC (IRQ9)	D25 (LA21)	NC (IRQ11)
5	DO4	NC (-5)	D26 (LA20)	DRQ4 (IRQ12)
6	DO3	DRQ2	D27 (LA1 9)	DACK0* (IRQ15)
7	DO2	NC (-12 V)	D28 (LA1 8)	DRQO (IRQ14)
8	DO1	NC (0 WS)	D29 (LA17)	D16 (DACKO*)
9	D00	+12 V	D30 (MEMR*)	D17 (DRQO)
10	NC (I/O CH RDY)	GND	D31 (MEMW*)	D18 (DACK5*)
11	BEO' (AEN)	MEMW	DO8	D19 (DRQ5)
12	LA21 (SA19)	MEMR*	DO9	D20 (DACK6*)
13	LA20 (SA18)	vow*	D10	D21 (DRQ6)
14	LA1 9 (SA17)	I/OR*	D I I	D22 (DACK7*)
15	LA18 (SA16)	DACK3*	D12	D23 (DRQ7)
16	LA17 (SA15)	DRQ3	D13	+5
17	LA16 (SA14)	DACKI*	D14	BE3* (MASTER*)
18	LA15 (SA13)	DRQ1	D15	GND
19	LA14 (SA12)	NC (Refresh)		
20	LA1 3 (SA1 1)	1 2-MHz MCLK		
21	LA1 2 (SA10)	NC (IRQ7)		
22	LA1 1 (SA09)	IRQ6		
23	LA1 0 (SA08)	IRQ5		
24	LA09 (SA07)	IRQ4		
25	LA08 (SA06)	IRQ3		
26	LAO7 (SA05)	DACK2*		
27	LAO6 (SA04)	NC (TC)		
28	LAO5 (SA03)	NC (BALE)		
29	LAO4 (SA02)	+5 v		
30	LAO3 (SA01)	NC (OSC = 7.7 MHz)		
31	LAO2 (SAOO)	GND		

Note: DACK[3:0] requires PAL16L8 to be installed

Table I-The *PID* expansion slot is based on the standard *AT pinout, but has been changed to include all 32 processor bits.*

More microcontrollers are on the horizon. One that I am enthusiastic about is the ARM7500. The ARM7500, announced at the fall '94 Microprocessor Forum, contains an ARM704 CPU with 4-KB cache, memory controller for DRAM and EPROM, interrupt, timer, keyboard, mouse, joystick, VGA, and sound output. I will report on this part when the development board and software are ready.

Table 2 offers a comparison between the various ARM development boards. All of these boards have a common debugger environment. The operating system DEMON includes a ROM debugger with a µC/OS real-time kernel. The development hosts include the PC, SPARC, HP, IBM PowerPC, and NeXT (Black).

THIRD-PARTY BOARDS

An ARM2-based module is available from Applied Data Systems. Called the Pixel *Press,* it attaches to a computer parallel port, has an SVGA output port, and a serial port for

Features	NPIE	PID	86LX650
Processor	ARM60	ARM600	86C650
ROM	1 27C010	4 27C080	l-4 27C080
Main Memory	128-KB SRAMs	l-l 6-MB DRAM	l-l 6-MB DRAM
Communications	RS-232	RS-232	RS-232
		Parallel (Host)	Parallel (Printer)
		Coproc socket	RS-170 Video
Dhrystone	20,000 Q40 MHz	28,000 @24 MHz	9500 Q50 MHz
	SRAM, 1 WS	4-KB cache	80-ns DRAM, 1 WS
Expansion	PC/1 04	AT style	XT style

Table 2—*There are several ARM development boards available, each with its own advantages.*

debugging code. It can operate in a high-level, interpretive-language mode which includes move, draw point, draw line, circle, fill, and character drawing. It is used primarily for offloading remote displays such as air traffic control or other industrial uses.

SUMMARY

Simple development boards can be made or bought that can harness the power of the ARM600 and ARM6 10. Microcontroller versions are being made and introduced that will make the task of using the ARM processors even easier.

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

Listing 1—*The normal A J-bus* mechanism of **packing** word accesses into two *byte* accesses must be done in software on *fhe PID* board.

a) Pack bytes to double word

```
; IO Port pointed to by R5, R4 has DMA address, R6 is DMA limit,
; RO-R3 are used as scratchpad registers

DMALOAD8
      LDRB  RO, [R5]          ; Load first byte into R0=0,0,0,B0
      LDRB  R1, [R5]          ; Load second byte into R1
      LDRB  R2, [R5]          ; Load third byte into R2
      LDRB  R3, [R5]          ; Load fourth byte into R3
      ORR   R0,R0,R1,LSL #8   ; RO <-0,0,B1,B0
      ORR   R0,R0,R2,LSL #16; RO <-0,B2,B1,B0
      ORR   R0,R0,R3,LSL 1124; RO <-B3,B2,B1,B0
      STR   R0,[R4],#4
      CMP   R4,R6
      BNE   DMALOAD8
```

b) Pack word to double word

```
; IO Port pointed to by R5, R4 has DMA address, R6 is DMA limit,
; R0-R2 are used as scratchpad registers

      MOV   R2,#-1            ; R2=0xFFFFFFFF a useful constant
DMALOAD16
      LDR   R0, [R5]          ; Load first word into R0=X,X,B1,B0
      LDR   R1, [R5]          ; Load second word into R1=X,X,B3,B2
      BIC   R0,R0,R2,LSL #16; RO <-0,0,B1,B0
      ORR   R0,R0,R1,LSL #16; RO <-B3,B2,B1,B0
      STR   RO, [R4],#4
      CMP   R4,R6
      BNE   DMALOAD16
```

c) Store double word to bytes

```
; IO Port pointed to by R5, R4 has DMA address, R6 is DMA limit,
; RO-R3 are used as scratchpad registers

DMASTORE8
      LDR   R0,[R4],#4        ; qet DMA DATA from memory
      MOV   R1,#4               R0 <-B3,B2,B1,B0,01
      STRB  RO, [R5]            Store byte into IO=B0,B0,B0,B0
      SUBS  r1,#1
      MOVNE RO,RO,LSR #8        shift data to right=0,B3,B2,B1
      BNE   %B01                more shifts and stores
      CMP   R4,R6
      BNE   DMASTORE8
```

d) Store double word to word

```
; IO Port pointed to by R5, R4 has DMA address, R6 is DMA limit,
; R0-R2 are used as scratchpad registers

DMASTORE16
      LDR   RO, [R4],#4         get DMA DATA from memory
                                RO <-B3,B2,B1,B0
      STR   R0, [R5]            Load first word into I/O=B3,B2,B1,B0
      MOV   R0,R0,LSR #16       shift data to right
      STR   R0, [R5]            Load first word into I/O=0,0,B3,B2
      CMP   R4,R6
      BNE   DMASTORE16
```

SOURCES

PIE and PID boards, ARM coprocessors, and ARM Cross-Development Kit 86CDEV-C (including cross-development software, toolkit, and manuals):

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

Other suppliers of ARM processors and information:

GEC Plessey Semi. (U.S.)
1500 Green Hills Rd.
Scotts Valley, CA 95066
(408) 438-2900

GEC Plessey Semi. (U.K.)
Cheney Manor
Swindon, Wiltshire
United Kingdom SN2 2QW
(0793) 51800

Sharp Electronics, Inc.
5700 NW Pacific Rim Blvd.
Camas, WA 98607
(206) 834-2500

Other ARM suppliers:

Applied Data Systems, Inc.
409A East Preston St.
Baltimore, MD 21202
(410) 576-0335
Fax: (410) 576-0338

CASM Compiling Assembler
Nikos
1126 Taylor Draper La.
Austin, TX 78759

I R S

410 Very Useful
411 Moderately Useful
412 Not Useful

DEPARTMENTS

56 Firmware Furnace

66 From the Bench

72 Silicon Update

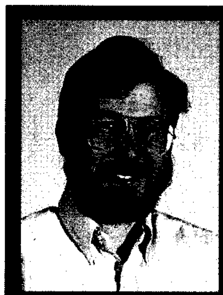
78 Embedded Techniques

85 ConnecTime

FIRMWARE FURNACE

Ed Nisley

Journeyto the Protected Land: Fancy Text Output and a Boot Mystery



It's scary to realize just how easy it is to introduce a system bug so obscure it only shows up under exactly the right combination of circumstances. Ed came across one while showing how to display diagnostic text and it's a beauty.

Our base camp at 1 megabyte communicates with the outside world using 9600-bps serial data and a few LEDs. Before venturing further into the protected-mode wilderness, we need more bandwidth for on-the-scene reports.

A serial link can display relatively slow status information such as a register dump following a system crash. It simply cannot keep up with the torrent of information needed to track the activity of a task-switching operating system. If FFTS has 100 task switches per second, it can send only 10 characters to the serial port during each task!

One of Murphy's corollaries says if you don't display all the information all the time, the most interesting crashes will show the least information.

The Firmware Development Board's Graphics LCD Interface is a better choice for a status display. It doesn't interfere with the PC's BIOS or video-display hardware, and a 640 x 200 panel has lots of room. The only catch is that the CPU must build each character literally dot by dot. This means we should make some timing measurements.

If your embedded application doesn't use the PC video display, of course, you can show status information on a standard CRT. Should your desk have room for two monitors, you can devote a second video channel to the debugging display. The hardware character generator makes video output much faster than the roll-your-own LCD interface.

This month, we'll add the protected-mode code required for character output to both a VGA board and the Graphics LCD Interface. I'll also explore a mystery in one of the drivers that may save your bacon some day.

ARRANGING THE CHARS

DOS regards even the fanciest video hardware as a glorified Teletype: characters appear one by one starting at the upper left. When the bottom line fills up, a vertical scroll makes room for more text. The speed of that operation contributes mightily to the board's DOS-video benchmark rating. There is no standard way to position the cursor or change text colors without using ANSI control strings.

Fortunately, we use hardware that doesn't have to look like a dimwit terminal. The real-time system-status output will fit neatly on a fixed-format screen which doesn't scroll vertically. Because a program generates all the output text, presumably without typing errors, we don't need even rudimentary character editing. Even better, we can add features as we need them rather than writing a huge lump of code at once.

The status display does require cursor positioning and color control, but the prospect of writing an ANSI command parser in protected-mode assembler gave me pause. Rather than get bogged down in overly complex routines, I opted for a simpler system with binary codes. If you'd like to write a full ANSI decoder, the details are in *CAJ* 46!

The code in Listing 1 copies a string to the video display. When the loop detects a VIDCMD byte, it calls the command decoder to interpret the next few bytes in the string. A trailing zero marks the end of the string, an idiom familiar to C aficionados.

Listing 2 is the video-command decoder. The snippet of code following each CMP converts the byte after a VIDCMD into a row, column, or attribute. When we need a few more commands, we won't invoke any rocket science to add them, although changing to a table-driven decoder may be a good idea at some point.

Listing 1—This routine displays a character string on the video hardware. The loop scrutinizes each byte to locate cursor and color-control sequences as well as the binary zero marking the end of the string. The input parameters are the segment and offset of the string, which allows the string to reside in any valid data segment. A similar routine produces output on the LCD panel.

```
            PROC   VidSendString
            ARG    StrSeg:DWORD,pString:DWORD
            USES   EAX,ESI,ES

            MOV    EAX,[StrSeg]
            MOV    ES,AX
            MOV    ESI,[pString]
            OR     EAX,ESI             ; skip if pointer is null
            JZ     SHORT @@Done

@@Continue:
            LOOS   [BYTE PTR ES:ESI]; fetch the byte
            CMP    AL,0                ; check for terminator
            JZ     SHORT @@Done

            CMP    AL,VIDCMD           ; command byte?
            JE     SHORT @@CmdCode     ; yes, special case
            CALL   VidPutChar,EAX      ; no, show the char
            JMP    @@Continue          ; and pick up the next one

@@CmdCode:
            CALL   VidCommand          ; yup, invoke command decoder
            JMP    @@Continue          ; and pick up next char

@@Done:     RET
            ENDP   VidSendString
```

This simple command encoding has a gotcha. Setting the cursor to row or column 0 embeds a binary zero in the string. Our command decoder interprets the result correctly because it expects a numeric value rather than a character at that spot. The C-style string routines, which we haven't written yet, will terminate early when they encounter that zero. I opted for a simple solution: the code ignores the high-order bit of the row and column. You can OR each coordinate with 80 hex or just replace each zero with 80.

Remember that only the FFTS kernel will display status and tracing information through this interface. We'll build a more polite routine for user code when we need it.

The process of moving characters and color attributes to the video buffer should be familiar from previous columns as well as your own experience in real mode. There is one exception-we need a segment descriptor for the video buffer!

GAINING ACCESS, LOSING RISK

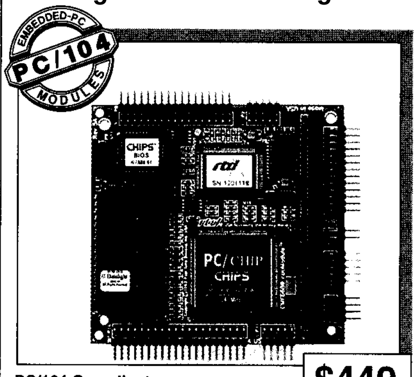
The original PC had two video adapters: a monochrome card (MDA) for decent text and a color card (CGA) for mediocre graphics with ugly text. You could install both cards in the same machine because their I/O ports and memory addresses were different. BIOS and DOS text output went to the "primary" display selected by a system-board switch.

The VGA BIOS ignores that switch. It reads the monitor ID bits through the video cable and sets itself up accordingly. Fortunately, the PC-compatibility barnacles dictate that the BIOS data area must identify the VGA as either an MDAish or CGAish card. The advanced capabilities of the VGA aren't obvious at this level.

The segment descriptor required in protected mode must include the video buffer's starting address and length. The length is easy enough because the BIOS puts the video page size at 0040:004C. A standard 80-

Replace Four Conventional PC/I 04 Modules with One SuperXT™ CMF8680 cpuModule™

Embedded PC/XT Controller with Intelligent Power Management



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

$449 100 pcs.

- PC/XT compatibility with 286 emulation
- 14 MHz, 16-bit 8086 CPU
- +5V only; 1.6W at 14.3 MHz, 1 W at 7.2 MHz
- Intelligent sleep modes, 0.1W in Suspend
- ROM-DOS and RTD enhanced BIOS
- Compatible with MS-DOS & real-time operating systems
- 1 M bootable Solid State Disk & free software
- 4K-bit configuration EEPROM (2K for user)
- 2M on-board DRAM
- IDE & floppy interfaces
- CGA CRT/LCD controller
- Two RS-232 ports, one RS-485 port
- Parallel, XT keyboard & speaker ports
- Optional X-Y keypad scanning/PCMCIA interface
- Watchdog timer & real-time clock

Expand This Or Any PC/1 04 System with the

CM106 Super VGA Controller utilityModule™

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

$223 100 pcs.

Speed Product Development with the

DS8680 Development System

Your DS8680 includes the CMF8680, CM102 keypad scanning/PCMCIA, CM1 04 with 1.8" 85MB hard drive, CM106 SVGA controller& DM5406 12-bit, 100 kHz dataModule™ in an enclosure with external power supply, 3.5" floppy, keyboard, keypad, TB50 terminal board, SIGNAL*VIEW™, SIGNAL*MATH™, MS-DOS, SSD software & rtdLinx™ for just $2950.

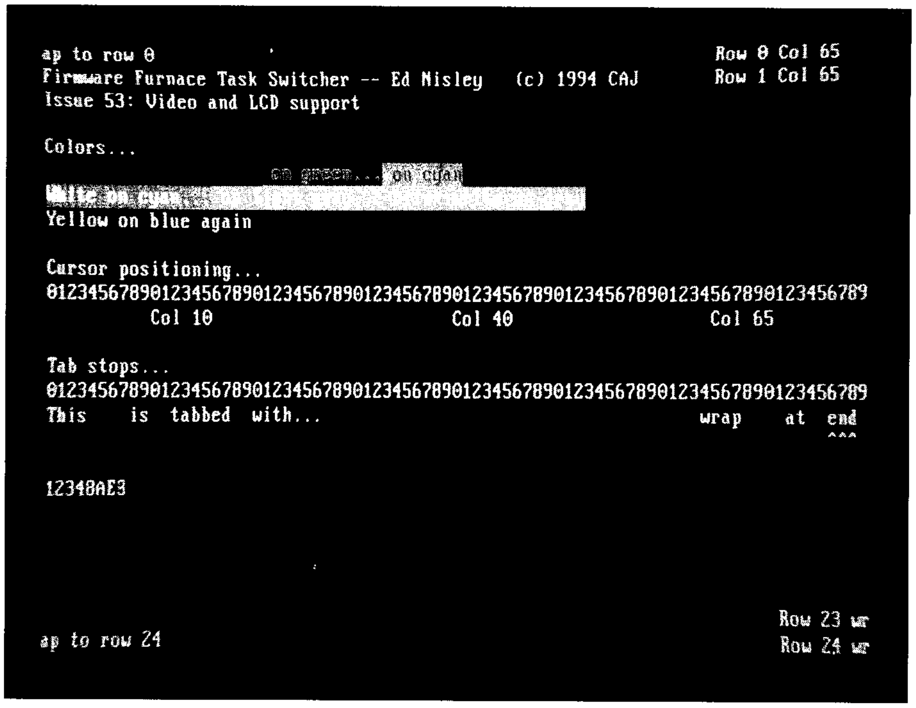
For more information on our PC/104 and ISA bus products, call today.

rtd®

Real Time Devices USA
200 Innovation Blvd. • P.O. Box 906
State College, PA 16803 USA
(814) 234-8087 / Fax: (814) 234-5218

RTD Europa • RTD Scandinavia

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



```
ap to row 0                                              Row 0 Col 65
Firmware Furnace Task Switcher -- Ed Nisley   (c) 1994 CAJ       Row 1 Col 65
Issue 53: Video and LCD support

Colors...
                          on green... on cyan
White on cyan...
Yellow on blue again

Cursor positioning...
01234567890123456789012345678901234567890123456789012345678901234567890123456789
          Col 10                        Col 40                   Col 65

Tab stops...
01234567890123456789012345678901234567890123456789012345678901234567890123456789
This    is  tabbed  with...                                       wrap    at  end
                                                                              ^^^

12348AE3




ap to row 24                                                         Row 23 wr
                                                                     Row 24 wr
```

Photo I--This *test* pattern exercises most *of the video driver's functions by displaying text in a fixed format. Because the driver does not scroll, the screen fexf wraps from the lower-right corner to the upper-left. The eight-*character *counter just* below *the fab-stop test shows an incrementing value that may be slightly blurred due to photo exposure. You can't see blinking fexf on this page, either, sad fo say.*

column, 25-line display requires 4000 (decimal) bytes (don't forget the attributes!) and thus has a 4096-byte page size. Even though those 96 bytes at the end are not visible on the screen, I felt tweaking the official BIOS page size wasn't worthwhile.

As with all protected-mode segments, any write beyond the video page will trigger a protection exception. If you need access to other pages, you must either expand the segment or create additional descriptors for the new pages. Using one descriptor per page is a nice way to handle output from separate tasks.

In fact, putting the screen-page descriptor in the task's LDT ensures that it cannot write into any other page. You can have several tasks, each using the same LDT selector to access the screen, but each selector refers to a different chunk of the video buffer. Might come in handy, indeed!

Oddly enough, the BIOS doesn't store the video-buffer address in its data area. It does, however, save the CRT controller I/O port address at 0040:0063, and that gives us enough to locate the buffer. The bad news is the BIOS stores the MDA address, even if no video board is installed.

Listing 3 presents the video-initialization code. It extracts several values from the BIOS data area, creates the segment descriptor covering the buffer, then performs a simple memory test. If the buffer holds data correctly, the code sets a status flag that enables the remaining video functions, turns off the cursor, and clears the screen.

The `MemPeekReal()` function converts a real-mode segment:off address into a 32-bit linear address and returns the double word at that address in **EAX**. The familiar shift-segment-left-four and add-the-offset dance produces a 20-bit value. The protected-mode startup code created a read-only descriptor that maps all of installed memory starting at address 00000000. That 20-bit address is our first non-trivial 32-bit flat address, albeit with a dozen high-order zeros.

`MemSetDescriptor()` handlesall the hocus pocus required to load a descriptor entry. The code is similar to Listing 6c in last month's column. The functions accessing the buffer load a 48-bit full pointer called *CurPointer* using **LES** instructions. **Cut-Pointer's** segment component is just the `GDT_VIDEO` **selector** for the video-buffer descriptor.

#118

The proof of the coding is in the viewing. Photo 1 shows the video test pattern you should see when you run this month's code. The first few words on the top line are wrapped from the bottom to demonstrate that the screen does not scroll vertically. The driver supports cursor positioning, color changes, and the usual CR, LF, and tab control characters.

The eight-hexit number a few lines below the tab-stop test pattern is a double-word counter driving a simple binary-to-ASCII conversion routine. The last digit or two may be slightly blurred because of the photo's lengthy exposure time. You'll also see a moving dot on the LPT1LEDs to indicate that the test code is really alive inside your system.

That's enough to let FFTS report back in style!

DOING IT WITH DOTS...

I originally planned to use the FDB's Graphic LCD Interface for this part of the project. The folks at the local robotics club convinced me that standard VGAs outnumbered hand-wired LCD panels by umpty-zillion to one. My wounds are healing nicely, thank you very much.

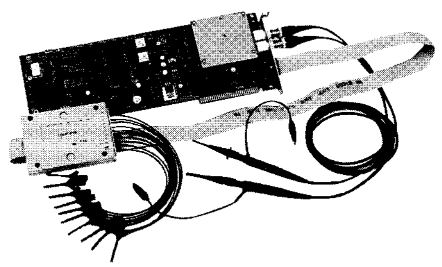
As it turns out though, the top-level code is essentially identical for both displays. I suppose a device-independent interface with automatic color mapping would be a nice touch, but you'll probably just omit the code for the hardware you don't have.. .and that simplifies things a lot.

Earlier this year, you saw that graphic LCD panels have a wide variety of interfaces, timing specifications, and dot arrangements. The low-level, hardware-dependent code required for each panel is bottled up in an assembler file with an obvious name: DMF651 . ASM, for instance. While the drivers may work with similar panels, you must verify that on your own.

Refer back to *CAJ* 46 for a cram course on the BIOS CGA font table and similar matters since I've recycled some of that code into 3%bit protected-mode assembler. I'll skip the detailed background discussions here to save space.

PC-Based Instruments
200 MSa/s DIGITAL OSCILLOSCOPE

HUGE BUFFER
FAST SAMPLING
SCOPE AND LOGIC ANALYZER
C LIBRARY W/SOURCE AVAILABLE
POWERFUL FRONT PANEL SOFTWARE



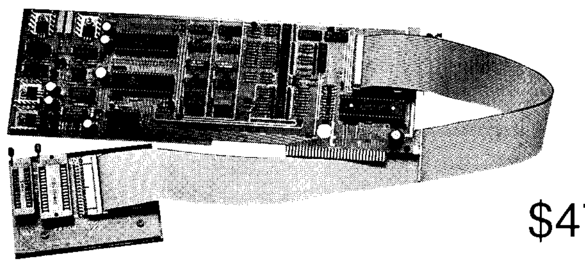
DSO Channels
2 Ch. up to 100 MSa/s
or
1 Ch. at 200 MSa/s
4K or 64K Samples/Ch
Cross Trigger with LA
125 MHz Bandwidth

Logic Analyzer Channels
8 Ch. up to 100 MHz
4K or 64K Samples/Ch
Cross Trigger with DSO

$1799 - DSO-28204 (4K)
$2285 - DSO-28264 (64K)

Universal Device Programmer

PAL
GAL
EPROM
EEPROM
FLASH
MICRO
PIC
etc..

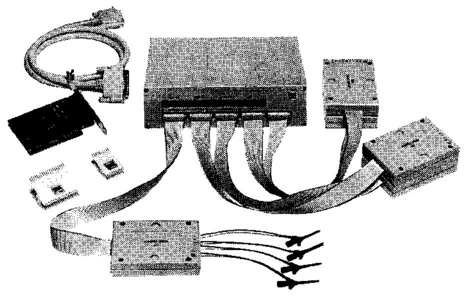


$475

Free software updates on BBS
Powerful menu driven software

400 MHz Logic Analyzer

- up to 128 Channels
- up to 400 MHz
- up to 16K Samples/Channel
- Variable Threshold Levels
- 8 External Clocks
- 16 Level Triggering
- Pattern Generator Option



$799 - LA12100 (100 MHz, 24 Ch)
$1299 - LA32200 (200 MHz, 32 Ch)
$1899 - LA32400 (400 MHz, 32 Ch)
$2750 - LA64400 (400 MHz, 64 Ch)

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

#120

The LCDWriteGlyph routine in Listing 4 converts an ASCII character into the appropriate bit pattern from the BIOS 8 x 8 CGA font. The code substitutes a question mark for any characters beyond the 128 present in that table. Multiplying the resulting numeric character value by the font height gives the character's offset from the start of the BIOS table.

LCDWriteGlyph also combines the current cursor location with the font height and width to get the dot address of the upper-left corner of the character cell on the panel. This calculation is easy because the characters are always aligned on an 8 x S-dot grid. If you want proportional fonts (yikes!), this is the place to keep track of character widths on each line.. .which I leave as an exercise.

The compatibility barnacles anchor the CGA font table to the same address in every PC. That makes the 48-bit cpFont pointer a simple constant in the _prot.const segment. You can substitute a different fixed-pitch font table by aiming cpFont at it and tweaking the font-size constants. If you have a VGA card, you can filch its bitmapped fonts to trade off information density for eye appeal using the techniques I covered in *CAJ* 46.

The LCDWriteGlyph loop fetches successive bytes from the font table and writes them into the refresh buffer using the low-level LCDWriteByte routine. Each panel has a different dot layout, which means a custom routine must distribute the dots into the buffer. That code translates blinking characters into something useful on panels that don't support blinking.

For more grubby, bit-twiddling details, check the source code on the BBS. I've written drivers for three different panels in the hopes they'll either match what you have or be close enough that you can adapt the code without too much trouble.

The test pattern on a TLY365 is essentially the same as Photo 1. The last few digits of the double-word counter are harder to read because LCD panels have a slower response time. If you don't have the LCD interface installed, the code will simply disable itself.

Listing 2-Choosing a *simple binary encoding makes interpreting the cursor and co/or-control commands a/most trivial. The loop in* Listing 1 *defects fhe* VIDCMD *byte preceding each command and invokes this routine to examine the remaining bytes. The next byte determines the operation and specifies how many* data bytes are *included.* If you add more commands, a table-driven decoder would tidy up *the code* by *eliminating the chain of* CMPs.

```
        PROC    VidCommand
        USES    EAX,EBX

        LODS    [BYTE PTR ES:ESI]   ; pick up command byte

 --- cursor positioning

        CMP     AL,VIDROW
        JNE     SHORT @@NotRow

        LODS    [BYTE PTR ES:ESI]   ; set new row
        AND     EAX,07Fh
        CALL    VidSetRowCol,EAX,[CurCol]
        JMP     @@Done

@@NotRow:
        CMP     AL,VIDCOL
        JNE     SHORT @@NotCol

        LODS    [BYTE PTR ES:ESI]   ; set new column
        AND     EAX,07Fh
        CALL    VidSetRowCol,[CurRow],EAX
        JMP     SHORT @@Done

@@NotCol:
        <<< code to set both row & column omitted >>>
        JMP     SHORT @Done

@@NotRC:
;--- on-the-fly color changes

        <<< code to set foreground & background omitted >>>

        CMP     AL,VIDFGBG
        JNE     SHORT @@NotFGBG

        LODS    [BYTE PTR ES:ESI]   ; set foreground
        AND     EAX,OOOOOOOFh
        MOV     EBX,EAX
        LODS    [BYTE PTR ES:ESI]   ; and background
        AND     EAX,0000000Fh
        SHL     EAX,4
        OR      EAX,EBX
        MOV     [CurAttr],EAX
        JMP     SHORT @@Done

@@NotFGBG:
        NOP
@@Done:
        RET

        ENDP    VidCommand
```

PM PERFORMANCE

The video-output routines are the first nontrivial 32-bit protected-mode code we've seen so far, although I'd argue that just getting into 32-bit PM is nontrivial enough for most purposes. In any event, the question comes up: how fast does this stuff run, anyway!

The counter shown on each display provides a convenient way to collect some data because the code is running in a known pattern with all interrupts disabled. I flipped the LPT1 strobe bit at key points during the test loop to produce the upper trace in Photo 2.

Listing 3—*The protected-mode descriptor covering the video buffer must include ifs starting address and length. The BIOS* initializes several key values *in ifs data area during each reset, making if reasonably easy to* figure out what kind *of* video hardware is *installed. The* MemPeek Real() *function returns the double word at the protected-mode address corresponding to a real-mode* **segment:off** *address.*

```
        PROC    VidInitialize
        USES    EAX,EBX,ESI,ES

    extract a few BIOS variables for our use

        CALL    MemPeekReal,BIOS_SEG,4Ah;  columns
        MOVZX   EAX,AX
        MOV     [NumCols],EAX

        CALL    MemPeekReal,BIOS_SEG,84h ; rows 1
        MOVZX   EAX,AL
        INC     EAX
        MOV     [NumRows],EAX

        CALL    MemPeekReal,BIOS_SEG,63h; CRT controller addr
        MOVZX   EAX,AX
        MOV     [CRTCBase],EAX

        ADD     EAX,06h                  ; status port address
        MOU     [CRTStatus],EAX

 --- create a GDT descriptor covering the video buffer
     we tweak the memory starting address based on the CRTC I/O
     address...and, if it's a color display, we assume it's a VGA

        CALL    MemPeekReal,BIOS_SEG,4Ch; video page length
        MOVZX   EBX,AX
        MOV     [PageLength],EBX

        MOV     EAX,000B8000h            ; assume color
        MOV     [CRTAttrCtl],03c0h       ; for VGA Attribute Ctl
        CMP     [CRTCBase],03D4h
        JE      SHORT @@UseColor
        MOV     EAX,000B0000h            ; assume monochrome
        MOV     [CRTAttrCtl],0           ; disable this access
@@UseColor:

        CALL    MemSetDescriptor,GDT_VIDEO,GDT_GDT_ALIAS, \
                EAX,[PageLength],ACC_DATA32,ATTR_32BIT

;--- see if the video buffer actually holds data
     if not, disable the video functions

        MOV     [CurPointer.Seg],GDT_VIDEO; set char pointer

        LES     ESI,[FWORD PTR CurPointer]
        MOV     AL,[ES:ESI]              ; fetch it
        MOV     AH,AL                    ; save for later
        NOT     AL                       ; flip all the bits
        MOV     [ES:ESI],AL              ; write it out
        CMP     AL,[ES:ESI]              ; see if it stuck
        MOV     [ES:ESI],AH              ; restore orginal value
        JNE     SHORT @@NoVideo          ; skip if no match
        INC     [VideoEnabled]           ; indicate that we're OK
@@NoVideo:

;--- these functions will bail out if video is disabled

        CALL    VidTurnCursorOff
        CALL    VidClearScreen,VID_DEFAULT

        RET

        ENDP    VidInitialize
```

The VGA, running in text mode, writes eight pairs of attribute and character bytes in 270 µs, or 34 µs per character. I eyeballed the code listing to come up with about 370 instructions for the complete display. That works out to about 1.4 MIPS or, inversely, 730 ns per instruction. The '386SX CPU is running at 33 MHz, implying that each instruction requires 24 clock cycles.

The Graphic LCD Interface is a bitmapped graphics device with a byte-wide data path. Each character requires eight font-table reads and, for the TLY365, 16 writes into the refresh buffer. The lower trace in Photo 2 shows those accesses blotting up a total of 2 µs, or 250 µs per char. Another eyeball count reports 2700 instructions or 1.4 MIPS again.

Bear in mind that those averages include the '386SX bus-interface overhead for 32-bit data and stack accesses through a 16-bit data path, ISA bus delays, prefetch queue flushes, DRAM refresh interference, and memory wait states. The instruction-cycle counts that you read in the manuals quietly exclude all that, giving the novice a rather optimistic view of the world.

Homework assignment: if you think this is lots worse than real mode, recode the test program to run in 16-bit real mode, make the same measurements, and report back on the BBS. My guess is that real mode will be about 10–15% faster-maybe 20 clock cycles per instruction instead of 24. Hmmm?

This code has an unusually high number of accesses to unusually slow memory. The system-board memory runs much faster than the video and LCD RAMs on the ISA bus. For extra credit on your homework: map the accesses into fake buffers in system memory and retime the code. I bet you'll pick up another 10% right there!

THE CASE OF THE CAPITAL "T"

When I wrote the TLY365 driver, my '386SX developed a curious problem. The LCD panel worked fine, but the system hung midway through the next BIOS boot sequence after I pressed the reset button. Cycling the power worked fine. It hung reliably after every manual reset. Hmmm...

A little probing showed that the CPU was stuck in a loop doing a little I/O and a lot of memory writes. It surely wasn't any of my code because I didn't have any BIOS extensions installed. Just to make sure, I pulled the battery-backed RAM out of the Firmware Development Board's socket. The CPU still got wedged.

For lack of a better idea, I pulled the Graphic LCD Interface's refresh RAM. As you might expect, the system worked perfectly even though the LCD wasn't displaying much of anything. Swapping RAM chips didn't solve the problem.

The system worked correctly with the DMF65 1 and LG64AA44D panels and the appropriate test code. The TLY365 worked OK with the Game of Life and ANSI test code from earlier this year. It failed only after displaying the test pattern in protected mode.

I modified the test pattern to write all blanks into the panel's RAM, and found that the system boot normally. A quick divide-and-conquer search showed the failure occurred when a "T" appeared in the first position of line 12. No other characters seemed to matter: "Tab stops.. ." failed just like a single "T" followed by blanks.

I whipped out my jeweler's loupe, examined the panel, then drew up Figure 1 showing the bit patterns, dot row numbers, and RAM addresses for character line 12. If you've been paying attention for the last year or so, the problem should be obvious.

This is a quiz!

OK, here's the story. Twelve lines (O-l 1) of 8 x 8 BIOS character cells puts the top bar of the T on dot row 96. Each dot row occupies 320 bytes of RAM because the TLY365 has 1280 dots on each of 100 rows, arranged in a 640 x 200 physical array. Row 1 starts at address 0000 (for reasons I covered in *CAJ* 43), which puts row 96 at address (96-l) × 320 = 76C0.

Row 97, the second row of the character cell, begins at address 7800. The dot pattern for that row begins with the tips of the T's serifs and its two-dot-wide central stroke. The hex value is 5A, which looks suspicious

Listing 4—*The Graphic LCD Interface does nof include a hardware character generator. This routine draws character glyphs into the LCD refresh buffer using the BIOS CGA font fable. Segment register FS contains the GDT_CONST selector needed to read values stored in the* _prot c onst segment *Their names start with a lowercase "c" to indicate fhaf they are not in the same segment as the usual read-write variables.*

```
        PROC     GLCDWriteGlyph
        ARG      CharValue:DWORD,CharRow:DWORD,CharCol:DWORD,     \
                 Attribute:DWORD
        LOCAL    DotRow:DWORD,DotCol:DWORD
        USES     EAX,EBX,ECX,ESI,ES

        MOVZX    EAX,[BYTE PTR CharValue]       ; can we draw it?
        CMP      EAX,[FS:cNumFontChars]
        JB       SHORT @@CharOK
        MOV      AL,'?'                         ; nope, flag the char
@@CharOK:

        IMUL     [BYTE PTR FS:cFontHeight]      ; get char offset in font
        LES      ESI,[FWORD PTR FS:cpFont]      ; pick up font base
        ADD      ESI,EAX                        ; add char offset
        MOV      ECX,[FS:cFontHeight]           ; set up row counter

        MOVZX    EAX,[BYTE PTR CharRow]           convert coordinates
        IMUL     [BYTE PTR FS:cFontHeight]             into dots
        MOV      EBX,EAX
        MOVZX    EAX,[BYTE PTR CharCol]
        IMUL     [BYTE PTR FS:cFontWidth]
        MOV      EDX,EAX
        XOR      EAX,EAX                          preload OFF dots

@@NextRow:
        MOV      AL,[ES:ESI]                    ; pick up dot row
        CALL     LCDWriteByte,EAX,EBX,EDX,[Attribute]
        INC      ESI                            ; next font line
        INC      EBX                            ; and next screen line
        LOOP     @@NextRow

        RET

        ENDP     GLCDWriteGlyph
```

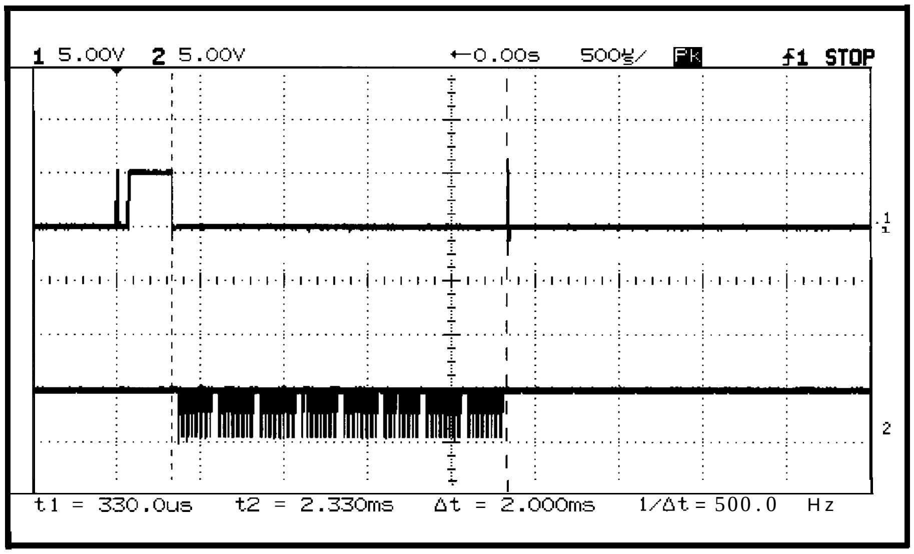


Photo 2—*This scope shot shows how rapidly the VGA and TLY365 LCD drivers write the eight-character counter value to fheir respective displays. The VGA driver is active during the second pulse on fhe upper trace; if takes about 35 µs per char. The LCD driver starts immediately after that. Drawing each row of each character using the BIOS font fable requires about 250 µs per char. Each of the eight-pulse groups in the lower trace correspond to one character. Each character has 8 rows, and each row requires 2 LCD refresh buffer writes.*

already. Recall that the TLY365 has a four-bit data interface and the Graphic LCD Interface implements blinking by alternating between the two nybbles of each refresh RAM byte.

Because the T isn't blinking, the refresh-buffer bytes at addresses 7800 and 7801 have identical nybbles: 55 AA. If that doesn't raise your hackles, you flunk..

Recall that the BIOS boot routine scans memory between C000 and EFFF for BIOS extensions. By definition, a BIOS extension must start on a 2-KB memory boundary with two flag bytes and the length of the extension in multiples of 512 bytes. Yes, the flag bytes are 55 and AA.

The second character on line 12 was either a lowercase "a" or a blank, neither of which has any dots on row 97. That translates into a pair of binary zeroes in the refresh RAM after the 55 and AA.

The test pattern's first three bytes define a BIOS extension starting at 7800 with a length of zero bytes.

Gotcha!

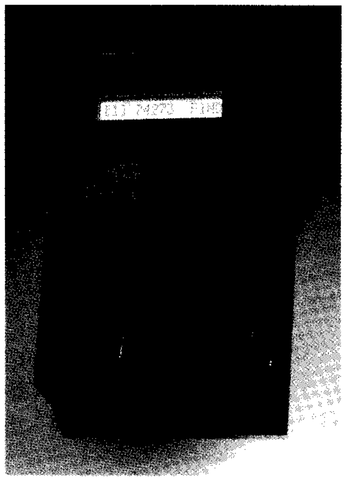
A valid BIOS extension also includes a checksum byte to make the sum of all the bytes defined by the length equal to zero. Evidently, the BIOS checksum routine in my PC concludes that a zero-length extension, lacking any content, is always valid. Remember that the refresh buffer contents after the header had no effect on whether the CPU got wedged.

So the situation goes a little something like this..

During a power-on reset the BIOS finds nothing particular in the LCD refresh RAM and boots normally. My protected-mode code displays a test pattern on the LCD panel which, quite accidentally, plunks what looks like a BIOS extension header on a 2-KB boundary within the refresh buffer.

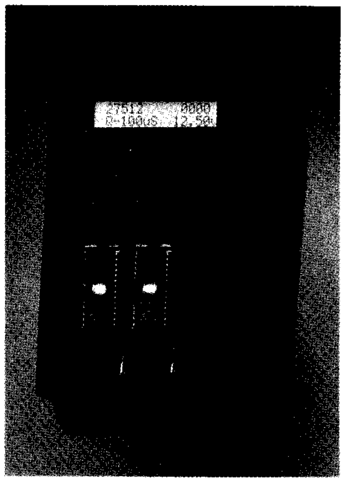
Pressing reset sends the BIOS through its extension scan again, where it finds the header at address D000: 7800. It (erroneously) concludes that the extension's checksum is valid and branches to offset 7803 in the LCD refresh buffer, which is the second zero byte. What happens after that is up for grabs. On this system, the CPU finds a loop that never ends.

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



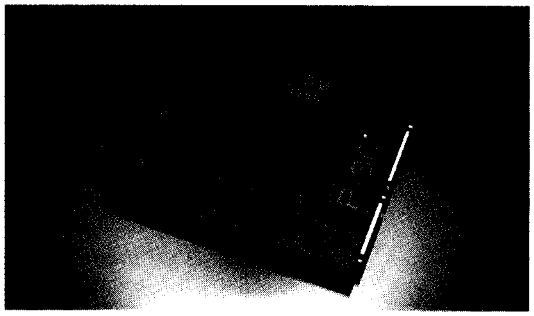
HT-28 *Express*

The HT-28 is a very convenient way of testing Logic IC's and DRAM's. Tests most TTL 74, CMOS 40/45 and DRAM's 4164-414000, 44164-441000. It can also identify unknown IC numbers on TTL 74 and CMOS 40/45 series with the 'Auto-Search" feature.
$189.95



HT- 14 *Express*

The HT-14 is one-to-one EPROM writer with a super fast programming speed that supports devices from 27328 to 27080, with eight selectable programming algorithms and six programming power (VPP) selections.
$289.95



P-300

The Data Genie P-300 is a useful device that allows you to quickly install add-on cards or to test prototype circuits for your PC externally. Without having to turn off your computer to install an add-on cards, the P-300 maintains complete protection for your motherboard via the built-in current limit fuses.
$349.95

MING
Microsystems
Division of MING C & P. INC™
17921 Rowland Street
City of Industry, CA 91748
TEL: (818) 912-7756
FAX: (8113) 912-9598

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

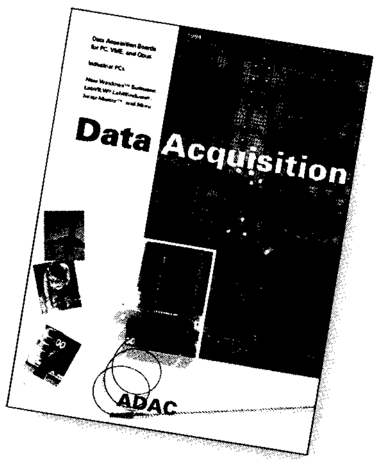
Data Genie products are backed by a full lyear limited factory warranty

©1994 MING Microsystems. All rights reserved. Data Genie logo k a registered trademark of MING Microsystems. CCDG01

#121

NEW Data Acquisition Catalog

Covers expanded low cost line.



FREE!

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

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

NEW Low Cost I/O Boards

NEW Industrial PCs

NEW Isolated Analog and Digital Industrial **I/O**

New from the inventors of plug-in data acquisition.

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

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

#122

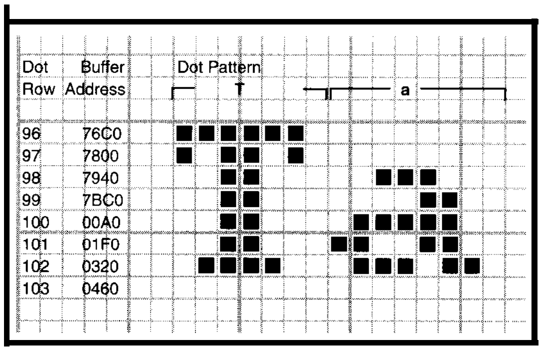


Figure 1--The LCD *test code writes "Tab stops.. " on Line 12. This figure shows the LCD refresh buffer addresses and bit patterns corresponding to the first two letters for the TLY365 640 x 200 pane/, which is electrically a 1280 x 100 panel. The Graphic LCD Interface hardware blinks by alternating the upper and lower nybbles of each byte, so every four dots on the panel require an 8-bit byte in the buffer.*

So much for real mode, hmmm!

The solution is easy enough: disable the LCD refresh RAM when the system reset line goes active so the BIOS scan cannot find a bogus extension in the buffer. The Firmware Development Board sprouted a MAX691 watchdog timer in *CAT 37,* with a latch to stretch the timeout interval after a reset. That extra guardian hardware provides the signal we need to keep the BIOS under control.

Add a wire from the -Q output of the latch (U18.8) to the RAM's -CE input (U45.20). The FFTS code sets that latch and enables the RAM chip shortly after the CPU enters protected mode. The LCD driver will activate the 8254 timer, clear the buffer, and set up the test pattern fast enough that you'll see just a blink of the previous buffer contents.

A different cure would be a (deliberate!) BIOS extension in the FDB's battery-backed RAM that gets control before the BIOS hits the refresh buffer. That extension would set up the Graphic LCD Interface for the particular panel and clear the buffer to ensure the BIOS doesn't find anything disturbing. If you've converted PMLoader to an extension, just add the requisite lines of code.

This error is an oversight, pure and simple. I knew [and so did you!) that the LCD refresh RAM contents survived a reset. It never occurred to me that the BIOS might discover an extension in the bit patterns left over from the last display!

If you build anything with dynamic bit patterns in the region where the BIOS expects extensions, take heed. You, too, may spend hard time wondering why your system doesn't boot correctly once in a while.

In a few columns, the watchdog timer will stand guard over the FFTS kernel, making that port output part of the normal startup activities. Until then, just watch the watchdog LED blink merrily along at its fast rate.

RELEASE NOTES

The code this month displays a test pattern on both a 640 x 200 Toshiba TLY365 LCD panel and an ordinary VGA. The drivers for 640 x 200 Optrex DMF65 1 and 640 x 400 Matsushita LG64AA44D panels are included; just uncomment the appropriate line in MAKE F I L E and rebuild FFTS. PMO to suit your system.

The test pattern still includes that fateful T, so you should add the wire to disable the RAM after each system reset. I suspect many BIOSs ignore zero-length extensions and reject invalid checksums. Don't take any chances. After all, a bit pattern that looks like a valid extension will occur just before your big demo.. .

The video driver should also work with a text-only monochrome adapter or an old Hercules card, but I can't test that here. If you have the appropriate hardware, give it a shot and report back on the BBS.

Next month we start '386SX multitasking and measure just how long a task switch really takes. ❑

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

I R S

413 Very Useful
414 Moderately Useful
415 Not Useful

FROM THE BENCH

Jeff Bachiochi

Engineer Seeks Personal Gardener

Assisting Vocally Challenged Vegetation

Even if you have a green thumb (which many of us don't), you still need a good memory to remember to water your plants. Leave it to Jeff to come up with an automated watering system that minimizes your memory requirements.

Corn farmers don't talk to their crops. But, many green thumbers do talk to their plants. Some even have names for them.

Me-I like plants. But, I don't know the needs of each variety and, in ignorance, I often neglect them—sometimes beyond the point of no return. You could call it murder in the third degree (or perhaps "plantslaughter"?).

When the cat starts pacing figure-eights about my feet, I know it's looking for food or water. When my six-year-old, Kristafer, greets me at the door with "What's for supper?" instead of "Hi daddy! ", there is no doubt where the priority is. Occasionally, my stomach will rumble a bit if I've skipped lunch. These are signs I can interpret as a need for sustenance.

On the other hand, by the time plants show signs of neglect, it's often too late. Root systems have shriveled. Leaves are turning (and I don't mean fall colors). In general, they are experiencing total body (cellular) collapse. If only plants could gasp out a message when in need.

My first thought was some kind of biomonitor, but that would require a different monitoring sensor for each type of plant. Certainly, monitoring an ivy leaf would be different from a cactus appendage.

And, what about interpreting the differing dialects of each of the plant species? Not something I want to devote my life to.

HELP I'M DROWNING

My real problem is under or over watering. By the time I do remember to water my plants, they are bone dry and I over compensate by giving them too much.

So, let's try to monitor the moisture content of the soil and sound an alarm if it becomes too dry. I can then respond with a small sip of that mineral-collecting essence, H_2O. Also, I can add a bit of fertilizer to the water in an attempt to restock the depleted soil.

There are many ways to measure the moisture content of the soil. One of the least expensive is to measure the conductivity of the soil. The presence of moisture within the soil

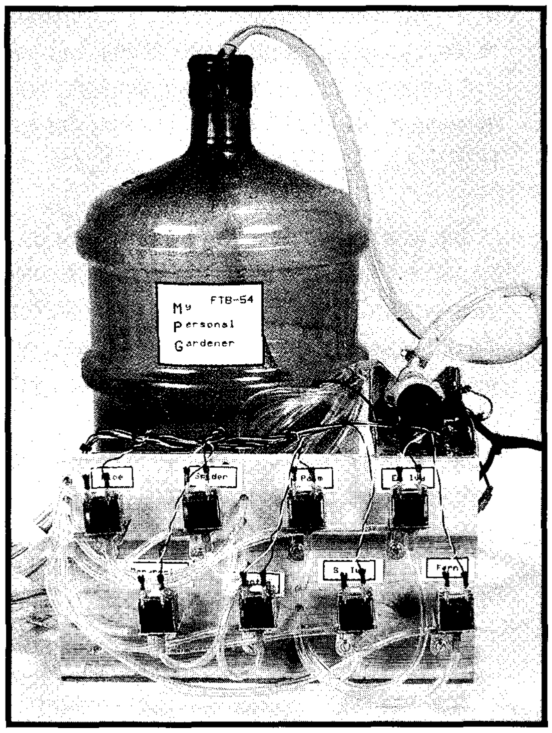


Photo 1—*Able to handle up to* eight plants, *the Persona/ Gardener tracks soil moisture and automatically delivers wafer to parched plank.*

permits current to flow through it. Resistance increases as moisture content decreases.

We can take advantage of this change in resistance by using it to indicate the relative moisture content of the soil. All we have to do is compare the moisture content to a set point we have predetermined to be too dry. Then, we sound an alarm when the two points become equal.

Two electrodes are inserted into the soil enabling it to act as a variable resistor. To eliminate the potential plating problems and charge build-up associated with exposure to direct current, I use an AC source.

Any small-signal source can be used, but I chose a '555. This output is fed into one electrode. The other electrode becomes the input to an amplifier and filter (detector) which will, at some input, switch on an open-collector output capable of driving an audio transducer. (An inexpensive op-amp and transistor driver may come to mind.) If the oscillator frequency (of the '555) is chosen for the peak output of the transducer, the circuit would be tuned for optimal response without the necessity of a separate piezo-driver circuit.

AUTOMOTIVE HERO

Allegro Microsystems has prepackaged all the necessary circuitry to perform fluid detection in a single

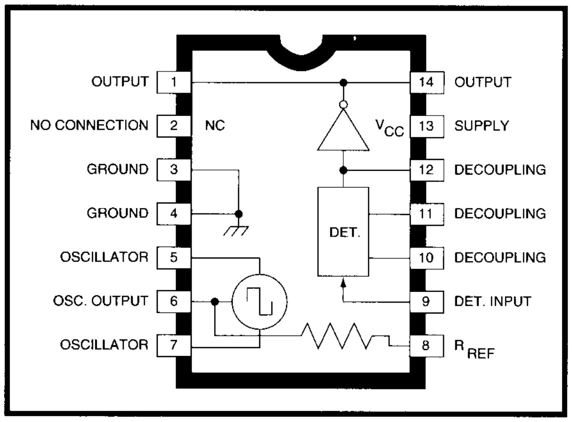


Figure 1—*The Allegro Microsystems ULN2429A is specifically designed to sense conductive fluids.*

chip: the ULN2429A. Originally designed for the automotive industry, the '2429 performs well in harsh environments. It has reverse-voltage protection, internal regulation, and temperature compensation built into the design. See Figures 1 and 2 for the innards of this 14-pin DIP device.

Although designed specifically for conducting fluids, the '2429 can detect other nonconductive fluids by replacing the probes with other variable-resistance elements. This might be a photoconductive cell, rotary or linear position sensor, or thermistor.

External parts count is minimal—only a couple of small capacitors and possibly a resistor (I say "possibly" because of the '2429's flexibility). The chip can be used to alarm on either the presence or absence of the conducting fluid.

When used to detect fluid (as shown in Figure 3), the oscillator is coupled through the fluid to the detector. A good path through the fluid provides detection. The absence of fluid disconnects the path and as such, inhibits detection.

When used to detect the absence of fluid (as shown in Figure 4), the oscillator is coupled directly to the detector. The absence of fluid does nothing to inhibit detection. However, the detector's input is shorted to ground by the presence of fluid, inhibiting detection.

SWIFT FLOWING CURRENTS

The circuit draws 5 mA when there is sufficient moisture to keep it happy. The level of moisture depends on the kind of soil you are measuring as well as the conductivity of the water you are applying.

We can adjust these variables with a change in the resistance used to couple (or short out) the oscillator to the detector. It is assumed that once the adjustment is made for a particular soil and water makeup, you will not be changing the soil's composition, nor the source for the water. If you do, you may be required to readjust the trimmer.

Even at 5 mA, a cheapo 9-V battery will only last a day. This doesn't say much for battery technology. But even so, when plant watering requires 24-h surveillance, battery power is not practical. You could stop

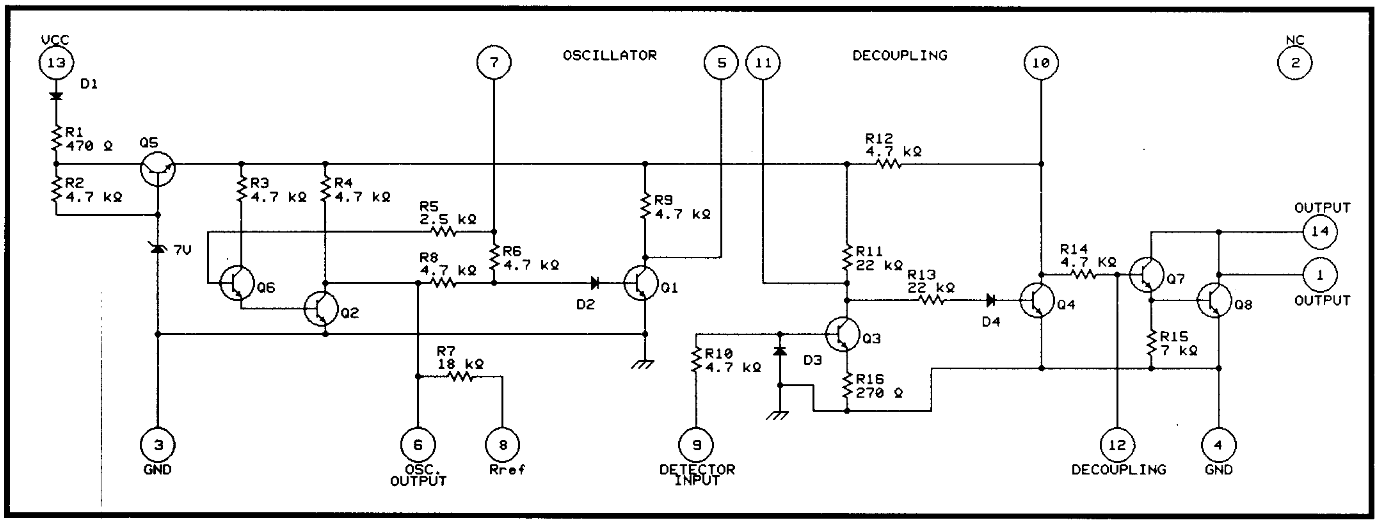
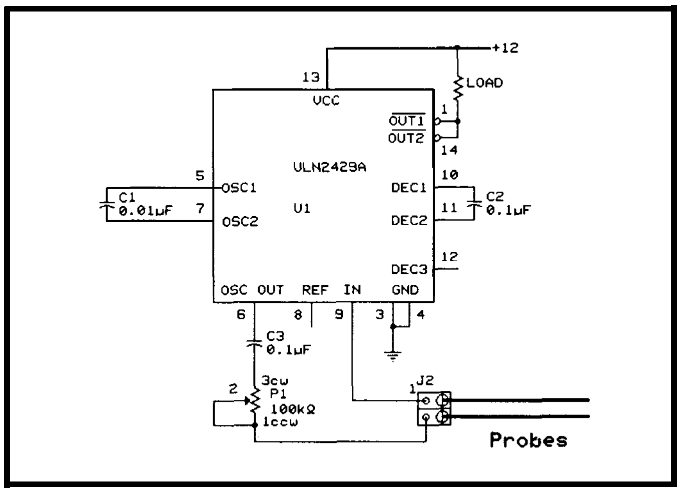


Figure 2—*The oscillator output of the chip is coupled to its detector input through the fluid being monitored. The final output indicates the presence or absence of the fluid.*



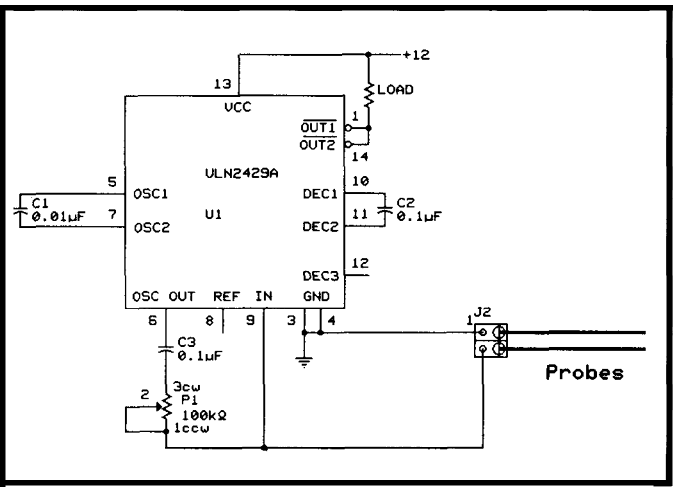


Figure 3—*In a fluidpresence defector, the probe's conduction path couples the signal, raising the output (no alarm). With loss of the conductance path, the loss of the signal lowers the output and sounds the a/arm.*

Figure 4-/n *a fluid absence detector, the probe's conduction path shorts the input signal to ground, raising the output (no alarm). With loss of conductance path, the signal lowers the output and sounds an alarm.*

here and put a push-button switch between the battery and the circuit. It would require a push of the button to test the soil, but it will still only last for the shelf life of the battery.

I want this to be on guard 24 hours or at least autonomously. To achieve this, the circuit must be externally powered not just for the 5 mA, but for the current needed by the solenoid replacing the beeper.

The replacement of the beeper with the solenoid gives me a closed-loop system. Water will be supplied to the plant whenever the fluid detection drops below the alarm threshold. The hysteresis is automatic, but can be controlled by the speed at which water is metered into the plant's container. Rapid flow might put too much water on the plant before the fluid detector could shut off the supply solenoid. A slower flow sustains a more uniform moisture level. You may wish to water from above or below the soil's surface depending on the type of plant or the container.

C COMPILER FOR PIC CONTROLLERS

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

```
#include <PIC16C56.h>
Ruse  Delay(Clock=20000000)
#use  RS232(Baud=9600,Xmit=pin_1,RCV=pin_2)

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

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

CCS, PO Box 11191, Milwaukee WI 53211
414-78 l-2794 x30

#123

TIRED OF WAITING FOR THE PROMPT ?

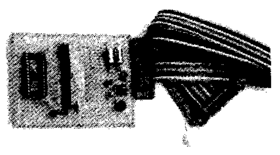
Speed up with a ROM DRIVE! Boots DOS and programs instantly. Also used to replace mechanical drive completely in controllers on diskless workstations. The only perfect protection from viruses. Easy to install half-size card.

DOS IN ROM!

MVDISK1	128k	575
MVDISK2	**720k**	**5150**
MVDISK3	2 88m	5195

$75

Quantity discounts!



$95 EPROM PROGRAMMER

- Super Fast Programming
- Easer to use than others
- Does 2764/27080 (8 Meg)

WORLDS SMALLEST PC !!!

ROBOTS ALARMS RECORDERS DOS

8088 SINGLE BOARD COMPUTER

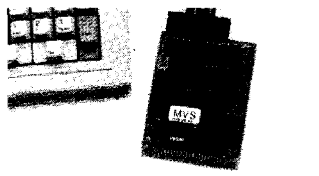
THREE EASY STEPS:
1. Develop on PC
2. Download to SBC
3. Burn into EPROM

$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



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

#124

Figure 5 depicts a convenient, single-plant, gravity-fed system which can keep the green stuff happy without daily attention even for a sustained vacation. Notice the water container (a 2-l pop bottle] is hung IV style between the hanger and the plant. I used two pop-bottle bottoms (the old style bottle with the added cup or foot since the new ones are one molded part] to capture the bottle and hold it upside down.

The lower cup's center is removed allowing the bottle's neck to hang through it. A small hole is needed in the bottle bottom to let air in as the water drains from the bottle's mouth. A small cork with a "T" (fashioned from a bobby pin or paper clip) poked into it (like a squirt gun's plug) keeps the hole closed when filling the bottle and doesn't get lost.

MOISTURE CONTENT VERSUS SOIL **RESISTIVITY**

To set the moisture and alarm levels, we need some kind of reference. Let's look at a way to develop a

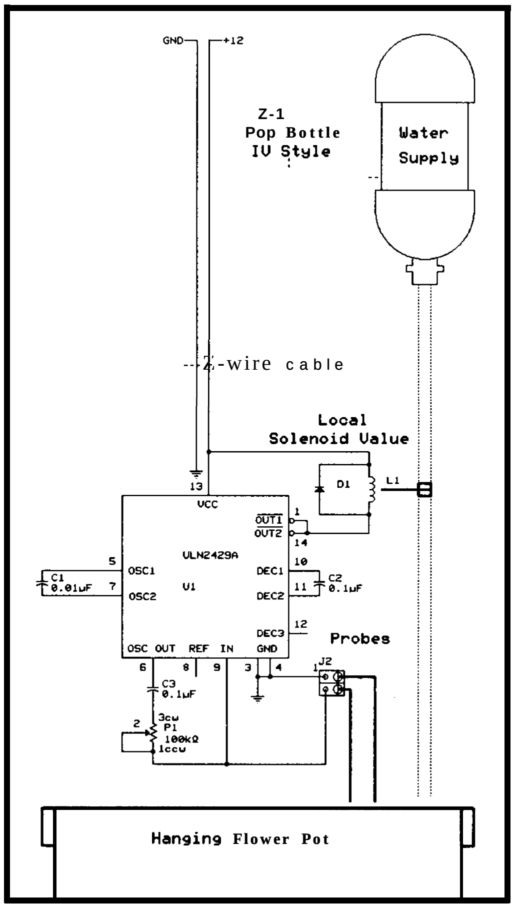


Figure **5**—*By* using a *ULN2429A for each plant plus a common wafer reservoir, the system controls a solenoid to wafer the* ***plant*** *when necessary.* ***Potentimeter P1*** *is used* ***to adjust*** *how dry the soil must be before watering commences.*

breakdown between types of soil, the relative volume of water they contain, and the ohmeric values produced by the water content of each soil composition.

To start, three equal-sized containers are filled with equal volumes of different plant media. In the first is a desert mixture, good for growing plants of the succulent variety like cactus. It is gritty and low in organic matter. The second is a meadow mixture with the high-humus content preferred by many plants including the ivy and fern families. In the third container is a swamp mixture, which is peat-moss rich to

PLIX™ HARDWARE X-10" TRANSCEIVER CHIP

Micromint introduces its new Power Line Interface for X-10. PLIX is an 18-pin ASIC chip that automatically handles all the specialized X-10 timing and bit-shuffling between a computer and a TW523 power line module.

✖ Complete interface between parallel port and TW523 or PL513 modules
✖ Performs all X-10 transmit and receive functions
✖ Detects AC power loss
✖ Simple interface and timing — operates even in interpreter BASIC
✖ Low power — only 1.8 mA @ 5 V

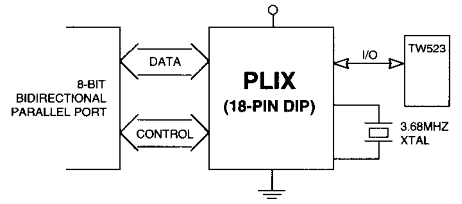
PLIX chip and data sheet **$20**
100 qty. OEM **$12**
PLUS SHIPPING

Call **1-800-635-3355**
or Write for a PLIX data sheet

PLIX EVALUATION BOARD

To speed your evaluation, we have produced a low-cost PLIX interface that plugs into a parallel printer port. The PLIX-EKit includes a PLIX chip and data sheet, PCB with all components, application note, and PC-compatible software on diskette.

Add $30 for TW523 X-10 transceiver module **$39.00**
PLUS SHIPPING



M I C R O M I N T, I N C.
4 PARR ST., VERNON, CT 06066
(203) 871-6170 FAX: **(203) 872-2204**

X-10 is a trademark of X-10(USA), Inc.

#119

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 I 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

#125

188SBC

Use Turbo or MS 'C
Intel 80C188XL
Two 1 meg Flash/ ROM sockets
Four battery backed, 1 meg RAM
16 channel, 12 or 16 bit A/D
8 channel, 12 bit D/A
2 RS-232/485 serial, 1 parallel
24 bits of opto rack compatible I/O
20 bits of digital I/0
Real-time clock
Interrupt and DMA controller
8 bit, PC/104 expansion ISA bus
Power on the quiet, 4 layer board is provided by a switcher with watchdog and power fail Interrupt circuitry.
The 188SBC is available with Extended Interface Emulation of I/O - a Xilinx Field Programmable Gate Array and a breadboard area. Define and design nearly any extra Interface you need - we'll help! 188SBC prices start at $299.
Call right now for a brochure!

552SBC

The BOC552 is an 8051 with:

8 ch. 10 bit A/D	**2 PWM outputs**
Cap/cmp registers	**16 I/O lines**
RS-232 port	**Watchdog**

We've made the **552SBC** by adding:
2-RS-232/485 multi-drop ports

24 more I/O	**Real-time Clock**
EEPROM	**3-RAM & 1-ROM**
Battery Backup	**Power Regulation**
Power Fail Int.	**Expansion Bus**

Start with the Development Board all the peripherals, power supply, manual and a debug monitor for only $949. Download your code and debug it right on this SBC. Then use OEM boards from $149.

True Low-cost In-circuit Emulation

The DryICE Plus is a low-cost alternative to conventional ICE products. Load, single step, Interrogate, disasm, execute to breakpoint. Only $449 with a pod. For the 8051 family, including Philips and Siemens derivatives. Call for brochure!

8031SBC as **low** as $49

Call for your custom product needs. Quick Response.

VISA MasterCard

HTE
HiTech Equipment Corp.
9400 Activity Road
San Diego, CA 92126
[Fax: (619) 530-1458]
Since 1983
— (619) 566-1 892 —
70662.1241 @ compuserve.com

#126

increase the drainage and aeration favored by palms and peperomia (I suddenly have an uncontrollable urge for pizza).

Prior to any measurements, the containers, each holding a different mixture, are allowed to dry thoroughly by exposing them to a dry heat source for three days. Two stainless steel probes are implanted (sorry) into each container to serve as measurement contacts. Then, systematically, an equal and measured amount of water is added to each container. Each container is measured after a short waiting period to give the water time to permeate throughout the soil.

The measurement process is repeated until the saturation point is reached and water leaks out the bottom of the container. The graph in Figure 6 shows the results of the tests.

MULTIPATIENT CARE

This all started with Elaine's rejected fern plant I fished out of the trash at the office. When it began to show signs of improvement, I brought in another plant from home. Then, I rooted cuttings that Rose and Dottie (two other coworkers) had given me. Soon, I was up to eight plants.

At this point, the system has to change. One missed watering cycle leads to irreparable damage in some, and an overall gloom in the atmosphere. This is not the effect I was after. Having healthy plants indoors helps camouflage an office environment. (If I wanted death and destruction, I would watch the evening news.)

To prevent my office from looking like an infirmary with IV bottles suspended all over, I created one reservoir for all the plants. A 5-gal water-cooler bottle serves as a spring to all ye wishing a drink. Each plant receives water through its own umbilical cord consisting of a ⅛″, clear plastic tube and two conductors. The two wires bring the probe leads to and from the plant. At the far end of the umbilical, the circuit senses conductivity and enables a solenoid valve. Once enabled, the solenoid releases the nutrients for thirsty vegetation.

Eight such subsystems come together and are fed from the same source. Once primed, a gravity feed is all that's necessary as long as the water source remains above all the plants. Since most of my office plants are hanging, this becomes a problem. The solution is to add a small water pump which is enabled whenever any of the subsystems request water (see Photos 1 and 2).

The pump was a great find. It has a 120-V, fan-cooled motor driving a bellows-type pump. A small microswitch is cammed to operate once per stroke. The volume is adjusted by changing the length of the bellows' drive arm, and the intake and output lines fit into ½″ plastic tubing.

I made a manifold from short lengths of ⅛″ copper tubing I had left over from a previous project. I gathered eight pieces in a group and soldered the openings between the tubes. Once cooled, I bent the free ends away from one another so they would accept the plastic ⅛″ IV tubing going to each solenoid. I coated the other end with Goop (a trade name, believe it or not) and forced it into a short length of ½″ plastic tubing. A small relay turns on the pump whenever any of the eight sensors request nutrition.

Since all alarm inputs and control outputs are now local, I could add some intelligence to the system. Using a microcontroller, I could log and retain a history about the use of water on a plant-by-plant basis. If incessant watering was requested beyond a reasonable average, the microprocessor could command the flow to be halted, thereby preventing the catastrophe of drowning more than the plants.

The same sensors might be used outside the home to request the automatic sprinkling of the lawn when watering was necessary instead of on a scheduled basis. This refinement would help conserve water. (Usually, consumer devices of this kind cost around the $50.) It is also a good application for HCS, but would be a bit of overkill for me here at the office.

Now, I can forget about watering entirely. My plants will live on, even when I'm not around. I just have to remember to check the water reservoir once in a while. Maybe if I place a ninth sensor in the reservoir and.. ❑

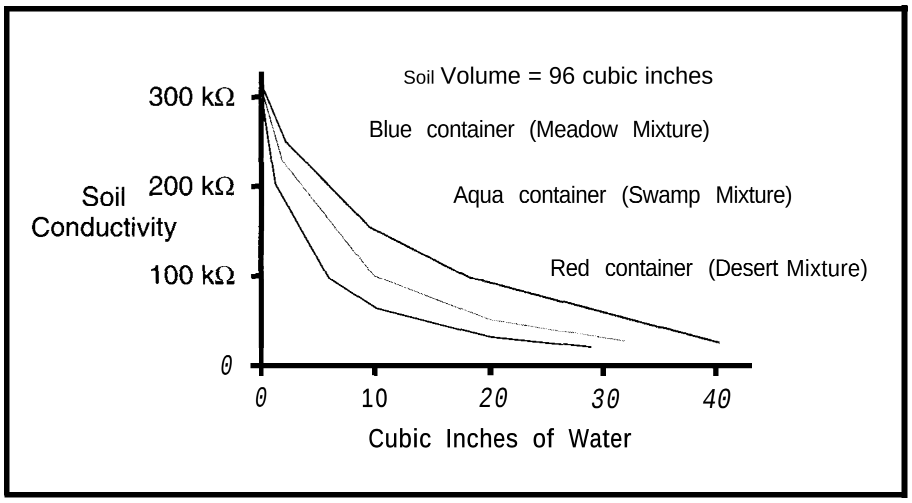


Figure 6—*Different kinds of soil have different wafer-retention characteristics, so the watering **set** point on each plant (and each kind of soil) must be individually adjusted*

*Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on the Computer Applications Journal's engineering **staff**. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circellar.com.*

CC

Sensor ULN2429A
Allegro Microsystems, Inc.
115 Northeast Cutoff
P.O. Box 15036
Worchester, MA 016 15
(508) 853-5000
Fax: (508) 853-7861

Solenoid G42,533
Edmund Scientific Company
101 E. Gloucester Pike
Barrington, NJ 08007-1380
(609) 573-6250
Fax: (609) 573-6295

Pump #35-139
Megatronics, Inc.
2700 Sunset Blvd.
Steubenville, OH 43952
(614) 266-2223

I R S

416 Very Useful
417 Moderately Useful
418 Not Useful

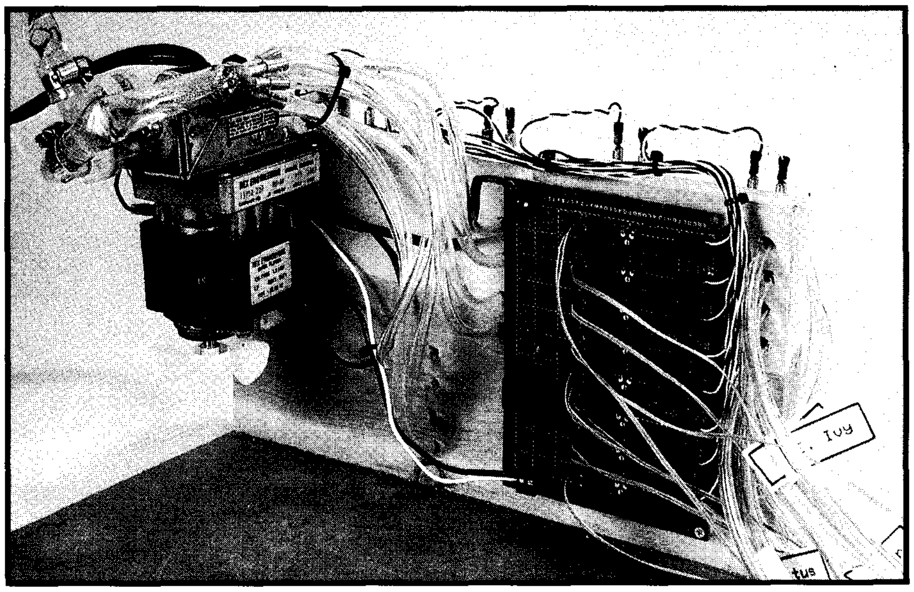
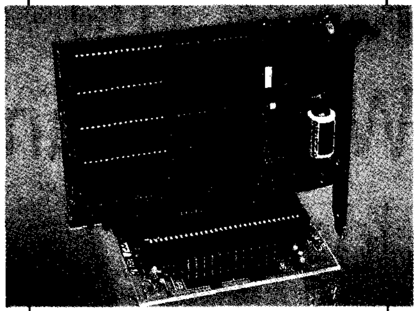


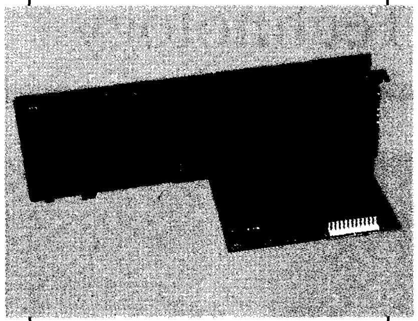
Photo 2-*Since many of the plants being monitored and watered are hanging higher than the wafer reservoir, a small water pump pushes the wafer up to the plants.*

VMAX®

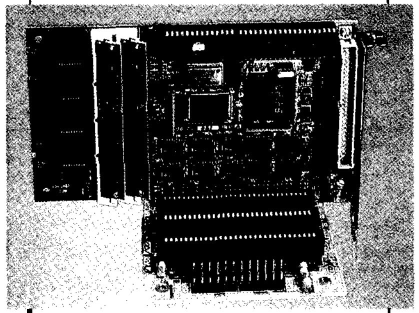
**QUALITY PRODUCTS
RESPONSIVE SERVICE
RELIABLE DELIVERY**



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/SRAM 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.

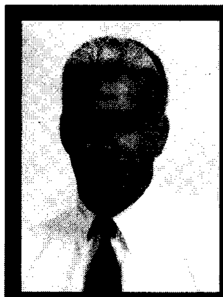
TEMPUSTECH, INC.
TEL: (800) 634-0701
FAX: (813) 643-4981

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

#127

Do You Know The Way To San Jose?

Precise Navigation Technology



For fear of gettin' right turned around, Tom has gone out and found the TCM1, a precise compass which interfaces with your favorite computer using an RS-232 port.

SILICON UPDATE

Tom Cantrell

besides being the name of a song by Dionne Warwick, it's a question I often ask myself, even after 15 years in Silicon Valley.

You see, Silicon Valley is actually composed of dozens of small communities ringing the southern portion of San Francisco Bay. Alternately named after saints (Jose, Clara, Teresa, Mateo, not to mention, the well-known Francisco] or geographic features (Blossom Hill, Mountain View, Sunnyvale, Redwood City), it's a hodgepodge of tract homes, defense contractors, universities, malls, and of course our beloved chip factories.

However, despite the homogenizing effects of suburban sprawl, the original balkanization can still be seen in the road system. Unlike the rational, chip-like grid system you might expect, this nether-world is a fractured network of highways and byways meandering hither and yon.

No wonder Dionne got lost-what with streets that change name half a dozen times as they meander across the valley floor. Perhaps the most troublesome aspect of the whole mess is a kind of cardinal fuzzy logic that makes giving directions a real challenge.

Heading north or south on Highway 101, you'll encounter many exits in which you are given a choice of heading-you guessed it-north or south. Choose wrong and, as the other old song goes, you may be stuck in Lodi again.

The problem is the highway is really headed SE-NW with exits heading SW-NE. Assuming a two-letter designation beyond the capabilities of your average commutoid, the road bosses, not fazed by ending up with two northbound routes that remarkably intersect at right angles, turn both NW and NE to N.

My wife happily gets around using a kind of "rip up and retry" routing scheme. (Another of her other handy driving tips includes "don't use the

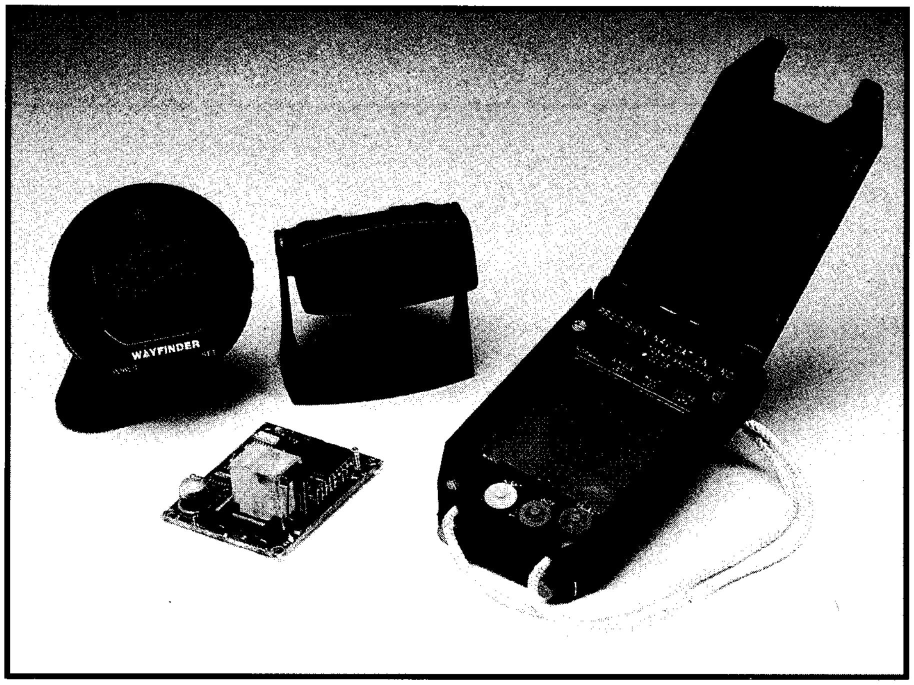


Photo I-Precision Technology's compass *offerings include the* military *TC3M3 (right), the civilian Wayfinder car compass (left), and the TCM1 compass with computer interface (bottom left).*

brakes or they might wear out.“) I, being the more logical sort, rely on a compass to keep things straight.

Which brings up the subject of this article. If you’re involved with applications such as navigation, robotics, or even virtual reality, you’ll be pleased to note the arrival of computer-savvy compass technology from the aptly named Precision Navigation.

NEEDLE THROUGH A CORK—NOT!

Thanks to the fact that the earth has a magnetic field, wayfarers and wanderers have been able to make their way with compass technology that has changed little over the centuries. However, once you delve a little deeper into the subject, you’ll find that needle-through-the-cork technology is by no means precise.

First of all, the magnetic field isn’t exactly aligned with the earth’s spin axis, which leads to a distinction between “magnetic” north and the “true” north. The difference between the two-known as declination-is roughly 11°, but varies with geographic location.

The magnetic field is best represented as a three-dimensional vector (X, Y, and Z) whose vertical component (Z), known as inclination, increases to 90° as the poles are approached. It’s easy to see how this can diminish compass effectiveness (i.e., increase susceptibility to interference) as the poles are approached since the needle is essentially being pulled down rather than forward.

Worse, this problem dictates that the compass must be exactly level for accurate results since each degree of uncorrected tilt can cause up to two degrees of heading error. To correct this, compasses are always placed in gimbals that are made up of one form (U-joints, pendulums; floats) or another of mechanical leveling.

The earth’s magnetic field is also subject to distortion by a variety of natural phenomena. On a planetary level, the solar wind causes shifts over time while, on a smaller scale, a compass is easily affected by proximity to local sources of magnetic interference. Forces such as an electric current or magnetically generated constant interference—proximity to other ferrous materials can cause inconsistent magnetic distortion.

It’s possible to calibrate for these local interferences, but note the catch-22 imposed by the gimbal. Yeah, the compass stays level, even though the surrounding (and interfering) materials shift in their relationship (i.e., the car going up or down a hill). A compass calibrated against local interference at one incline may exhibit large heading errors as the incline changes.

All these effects are easily seen in my car compass. I know mine shifts a healthy 20° or so as soon as I turn the key. It’s still generally good enough to figure out what the road bosses meant, and after all, the stakes aren’t all that high (heck, Lodi is kind of quaint).

Given the disparaging tone of the old saying “It’s good enough for government work,” it’s ironic that the government-specifically the military-has funded the development of more precise and reliable electronic compass technology. Here, the stakes are much higher, ranging from the career-limiting (getting lost in the woods) to the lethal (when calling in air or artillery).

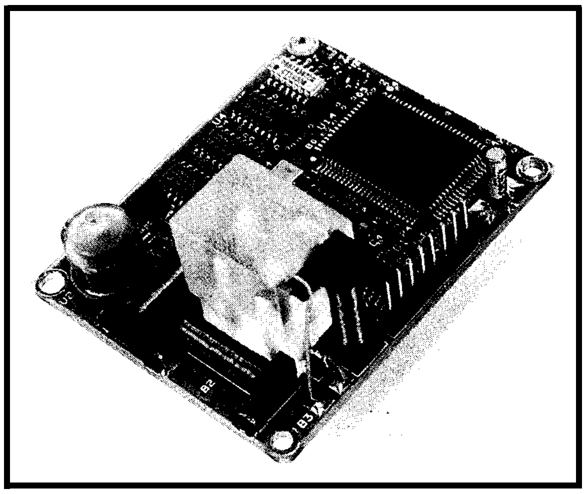
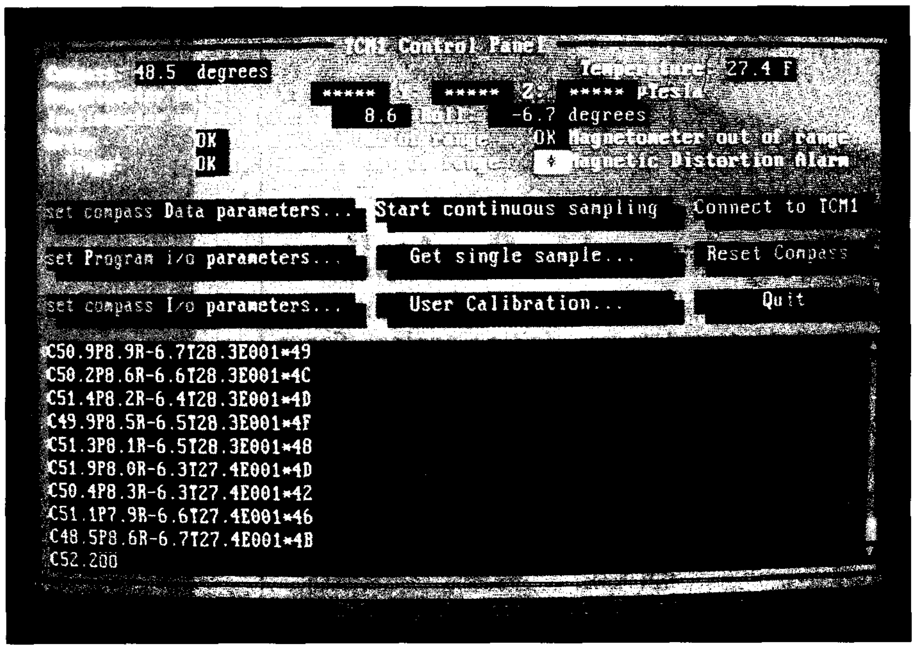


Photo 2—*The TCM1 compass uses three magnetometers (kept in alignment by the* plexiglass block) to sense magnetic direction, a circular inclinometer to *sense* board *tilt (far left), and a temperature sensor (just above the inclinometer). A Mitsubishi single-chip processor coordinates everything.*

INDUCTIVE LOGIC

Indeed, it was at the military’s request that Precision Navigation first developed their electronic compass technology in the form of the TC3M3.

This is a battlefield-hardened unit with a heavy-duty aluminum case and LCD shield, a marching-pace counter (for tracking distance), and nightvision navigation via low-intensity focused


TCM1 Control Panel
48.5 degrees
Temperature: 27.4 F
***** Y: ***** Z: ***** µTesla
8.6 Roll: -6.7 degrees
OK
OK
OK Magnetometer out of range
Magnetic Distortion Alarm
set compass Data parameters...
Start continuous sampling
Connect to TCM1
set Program i/o parameters...
Get single sample...
Reset Compass
set compass I/o parameters...
User Calibration...
Quit

```
C50.9P8.9R-6.7T28.3E001*49
C50.2P8.6R-6.6T28.3E001*4C
C51.4P8.2R-6.4T28.3E001*4D
C49.9P8.5R-6.5T28.3E001*4F
C51.3P8.1R-6.5T28.3E001*48
C51.9P8.0R-6.3T27.4E001*4D
C50.4P8.3R-6.3T27.4E001*42
C51.1P7.9R-6.6T27.4E001*46
C48.5P8.6R-6.7T27.4E001*4B
C52.200
```

Photo 3—*The TCM1 comes with PC-based software to provide calibration and evaluation of the board.*

LEDs recessed into the compass body. Marching orders can be entered as a sequence of headings and distances, and the TC3M3 will steer you to your destination.

Needless to say, the TC3M3 is kind of price/performance overkill when it comes to finding San Jose. So, Precision Navigation commercialized their technology in the form of the Wayfinder car compass (the TC3M3 and Wayfinder are shown in Photo 1).

At less than $100, the Wayfinder clearly can't offer military accuracy (e.g., ±9° vs. ±1°), temperature range (e.g., -10 to 60°C vs. -20 to +70°C), sampling rate (5 Hz vs. 8 Hz), and so on. Nevertheless, underneath their respective aluminum and plastic cases, both compasses are based on the same key concepts.

I'm no expert, but to my understanding the first electronic compasses (built in the '30s) were based on flux-gate technology which saturates a magnetometer with a high AC current. The second harmonics are amplified and signal conditioned. Furthermore, using the technology in a digital application requires an ADC, making the flux gate an altogether expensive, power hungry, and noise-sensitive approach.

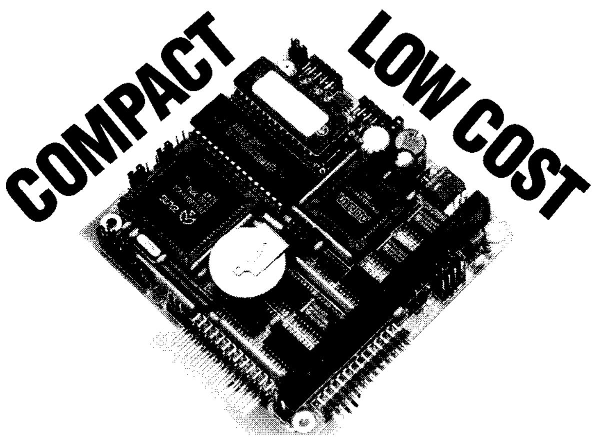
By contrast, modern magneto-inductive techniques use a low-power LR (inductive and resistive) oscillator circuit in which the inductance and thus, frequency of oscillation, vary with heading. The essentially digital output clock is easily tracked by a microcontroller, dispatching the need for an A/D converter.

Thanks to the low cost and power of the magnetoinductive sensor technology, it's feasible to provide full three-axis (two is more typical) capability. The notable result of this decision is that the compass, knowing the relationship between horizontal (X,Y) and vertical (Z) components of the field, can automatically detect transient distortions and thus warn the user that the current headings are suspect. Though automatically compensating for such disturbances is a challenge that remains unsolved, a warning beats having a compass you can never really trust. In a sense, it's like handling memory errors-ECC is best, but parity checking is better than nothing.

Another innovation is built-in electronic leveling which addresses the tilt and calibration concerns. Using an inclinometer yielding very accurate (0.2") pitch and roll, the compass can automatically correct for tilt. Further, the compass body itself can be physically strapped down which, unlike a gimbaled approach, maintains a fixed-and thus cancellable—interference relationship, even when the equipment tilts.

HIGHWAY DIRECTIONS

The TCM1 (Photo 2) is the latest addition to Precision Navigation's lineup and is targeted towards micro-based applications that need a little direction. It offers compass specs (accuracy, resolution, sampling rate, etc.) nearly matching those of the military T3CM3 while replacing the hand-held unit's LCD, LED, and



C-Programmable Controllers

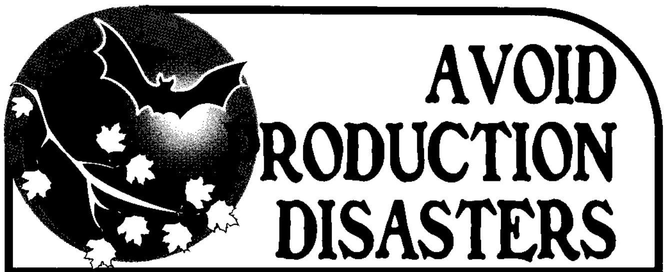
Use our controller as the brains of your next control, test or data acquisition project. From $149 qty one,. Features I/O to 400 lines, ADC, DAC, RS232/RS485, printer port, battery-backed clock and RAM, keypads, LCDs, enclosures and more! Our simple, yet powerful, Dynamic C™ makes programming a snap!

24-Hr AutoFAX: 916.753.0618. Call from *your* FAX. Request catalog 18.

1724 Picasso
Davis, CA 95616
916.757.3737
916.753.5141 FAX

Z WORLD ENGINEERING

#128



AVOID PRODUCTION DISASTERS

Before you send the next product design directly from engineering to production, have it independently reviewed. Because of our wide spectrum of experience, we specialize in catching the gottchas that aren't on the data sheets. If you already know that fine tuning a design is really a "black art", then you should also know that Steve Ciarcia and the Ciarcia Design Works team are there to spread the spell.

Fax us and let's discuss some real magic.

Fax (203) 871-8986

switch-based user interface with an RS-232 port for connection to your favorite computer. Looking at the photo, notice the three (i.e., 3-axis) magnetometers; the circular, liquid-filled inclinometer; and the Mitsubishi single-chip CPU (large chip) that runs the whole show.

Installation of the TCM1 is pretty straightforward. The user manual points out the obvious-the unit should be located away from sources of magnetic interference (transformers, motors, etc.). Otherwise, it's a simple matter of aligning the compass with your equipment (the alignment references are the mounting holes, not the board edges-see Figure 1) and leveling it as much as possible (to maximize the tilt range which is ±25°). The electrical connection is via 8-pin header (a pinout is shown in Figure 2) and is equally simple.

Power (≈10 mA) is deliverable either as a regulated 5 V (pin 1) or unregulated 6-25 V (pin 2) with associated ground (pin 3). RxD and TxD (pins 4,5) are the familiar RS-232 lines with fixed format (8N1) operating from 300 to 19.2 kbps (9600 is default). Note the separate data ground on pin 7 (from power-supply ground) that connects to pin 7 of a DB-25 RS-232 cable.

Pins 6 and 8 provide heading output for those who insist on doing things the hard(ware) way. The oddly named N+1 (pin 6) is a PWM output (i.e., duty cycle = heading) while the analog-output signal (pin 8) is just that. The latter is software programmable either as a linear (i.e., O-2.5 V, O-359" in 1.4" increments) or quadrature output (sine and cosine of heading).

Actually, neither the manual nor apps engineers I talked to go into great detail on these pins, because they typically aren't used. Besides the fact that the pins aren't very computer literate (i.e., need ADC, etc.), they only deliver heading information.

The RS-232 line supports the delivery of a lot of other neat information including pitch, roll, temperature, magnetic-distortion alarm, and various out-of-range conditions. The RS-232 port (with input capability) is also the only way to issue configuration and calibration commands to the TCM1 during installation.

You may question (as I did) the inclusion of a temperature sensor. It's not simply a case of "everything but the kitchen sink" marketing, though a temp sensor is often handy. Rather, it turns out that the liquid-filled inclinometer is affected by temperature. So, the compass needs to know the

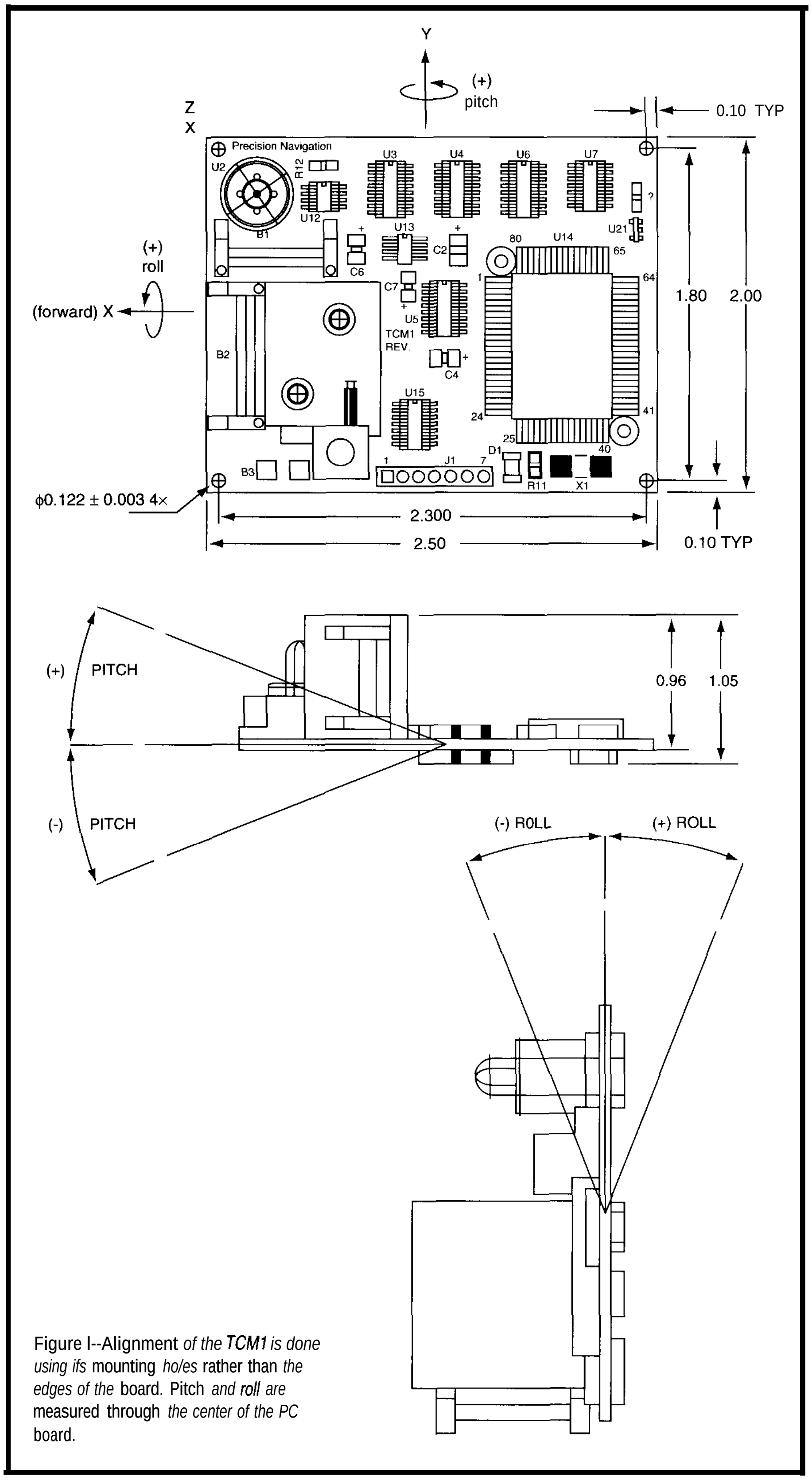


Figure I--Alignment *of the TCM1 is done using ifs* mounting *ho/es* rather than *the edges of the* board. Pitch *and roll are* measured through *the center of the PC* board.

temperature to determine the tilt to come up with the heading.

Anytime a smart gizmo uses an RS-232 port for host communication, I recommend adopting-as the TCM1 does-an ASCII-based command and response scheme. This makes it a snap to connect a terminal or PC with comm software and start tapping away. Further, experimentation and testing for both the device and host-driver software are eased in a WYSIWYG manner. Think twice before adopting a binary protocol—only do so if speed is a must.

The only gotcha that might cause a little head scratching is that the TCM1 defaults to a nonecho mode. You either have to enable a local echo at your terminal or configure the TCM1 to echo (with Ctrl-E) if you want to see what you type.

Once connected, you'll find the TCM1 monitor offers dozens of commands roughly classified into groups as either configuration or sampling.

The configuration commands, besides handling a few mundane matters like specifying baud rate, largely serve to define the format of the TCM1 output once sampling starts.

I should note that the TCM1 supports an arcane National Maritime Electronics Association (NMEA) 0183 format which, as best I can tell, is kind of an infohighway for the yachting set (i.e., connects GPS, compass, radar, etc.). Like the extra output pins, this option is flawed (fatally in my opinion) by being limited to heading info.

So, assuming you're using the more fulfilling TCM1 format, each sample (up to five times per second, but optionally less often) outputs an ASCII string (terminated by an XOR checksum and <CR><LF>) that encodes some or all of the following items: heading, pitch, roll, (raw) magnetometer output, and temperature. From time to time, you may also receive an error message warning of a magnetic or temperature distortion and incline or magnetometer out of range.

The TCM1 is rather smart when it comes to massaging the output. For instance, heading, pitch, and roll can be requested as degrees (360 per circle) or mils (6400 per circle). You're even given the choice of referencing the heading against the true or magnetic north. Temperature can be either °F or "C, and so on. Sure, your driver software could handle such conversions, but the more stuff the TCM1 does, the less software you have to write (and test, and debug, and maintain...).

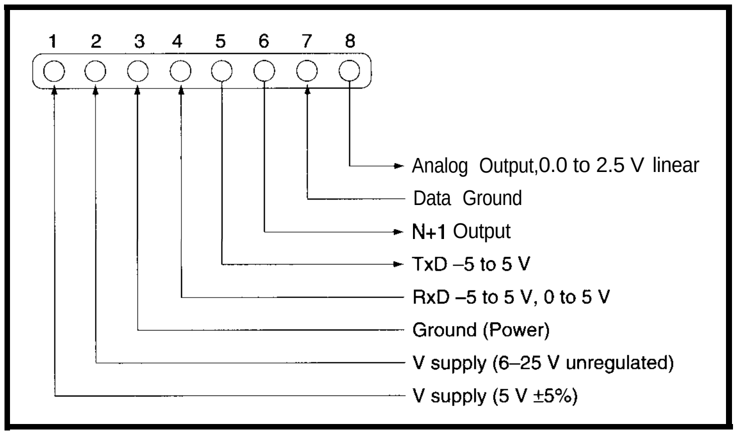


Figure 2—*The electrical connections to the TCM1* include *both* digital and *analog interfaces. The serial lines are most often used* when *connecting the unit to a computer.*

Besides specifying the output format, the other major installation commands deal with calibration. The idea is, with the compass mounted, to take a number of sample readings as the equipment is rotated and tilted, allowing the TCM1 to decompose the local magnetic field into the geographic (good) and interfering (bad) components. The local interference info is stored in EEPROM, and a new calibration is recorded if the compass mounting or equipment location is changed.

As mentioned before, with some knowledge of the ambient interference, the TCM1 can detect (though not correct) unexpected transients and issue a distortion alarm.

With the output format specified and the compass calibrated, you can issue single sample (heading, temp, incline, etc.) commands to check everything out. Indeed, if your application doesn't call for constant sampling, issuing a command each time you want an update may be the way to go. This is also a good approach for battery-powered applications since power consumption is reduced between samples.

Should you prefer (likely the case if you're up against the 6-Hz sampling limit), you can issue the command to put the TCM1 in continuous sample mode. As the name implies, the TCM1 will take continuous samples (at the preconfigured sampling rate, i.e., 6 to 1/60 Hz) and output a result string (with data items you previously selected) each time. A sample output string is decoded in Figure 3.

While the somewhat terse command and response protocol is best for computer communication, it admittedly is a

```
$C<compass>P<pitch>R<roll>X<Bx>Y<By>Z<Bz>T<temp>E<error
code>*checksum<cr><lf>
```

Example:
The TCM1 will return the following:

```
$C328.3P28.4R 12.4X55.1Y12.3Z-18.4T22.3E06*checksum<cr><lf>
```

under the following conditions:

compass heading = 328.3" (true or magnetic, depending on configuration)
pitch = 28.4"
roll = -12.4
Bx = 55.1 µT (x-component of magnetic field)
By = 12.3 µT (y-component of magnetic field)
Bz = -18.4 µT (z-component of magnetic field)
Temperature = 22.3" (F/C depending on configuration)
E07 = Distortion flag is raised-magnetic anomaly nearby

Figure 3—*The TCM1 outputs its data using printable ASCII strings, making it fairly easy to interact with the board using just a comm program if necessary.*

little cumbersome for experimentation and evaluation. Fortunately, the TCM1 comes with a DOS evaluation program (Photo 3) that offers an easy-to-use menu-based interface.

VIRTUALLY REAL

A key to making the much-hyped virtual reality (VR) more real and less virtual is the headtracker-that weird helmet with built-in displays (LCD) instead of a visor-which is at the heart of the experience. Obviously, the VR system needs to know which way your head is pointing and, if it does a poor job, will surely make you sea-sick-not a good selling point!

To specifically target the VR promised land, Precision Navigation plans to offer soon a new variant of the TCM1, the Wayfinder VR. The main differences are a tradeoff of slightly less accuracy (e.g., ±2° vs. cl" heading accuracy) in exchange for a yet-to-be-determined, faster sampling rate.

Of course, the VR wizards are demanding 30+ Hz so they can update position every frame. However, given the associated display rendering, bandwidth, and cost challenges, it may be a case of the designer's eyes being bigger than technology's stomach. Nevertheless, the customer is king. So, look for the Wayfinder VR to spit out samples as fast as it can.

Precision Navigation technology has a lot to offer. Alternative headtracking schemes include gyros, which must be periodically recalibrated due to drift, and active transmit and receive systems that must be physically installed in a single location. By contrast, the TCM1 combines excellent accuracy (notably, no drift) with low power, small size, and total portability.

The bad news is that, at $500+ in small quantities, the technology isn't really accessible for high-volume VR (or any other price sensitive) application. The good news is it's Precision Navigation's intention to license their technology to high-volume customers.

So, if you've got a serious application in mind, don't let the high sticker price mislead you into writing the technology off. Judging by the component bill of materials and the less-than-$100 price of the similar Wayfinder, you may well be wearing one of these things before long.

Meanwhile, I hope Dionne can finally find San Jose. If I hear that song once more, I'm moving to Lodi. ❑

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

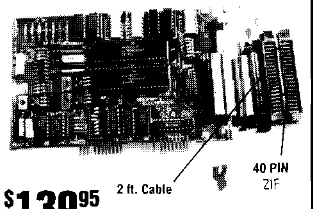
Precision Navigation, Inc.
1235 Pear Ave., Ste. 111
Mountain View, CA 94043
(415) 962-8777
Fax: (415) 962-8776

I R S

419 Very Useful
420 Moderately Useful
421 Not Useful

Device Programmers

PB-10 Internal Card for PC



- 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-PO only))
- **Micros** 8741A, 42A, 42AH, 48, 49, 48H, 49H, 55, 87C51, 87C51FX, 87C751,752
- GAL, **PLD** from NS, Lattice, AMD-16VB, 20VB, **22V10** (EMP-20 only)

FOR MORE INFORMATION CALL

NEEDHAM'S ELECTRONICS, INC.

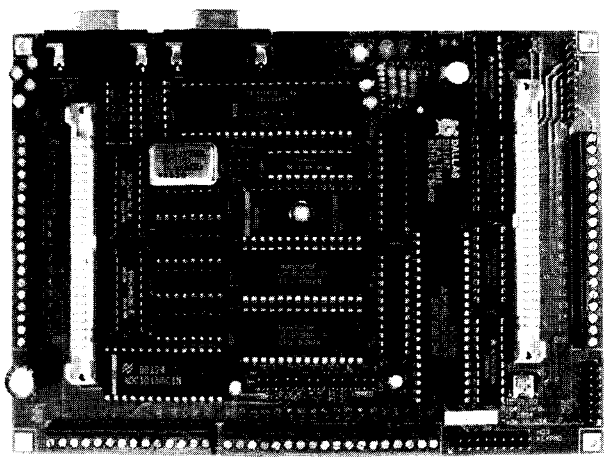
4539 Orange Grove Ave.
Sacramento, CA 95841
(Monday-Friday, 8 am-5 pm PST)

MasterCard VISA
C.O.D.

(916) 924-8037
BBS (916) 972-8042
FAX (916) 972-9960

#129

HIGH PERFORMANCE 8051 CONTROLLERS



Designed to handle demanding control applications, the **RIC320** is optimized for the **DS80C320** at 24 MHz or SABC501 at 40 MHz
(accepts any 40 pin dip 6051 including the Intel 6052 Basic)

Features: 32K EPROM, 64K EEPROM/RAM, 6, 10 or 10 bit with sign A/D converter, real-time clock/calendar, serial I/O's (RS232 / RS422), 16 bits TTL I/O, 16 bits of opto-isolated Inputs, and 16 bits of high-voltage/high-current outputs, LCD, keypad interfaces with software drivers, standard 50-pin headers to drive industrial input/output modular boards (Grayhill, Opto 22), industrial noise Immunity with ground and Vcc planes

Program the **RIC320** using C, Basic, Forth, or Assembly.
An integrated assembly language environment with example software programs comes with the RIC320

RIGEL CORPORATION
PO BOX 90040, GAINESVILLE, FL 32607
(904)373-4629 BBS (904)377-4435

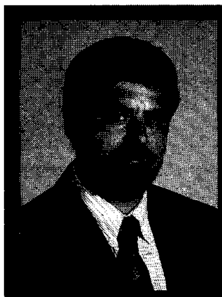
#130

Between the Lines

EMBEDDED TECHNIQUES

John Dybowski

Bar Code and Decoding Bar-code Algorithms



No matter where you look, it seems bar codes are becoming more and more prevalent. John takes a look at a few of the many different encoding schemes in use and their history.

Automatic Identification or Auto ID, for short, revolves around the art of decoding various machine-readable symbologies. This art is far faster and more accurate than manual data entry and therefore serves as a superior alternative. Although there are several machine-readable media, the most popular and enduring is bar code. Its ease of production, proven reliability, and especially its low cost make it a prime solution.

A bar code can be viewed as a form of paper memory, a printable machine language, or a machine-readable document. Any way you look at it, bar code easily translates directly into bitstreams of ones and zeros, the substance of modern computers.

With greater amounts of micro-electronic integration, miniature computers are everywhere, often masquerading as appliances. Bar-code readers, having attained appliance status, can be placed anywhere data capture is convenient or necessary. Combined with hand-held or stationary, contact or no-contact bar-code scanners, they function as front ends for automated data-collection systems.

In extreme cases, small computers are embedded in hand-held bar-code wands providing intelligent, stand-alone readers capable of outputting RS-232 datastreams. These datastreams are suitable for direct input into larger, data-processing computers.

Although a survey of the 50 or so prevailing bar codes reveals that some relatively weak encoding methods have survived, their numbers are few. They exist for historical reasons. That is, because of their rapid early adoption and large installation base, they became entrenched in specific application areas.

As a general rule, however, the industrial and financial sectors are firmly based on pragmatic footing. Poor performers just don't last.

Let's start with an overview of some alternate data-capture methodologies so we can get a better appreciation of bar code's simplicity.

KEYBOARD ENTRY

By definition, Auto ID is keyless data entry. Except as an auxiliary input device, the keyboard is gone. It is only used to enter supplementary information and as a backup if the encoded symbology cannot be read.

OCR-NICE TRY

Optical character recognition (OCR) applies electrooptical techniques to machine-read printed characters which are also humanly readable. OCR is attractive because the same printed characters can be read by people and machines. Using a matrix of phototransducers, the illuminated symbol is scanned, usually with a hand-held device. As the scan head travels across the label, the presence or absence of reflected light indicates the presence or absence of portions of a character. Once the scan is completed, the acquired optical characteristics are evaluated, and (hopefully) the scanned character is identified.

Unfortunately, the key word is hopefully. If you've watched someone labor at scanning an OCR label, then you know what I mean. The problem, of course, is that the information available to decode OCR is very sparse. If a void, spot, or smudge is present, information is often misread or not decoded. As well, OCR has a problem distinguishing similar characters.

To counter this ambiguity, the National Retail Merchants Association (NRMA) has developed two OCR standards. In OCR-A, machine readability is maximized by making each character as different as possible from all others. Although this helps with the problem of machine-induced substitution errors, it follows that the unfamiliar appearance of the OCR-A alphabet results in an increase in human-induced errors.

To make the character set more humanly pleasing, OCR-B was developed. Unfortunately, OCR-B cannot be machine read with the same assurance as OCR-A. Since it is impossible economically to maximize both human and machine readability using the same character set, machine and human readability are, by nature, contradictory.

Font	Character Set	Application
OCR-A Numeric subset (A)	0123456789 |⑂⑀⑁-	Font distorted to maximize machine readability
OCR-A Numeric subset(B)	0123456789 |$.+-	
OCR-A Alphanumeric	0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ {}%?&"*+$,.-/'=|;:▮	
OCR-A Numeric	0123456789+,-./|▮	
OCR-A Alphabetic	ABCDEFGHIJKLMNOPQRSTUVWXYZ {}%?&"*$'=|▮:;	
Farrington 7B	0123456789	Used on credit cards
E13B (MICR)	0 123456789⑆⑇⑈⑉	Used by banks on checks
OCR-B Numeric	0123456789 <>+-/|	Enhanced appearance to human eye
407	0123456789|+$.-/ ⌑	
1428	0123456789|./-+H	

Figure 1—*OCR characters are used when if's necessary fhaf both humans and machines be able to read the coded message.*

Why has so much effort been expended refining a technology that cannot deliver? Technological improvements can yield incremental OCR performance gains, but it's primarily based on economics, not technology. If you consider the inherent problems and the necessary tradeoffs, the whole concept looks like it's based on shaky ground. For your analysis, several OCR fonts are depicted in Figure 1.

MAGNETIC STRIPES

Recording data on a magnetic stripe results in a higher bit density that can easily be produced using a printing process on paper. Magnetics also offers the capability of altering or rewriting the recorded data after it's laid down. Although this rewriting capability is useful in a number of applications, it is actually a disadvantage in applications where data security is important.

Most commonly, magnetic stripes are read by passing an encoded card manually through a slot reader or by using a mechanized, insertion-reading device. Alternatively, a hand-held wand reader can be passed over a stationary magnetic stripe.

The primary disadvantage of magnetic media is its inability to be inexpensively printed. Advantages include a very high bit density and, for certain applications, its read/write capability.

TOUCH MEMORY

Touch memory, a fairly recent addition to the Auto-ID field, comes in a variety of forms. Available in read-only and read/write configurations, touch technology can serve applications that otherwise use OCR, magnetic stripe, or bar code.

The storage medium is silicon rather than ink-on-paper or the magnetic-flux reversals of the technologies I've already described. Silicon offers some intriguing possibilities. The fundamental device is ROM based and functions as a silicon number tag.

Other devices add nonvolatile RAM to the basic ROM configuration for storing changeable data. This provides numerous variations for high- and low-security storage regions. As well, they can be equipped with a real-time clock, a feature that opens applications which are otherwise simply unattainable.

A touch reader is easily the least expensive of the data-capturing technologies. It would be a great deal if your system had a lot of readers and just a few touch devices. However, this is seldom the case. Unfortunately, savings at the reader end are offset by the relatively high cost of the touch devices themselves,

The touch microcontroller interface consists of a single bidirectional port pin and a mechanical probe. The probe contacts the touch device's case, which resembles a small coin cell. The outer shell is the return connection and the center contact is the data connection. This simple and rugged case design is one of the touch device's principal features since this hermetically sealed, stainless-steel case survives hostile environments that would quickly turn paper or magnetic tags to pulp.

BAR CODE

Bar-code symbology provides the right feature mix for a lot of data-capture tasks, including identification of objects, locations, and people. The first step in deciphering a bar code involves acquiring the optical bar or space pattern using some form of electrooptical scanning device.

Let me briefly touch on some of the more common input devices before moving on.

Bar-code scanners can be hand held or stationary, fully automated or require a human operator, and they can operate in the visible or infrared spectrum. The least expensive way to scan a bar-code label is with a hand-held wand or a light pen. This device contains a built-in light source and a sensor for detecting the light reflected from the bar code.

Modern bar-code wands translate the optical symbol into a digital representation. To do this, they have the required amplification and wave-shaping circuitry built in. Optical slot readers operate essentially on the same

AMX™

The Real-Time Multitasking Kernel

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

Features

NEW
DOS Compatible File System
TCP/IP

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

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

For a free Demo Disk and your copy of our excellent AMX product description, contact *us today*.
Phone: (604) **734-2796**
Fax: (604) 734-8114

KADAK Products Ltd.
206 - 1847 West Broadway
Vancouver, BC, Canada V6J1Y5

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

#131

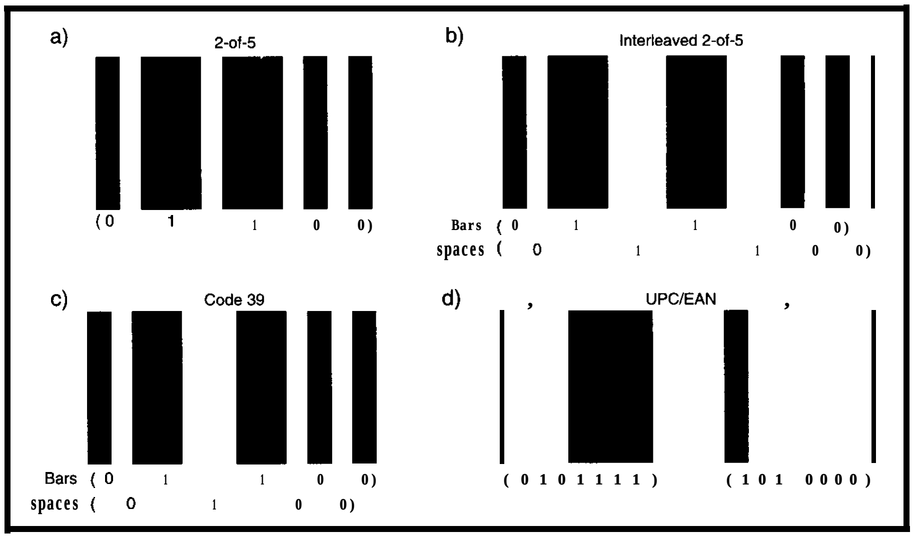


Figure 2—These four common bit-encoding methodologies all depict the character6 UPC goes one step further and defines both left- and right-handed symbols for the same character (d).

principal, but require bar code to be passed through a slot in a manner similar to a magnetic card.

Fixed-beam scanners require bar code to be moved past the scanner. These can range from very small devices designed to read high-density codes to large units that read containers as they pass along a conveyor belt.

Moving-beam scanners can be hand-held or stationary. A number of light sources have been used, but modern devices almost exclusively use laser light. Helium-neon or solid-state lasers are most often used. A great depth of field, combined with a rapid, repetitive light sweep, results in a much higher "first-read rate" than a single-pass scanner offers. This improved first-read rate is also the result of reading the symbol continuously until it comes out right.

Scanners based on charge-coupled device arrays (CCD) replicate the function of a moving-beam scanner without moving parts. Using such a linear array, a solid-state scan is achieved by flooding the symbol with light and reading the transducer-array outputs sequentially. The resulting datastream is accomplished without physical contact or relative motion between the symbol and the read head.

ENCODING TECHNIQUES

Bar codes are composed of a series of dark and light bars that represent letters, number, and other symbols. These dark and light bars (or bars and spaces) are organized according to the specific rules of a particular bar-code symbology. In contrast to OCR, bar codes are conceived as a machine language designed for computers.

Bar code uses the ones and zeros fundamental to digital logic and is essentially binary in form. That's not to say that human-readable characters may not be contained on a bar-code label. But, the characters are provided for information purposes only; there is no attempt to electrically decipher them. The computer and human information is kept segregated-a lesson learned from OCR. Similarly, for increased efficiency, bar-code symbologies are optimized for a specific application. Tradeoffs are made between conflicting properties.

I'll be looking at several of the more popular and useful, not to mention simpler, bar-code structures a little more closely. But first, let me cover some fundamental concepts.

The smallest element of a bar code is a module (also sometimes referred to as the X *dimension).* In most cases, the wider bars and spaces are integer multiples of a module. Clearly, these relationships remain consistent as the codes are magnified or reduced in overall size and as they are scanned at different velocities.

Modules translate optical bar code into a binary code. Ones and zeros are extrapolated from the bar-space pattern which varies according to the specific bar code. In some cases, wide

Character	Code
0	00110
1	10001
2	01001
3	11000
4	00101
5	10100
6	01100
7	00011
a	10010
9	01010
Start	110
stop	101

Table l--Two-of-Five *encoding defines just the numbers O-9 and unique start and stop symbols. The code has been wide/y used in industrial applications since the late '60s.*

elements translate to ones and narrow elements, to zeros.

Other bar codes assign binary values to dark and light elements. In such a scheme, a dark bar, which spans several module dimensions, accumulates the respective number of one bits. Similarly, zeros are accrued using spaces of varying width.

There are different ways these fundamental bit-encoding techniques are applied to creating a bar-code character set. Some bar codes use only the bars to represent data bits, with the spaces functioning merely as separators. Others use both bars and spaces to form a single-code representation of a character. Another method using both bars and spaces interleaves the coded characters so that bars encode odd characters and spaces represent the even.

Although a variety of other techniques are also in use, these methods predominate.

IT'S ALL RELATIVE

Bar-code decoding algorithms assume the velocity of the code scan and the size of the bar-space elements are unimportant. Provided the scanning velocity does not vary beyond a given parameter, the module relationship between bars and spaces can be determined by comparing the bar and space widths in the time domain.

Figure 2 presents the four most common bit-level encoding techniques. Some code structures, such as Two of Five (2/5), compare bar widths. A code such as Interleaved Two of Five (I 2/5) requires bar-to-bar and space-to-space comparison. Spaces are used in an identical fashion to the bars. Code 39 can be most easily understood as a series of dark and light bars rather than bars and spaces. Its wide and narrow elements, however, are interpreted the same as 2/5 and I 2/5.

Finally, codes such as UPC are structured so that a bar denotes a bit and a space denotes a zero. With UPC, width measurements are usually done by making comparisons between a bar and its associated space and the next bar and its associated space.

There are some characteristics that are important for evaluating bar-code symbologies. I'll introduce these concepts here and elaborate more fully on them later when I describe real bar-code symbologies.

First, some codes are classified as continuous and others are classified as discrete. These terms describe the way encoded characters are concatenated to form a multicharacter, bar-code label. In a continuous code, the intercharacter space is part of the code structure and must adhere to strict dimensional restrictions as defined in the code specification. In a discrete code, the intercharacter space is not part of the code and is allowed to vary within fairly wide dimensional limits.

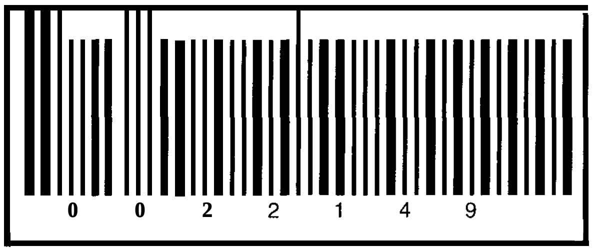


Figure *J-Two-of-Five code contains all the* **information** *in the width of the bars. The spaces function only as separators.*

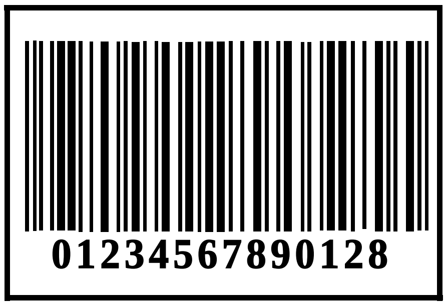


Figure 4-Interleaved Two-of-Five *attains a higher density than Two-of-Five by using the spaces as we// as the bars for encoding data.*

Secondly, some codes are designed in such a manner that they are self-checking. In other words, an algorithm can be applied to each character so that substitution errors can only occur if two or more elemental errors appear within a single character.

Finally, for added data security a check digit can be appended to the bar code. With some codes, this is mandatory, but is optional with others. In any case, detailed calculation methods are outlined in the respective bar-code specifications. It is important to realize that these check digits are treated as an inherent part of the bar code. If the check calculation does not yield the expected result, the reader does not decode the scan.

REAL BAR CODES

Two-of Five Code is one of the simplest and most straightforward bar codes. Having its origin in the late 1960s, this numeric-only code has been applied to a number of industrial applications and most recently has been used in sequentially numbering airline tickets.

Two-of-Five Code contains all the information in the width of the bars-the spaces function exclusively as separators. Two bar widths are defined as a wide bar typically three times the width of a narrow bar. Narrow bars are interpreted as zero and wide bars as one. The intervening spaces may be any width, but are typically equal to the narrow bars.

Data is encoded using a modified binary method in which the bar positions from left to right are assigned weighting factors of 1, 2, 4, 7, and parity. (Zero is an exception to this rule.) In addition to the numeric symbols, distinctive start and stop codes are defined for bidirectional scanning. The character set encoding for Two-of-Five Code is shown in Table 1.

From Table 1, you can see that there are two levels of algorithms for deciphering bar code. First, an algorithm is applied to recover the stream of one and zero bits. Second, these ones and zeros are combined to create the bar-code character set. Two-of-Five Code uses a simple modified binary approach in which conversion is direct. Other codes use different coding schemes or lookup tables.

Since the white spaces carry no information, it follows that their width within the characters is not critical. The spaces between characters can be loosely defined as well. Because of this, Two-of-Five Code is classified as discrete.

The Two-of-Five-Code structure is also self-checking since all characters are composed of two wide bars and three narrow bars. Notably, this is where the code gets its name-two of five bars must always be wide. A decoding error would require two independent printing defects within the same scan line, a condition which would dictate the alignment of a void on a wide bar with a spot on a narrow bar within the same character. A Two-of-Five bar code is shown in Figure 3.

Developed in 1972, Interleaved Two-of-Five Code attains higher character density than Two-of-Five Code by using the spaces as well as the bars for encoding data characters. The actual encoding methodology is identical to that used in Two-of-Five Code. This code has seen much use in warehousing, heavy industrial applications, and the automotive industry.

Bars are used for encoding those numbers that appear in the odd positions and the even-number positions are represented by spaces. As a result of this interleaving, the symbology requires that an even number of digits be encoded. If an odd number of digits must be represented, a leading zero initiates the data string.

Note that by placing information in the spaces as well as the bars, Interleaved Two-of-Five Code is no longer discrete. It does, however, retain the self-checking attributes of Two-of-Five Code. Figure 4 illustrates an Interleaved Two-of-Five bar code.

The ‘full alphanumeric Code 39 (also known as 3-of-9 Code) was developed in 1975. Each stand-alone Code 39 character is represented by a group of five bars and four spaces. Two of these bars are wide and one of the four spaces is wide (i.e., there are three wide elements out of a total of nine elements).

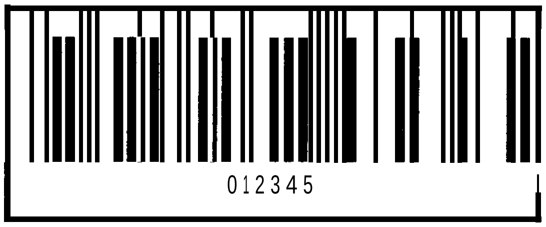


Figure 5—Each Code-39 character is represented by a group of five bars and four spaces. Two of the bars and one of the spaces are wide (giving three wide bars out of the nine elements, hence the name). Code 39 includes a full alphanumeric character set.

This arrangement results in 10 x 4 possibilities, or 40 characters. Four extra characters ($,/,+, and %) are formed with all narrow bars and three wide spaces. The character set includes a single start/stop code (*) and 43 alphanumeric data characters. Code 39 is a discrete, self-checking code. Figure 5 shows a Code 39 bar code.

JUST AS PLANNED

Pick up a spec on any of the current bar-code symbologies and you will find everything-print-to-contrast ratio, dimensional information, reference-decoding algorithms—spelled out in great detail. Judging from the vast information, you might conclude that the evolution of the bar code had been carefully orchestrated and strictly regimented. The fact is, as in so many human endeavors, these specifications are an attempt to document what had taken place.

Technology often evolves in surprising and unexpected ways. Sometimes it takes on a life of its own. Few of us like to admit this, especially if we’re charged with advancing the art. When we come up with something really good, the natural inclination is to say that it came out just the way we planned it.

Long before bar code became common in retail, it tracked the movement of millions of tons of freight. This 1959 implementation, developed by Sylvania, used a stationary reader. The reader included a Xenon light source and photomultipliers which sensed the reflected light from the bar code. The numeric symbology consisted of strips approximately ¾″ x 6”. These giant bar codes were scanned by moving past readers and were called the *Kartrak rail-tracking system.*

After overcoming numerous technical obstacles, the read rate was purportedly comparable with many of today’s UPC scan rates. This ultimately led to the installation of about 1000 readers and the bar coding of approximately 1.8 million railroad cars. In addition, 200,000 piggyback tractor-trailer containers were also bar coded. The readers were usually situated at junctions, interchanges, and entrances to rail yards. Kartrak usage peaked in 1948 and then gradually declined due mostly to inadequate label maintenance. Today, only about 50 scanners are still in operation.

MULTIDIMENSIONAL CODES

The codes I’ve described so far use only one dimension for information storage. However, there is a limit to how much data can be encoded using this, essentially serial, method.

Using two dimensions for data storage results in a substantially higher volume of data-per-unit size. However, this does rule out the use of

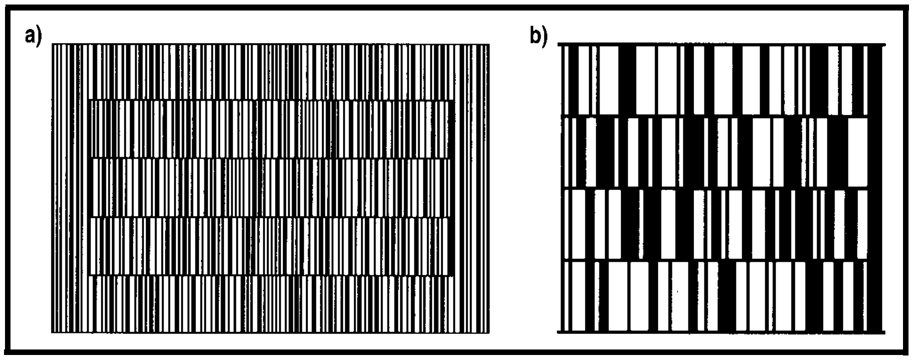


Figure 6-Examples of two-dimensional codes include (a) CodaBlock, which is representing 65 characters here, and (b) Code 49, which is depicting 26 characters here.

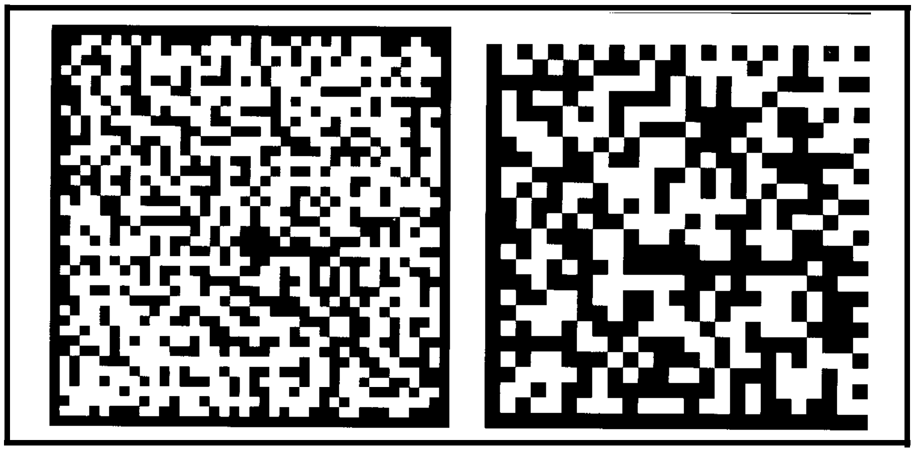


Figure 7-Among other2D codes, Veri Code (left) encodes 185 characters while its counterpart, Data matrix (right), represents 68 characters.

an inexpensive, hand-held scanner. Moving-beam laser scanners or CCD scan heads prove suitable for this purpose. These devices are available in fixed-base or hand-held configurations.

The capability of packing more information into a given area can be useful in a number of ways. First, it simply stores more information. A less obvious method is to provide enhanced data integrity using the extra storage to contribute some data redundancy and, more importantly, error-detection and error-correction information. If the error-recovery mechanism is implemented properly, a label can be accurately read even if substantial portions have been damaged or obscured by dirt.

Developers of these 2D symbologies have taken different approaches to concatenating symbols as well as implementing data security and binary bit-encoding techniques. Figure 6 shows a number of 2D bar codes.

The simpler 2D codes start with existing symbologies and add the required control information to handle longer records and to ensure data accuracy. CodaBlock, which is based on Code 39 with additional control information and check digits, is representative of this approach.

Another strategy is to combine computer science disciplines with a formal data-redundancy and error-correction regimen that detaches the symbology from its traditional bar-code precursors. Only the optical scanning characteristics are retained. Figure 7 offers representatives of this class of inventive thinking. Judging by this strange landscape, all of a sudden it looks like we're a long way from the rail yards of 1959.

A SLIGHT REPRIEVE

Last time, I promised to put a microcontroller behind bars. Even though I tried to keep this month's introduction as brief as possible, I've managed to run out of space. Although I've covered a broad range of topics, I've done little to explain the technology in depth. As a result, our microcontroller has gained a slight reprieve.

To gain a better understanding of how a bar code really works, it would be instructive to pick one apart. Next month, I'll narrow my focus and concentrate on the hardware and firmware issues involved in putting a real microcontroller to work decoding real bar code. ❑

John Dybowski is an engineer involved in the design and manufacture of embedded controllers and communications equipment with a special focus on portable and battery-operated instruments. He is also owner of Mid-Tech Computing Devices. John may be reached at (203) 684-2442 or at john.dybowski@circellar.com.

IRS

422 Very Useful
423 Moderately Useful
424 Not Useful

The Real Logic Analyzer

Test **your Logic** circuits **with** the printer **port** of your IBM or compatible computer!

☑ 5 Input capture channels via printer port
☑ High Speed 64K input capture buffer
☑ Glitch capture and display
☑ Full triggering on any input pattern
❑ Automatic time base calibration
☑ 4 cursors measure time and frequency
☑ Save, print or export waveforms (PCX)

The Real Logic Analyzer is a software package that converts an IBM or compatible computer into a fully functional logic analyzer. Up to 5 waveforms can be monitored through the standard PC parallel printer port. The user connects a circuit to the port by making a simple cable or by using our optional cable with universal test clips. The software can capture 64K samples of data at speeds of up to 1.2uS (Depending on computer). The waveforms are displayed graphically and can be viewed at several zoom levels. The triggering may be set to any combination of high, low or Don't Care values and allows for adjustable pre and post trigger viewing. An automatic calibration routine assures accurate time and frequency measurements using 4 independent cursors. A continuous display mode along with our high speed graphics drivers, provide for an "Oscilloscope-type" of real time display. An optional Buffer which plugs directly to the printer port is available for monitoring high voltage signals.

LOGIXELL ELECTRONICS
61 Piper Cr.
Kanata, Ontario
Canada **K2K 2S9**

Requires 286, or higher with EGA or VGA display.

LA20	Software Only	$79.95US
Software With Test Cable		$99.95US
BUFF05	Buffer	$39.95US

Tel: (613)599-7088
Fax: (613)599-7089

VISA MasterCard

#132

CONNECTIME conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(203) 871-l 988-Four incoming lines
Internet E-mail: **sysop@circellar.com**

With somwhat limited space this month, I decided to devote the who/e column to just one message thread. RS-232, RS-422, and RS-485 connections are the most popular in use today for relatively slow asynchronous serial lines. However, not understanding how to ground systems that use them can cause major headaches. Check out some tips from the experts.

RS-232, -422, -485 grounding

Msg#:30409
From: IAN GARMAISE To: ALL USERS

This may seem like a silly question, but it is perplexing to me. I have a control system that uses RS-232-to-RS-422 converters. After a lot of on-line consultation, I went with the following wiring scheme:

I use two twisted pairs of #22 wire, overall shielded. The converter takes RJ-11 connectors. The vendor supplied RJ-1 l-to-DB-25 female connectors for the RS-232 side. I made RJ-1 l-to-DE-9 connectors to connect the adapter to the RS-422 cable. I used shielded DE-9 connectors. On one side, I connected the ground directly from the converter board to water-pipe ground. I did not connect ground on this side to the cable shield. On the other end, I connected the ground from the converter board to the PC's metal case, and I also connected the shield wire from the cable to the same case.

What's the right way to deal with the shield wire when you're attaching a metal DE-9 cover? I just wrapped it back, attached it to the cover with one screw, and ran a wire from the other screw to my case ground. Somehow this doesn't look right. I couldn't solder the wire to the cover cause I couldn't get the cover hot enough. Any suggestions?

By the way, the RS-422 works fine as long as I use surge protectors with my cheap 9-V power supplies on each end, and as long as both sides are grounded. Without a ground on both sides, the RS-422 didn't work, which makes sense.

Msg#:30908
From: JAMES MEYER To: IAN GARMAISE

Not to me, it doesn't.

The way I understand it, there should be no need for a "ground" in a connection that is truly RS-422 based.

I would be *very* careful about "water-pipe grounds" if I were trying to implement an RS-422 connection.

The whole point of RS-422 is its balanced, push-pull, differential nature and the fact that it doesn't need any "grounds." If you need a "ground" to make things work, then you must be doing something wrong.

Perhaps I missed something when I read your description of the connections you made. Please expand a little on *exactly* what you've got in the way of converters and how they're hooked up.

Msg#:31048
From: IAN GARMAISE To: JAMES MEYER

I basically was following some precise instructions from a person on the CompuServe Eng forum. He said that the differential voltages used in RS-422 still need a reference ground. His instructions were as follows:

1. If concerned about ground loop, use isolated RS-422 drivers and receivers, and CONNECT the (isolated) signal ground of each driver and receiver to provide a reference ground (the RS-422 doc does in fact show a connected signal ground.

2. If ground loop is not an issue, then connect the ground of each driver and receiver to mains ground at each end (this is what I did]. As for the shield ground, lots of difference of opinion here, but the consensus of the noise books I borrowed and my converter supplier is to ground the shield at one end to the mains ground.

The CompuServe person said that sometimes RS-422 drivers connected without ground reference will appear to work correctly, but will fail eventually.

On one end, I have a PC serial RS-232 port. On the other end, I have a mag card reader with an RS-232 interface. They are connected as I described above.

On both ends, I'm using Tripp Lite surge protector/ noise filters with the 12-VAC/9-VDC power supplies to the converters. While testing I discovered that as the CompuServe person predicted, transmission was impossible if I disconnected the signal ground at one end.

Msg#:31960
From: JAMES MEYER To: IAN GARMAISE

Sorry. Your CompuServe person is entirely right, and I was entirely wrong.

No, I can't leave it at that....

The reason I put "ground" in quotes above, is because "ground" is a lot like the weather. Everybody talks about it, but very few know exactly what it is.

If you're in an experimental mood, you may want to try this:

Disconnect all of your "water pipe grounds." Then connect the shield(s) of your twisted-pair cables at *each* end of the cable to the "common" connection on the isolated RS-422 side of each converter box.

Don't connect the shields to anything else. Don't make any connection from the RS-232 side of the converter box to "ground" *or* the isolated RS-422 side of the network.

The RS-422 side of the converter boxes is described as "isolated" for a very good reason. That's because there should be no necessity to make any connection to anywhere other than another RS-422 circuit.

Msg#:31977
From: IAN GARMAISE To: JAMES MEYER

I'll give that a try. I believe that should work. But is it intrinsically safer than using water-pipe ground? (The converters I'm using do not have built-in isolation.)

Msg#:32412
From: JAMES MEYER To: IAN GARMAISE

I got confused. The CompuServe person said:

1. If concerned about ground loop, use isolated RS-422 drivers and receivers, and CONNECT the (isolated) signal ground of each driver and receiver to provide a reference ground (the RS-422 doc does in fact show a connected signal ground.

I went back and reread your earlier messages and you never said explicitly whether your converters were isolated or not. I just assumed they were.

In any event, I believe the following will get you going with as "clean" a connection as you can get without adding anything else to your system.

Wire the two twisted pairs to the + and – signal connection pins on the '422 side and connect the shield(s) of the pairs to the common on the '422 side. Do this on both ends of the cable between the converter boxes.

You already have the equivalent of a "water-pipe ground" in the third (ground) leg of the AC power line which is carried all the way through the system from the entrance box where the AC power comes into the building, through the wiring in the wall, through the three-prong AC plugs, through the Tripp Lite surge/noise box, through the AC power cord to the computer chassis and then to the RS-232 common connection on the back of the computer.

If you make any *other* connections to ground, such as another connection to a water pipe somewhere, you are *creating* a "ground loop" and run a risk of introducing stray currents and noise.

Msg#:32647
From: ED NISLEY To: JAMES MEYER

The catch with all this is the common-mode voltage rating of the RS-422 (or -485) circuits. If there's a nontrivial voltage difference between "ground" at the two locations you can fry the receivers without too much trouble. Adding a common connection between the two ends won't help, as it'd have to carry enough current to equalize the ground potentials.. .and that could be quite a lot!

Might be worth measuring just to find out how much trouble you're in..

Msg#:36141
From: PELLERVO KASKINEN To: IAN GARMAISE

There are two or three basic situations that determine the amount of ground potential difference between two sites. Let's call the first one a steady state, the second one a usual-transients state, and the third one, abnormal-transients state.

Starting from the abnormal-transients state, we are talking mainly of a direct hit or a near miss of lightning. The likelihood is low, but the consequences are very severe. You can have potential gradients of 100 V per foot or more for a very short time. The gradient decays with distance in a hyperbolic fashion, but the voltages between cable ends can be truly amazing.

The usual transients are due to in-rush currents to motors and transformers. If there is no sparking from the winding to the frame or similar faults, the resulting jolts to the grounding post are not likely to exceed a few tens of volts for a few tens of milliseconds. This is something that your communication lines can (and should) be fortified against. The lightning strike is an almost hopeless situation even with optocouplers and similar, but the usual transients should be covered.

A steady-state voltage is normally due to unbalanced loads on three-phase systems or something like an electric train system using the rails as a return path. The unbalanced loads with reasonable ground conductivity (moist soil, minerals, and so on) rarely results in more than about a 5-VAC difference between any two points within a plant or a building, even between separate buildings.

Now, what are the consequences for an RS-422 line? The 5 VAC is a little over the limit where communication errors creep in. RS-485 may work properly, depending on whose chips you use. Just a couple of weeks ago I read a new National Semiconductor app note about this particular issue.

You say your converters do not have any built-in isolation. Given that, your options are somewhat limited, because I assume the computers are locally grounded. With an isolation at least on one end, you could use the cable shield to make the connection back to the end that is grounded. Without that option, you probably have to build a big (60-HZ) common-mode choke at one or both ends. You could get commercial common-mode chokes for each signal line pair, but probably it would be best to have just one that includes the shield. What you would need to do is get a good-size transformer core (say 50-100 W size), take the windings-if any-out and loop your whole signal cable through the opening some 30 times or more if there is room. The more, the better, within reason. The common-mode choke takes care of the ground potential difference without affecting the overall signal (that is differential) fidelity.

This common-mode choke should bring the currents running in your cable under control, together with any resistance in the cable. It may sound funny, but beefing up the cable might be actually counterproductive, because the low impedances involved maintain the potential difference and just feed more current through a thicker cable.

Now, I understand your cable does not run between different buildings, What I have said is more for anybody who might have such a situation. But within a single building, the conditions can be equally bad. And I wanted to give the numbers for putting the safety issues a little into perspective. Leaving one end not locally grounded presents indeed the danger of having two potentials exposed side by side. That is, the local ground and the remote ground. Touching two different potentials simultaneously is the danger that safety grounding tries to avoid. However, it is also acknowledged that voltages below 24 V or even 48 V are considered "safe," except in bath tubs and such. If our voltages are typically below 5 V and with usual transients below 48 V, we would not violate the principles of safety if we have one end with optically isolated converter that is actually tied to the far side ground.

Finally, we naturally should provide touch protection anyway, just like the phones do. You just do not have an easy access to the conductive parts within the phone or the converter. See, where I'm coming from? The phone system actually uses transformer isolation but is otherwise very analogous with the isolated RS-422 converter. You also have the local overvoltage protectors somewhere near your home, tied to your home ground (or close enough to be effectively the same). But the only DIRECT ground is at the phone company premises. So you have the remote ground situation with an insulated phone in your hand. Dangerous? Yes, it CAN be dangerous during a direct lightning hit, but how often do you think that!

In other words, get at least one isolated RS-422 (or -485) converter and do the grounding at one end and you should have almost no problem, either for signal fidelity or for your safety.

Msg#:36346
From: IAN GARMAISE To: PELLERVO KASKINEN

Thanks for a well-reasoned discussion of this issue. This is not a subject that usually gets a good practical treatment, at least in the texts I've looked at.

I do still have a signal fidelity problem, but I have the impression this is coming from the RS-232 sections beyond the converters. The card readers I am using in this system do provide toroid chokes (intended to reduce radiated RFI) that they suggest running the signal cable around at least three times. I have not done this because the Canadian DOC does not require it (unlike the FCC) and I had the impression it might reduce signal strength.

For peace of mind, and to see if it improves fidelity, I'm going to try adding isolation on one end as you suggest. I assume you also would want to ground the cable shield on one end and have it not pass over the isolated converter. Would one connect the shield ground to the RS-232 signal ground (before the conversion, on the grounded end of the line), either with or without a resistor?

Msg#:44439
From: PELLERVO KASKINEN To: IAN GARMAISE

Regarding the toroids: Yes, the small toroids would help against radiating the RFI. And contrary to your fear, they would not reduce your signal strength, being used in a common-mode-reducing fashion. But they are too small to do any good for the 60-Hz common-mode noise you are assumed to experience.

What actually is likely to cause signal fidelity problems is exactly the common-mode noise, and that mostly due to the effects it has to the -transmitter_, if I read the National Semiconductor application note correctly. But why speculate, when we can determine the situation with a simple measurement?

Just hook up the cable at one end. Connect the shield to the local ground at the same end. Leave the other end completely floating by whatever is the easiest way. Then put a voltmeter on AC scale between the cable shield and the equipment frame and/or common at the disconnected end. Make several measurements over a period of time, to include potentially varying conditions due to starting motors and so on. This measurement period should be at least as long as your average time between the signal fidelity problems, if they are not continuous.

When you add an isolated converter at one end, that is exactly the end that you leave completely floating. You make the ground connections at the unisolated end, preferably to the computer (or other communication equipment) frame. At the isolated end, take precautions against accidental shorting of the cable shield to anything. Use heat shrink tubing or electrical tape to keep the end of the cable shield within an insulating jacket, unless you need that for completing the _common_ between the two RS-422 sections (i.e. if you do not have a wire in your cable for that purpose). But in principle, you should have two pairs for the signal and one wire for the common between the two units.

At the isolated end, definitely do not bridge the isolation barrier with anything, intentional or accidental! No resistors, capacitors, or anything. The only two things required besides the signal connection to the other unit are the optically isolated signals and an isolated power for the floating part. The power may be provided by a built-in DC/ DC converter on the isolated RS-422 converter or is requested through a marked terminal, in which case a simple wall mounted power supply should do fine.

Now one more concern. My experience points towards protocol problems as often as to ground noise. There may be bias resistors with jumpers to select an appropriate combination. We had to study the situation for some time before we got the idea straight the first time. The RS-232 single-ended system had been simple and had set our minds. Getting into the differential -422 with its biasing requirements and the + and – naming conventions at first did not seem to fit into place at all. But reading the instructions a few times and making a few sketches of the signal current flow paths finally made things drop into place. I hope you have all this part under control.

Msg#:46481

From: IAN **GARMAISE** To: PELLERVO **KASKINEN**

I believe I do have the noise problem mostly under control now, on both my RS-422 segment that I am testing and the other three lines that are straight RS-232. Right now I have no isolation on any of the lines. The RS-422 currently is referenced to mains ground at each end, and the shield is connected to ground at one end, as it is on the RS-232 lines. I was using solder connector DE-9s, but I am going to switch to inserted crimp pin DE-9s since these seem to present less opportunity for the ground wire to accidentally contact a signal wire. The only problem I've had so far is finding a good-quality crimping tool for these crimp connectors that costs less than $175 (cdn).

It did take me a while to be certain that I had the correct polarity on the RS-422 adapters, given the jumpers on each end and poor documentation, but by quadruple checking and careful diagramming I was able to achieve correct connection on the first try.

We invite you call the Circuit Cellar BBS and exchange messages and files with other Circuit Cellar readers, It is available 24 hours a day and may be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps. For information on obtaining article software through the Internet, send E-mail to info@circellar.com.

ARTICLE SOFTWARE

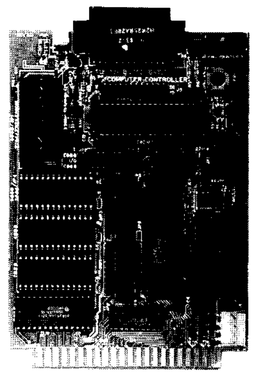
Software for the articles in this and past issues of *The Computer Applications Journal* may be downloaded from the Circuit Cellar BBS free of charge. For those unable to download files, the software is also available on one 360 KB IBM PC-format disk for only $12.

To order Software on Disk, send check or money order to: The Computer Applications Journal, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your VISA or Mastercard and call (203) 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

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" 27641126 EPROM programmer, three E-bit parallel ports, an auto-baud rate detect serial console port, a serial printer port, and much more.

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

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

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

To Order Call 1-800-635-3355
Tel: (203) 871-6170
Fax: (203) 872-2204

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

CALL FOR OEM PRICING

MICROMI NT, INC. 4 Park Street, Vernon, CT 06066
in Europe: (44) 0285-658122 • in Canada: (514) 336-9426 • in Australia: (3) 467-7194
Distributor Inquiries Welcome!

#119

STEVE'S OWN INK

What's in a Name?

This issue is the last magazine of our sixth year. I guess that's a milestone of sorts. It's not easy starting a magazine, and it's a lot harder keeping one going. It requires a constant balancing act among editorial focus, advertising support, newsstand recognition, and subscriber patronage. Change too much in any one direction and the whole thing can start sliding.

As you already know, Circuit *Cellar INK* was started as a spin-off of my original Ciarcia's Circuit Cellar projects in *BYTE.* I was offered the opportunity to continue at *BYTE* following their new direction catering to "computer-purchase influencers," but I declined. MIS sounded like a disease to me.

The first issue of Circuit Cellar *INK* purposely looked like the hardware glory days of the computer revolution. I titled it "Inside the Box Still Counts" as a reminder that, although computers were turning into appliances, someone earlier in the process was actually responsible for designing the device. Circuit Cellar *INK* was to be the new refuge for displaced technical inquisitiveness.

Over the years, we have tried hard to preserve that foundation and direction, While original readers identified instantly with the name "Circuit Cellar" and the Tinney cover paintings, we wondered if a more descriptive name might be less confusing. For the last couple years, we've sort of been INK, *Circuit Cellar, Computer* Applications Journal, and "You know, Ciarcia's magazine." Interestingly, if we've had five letters in total regarding all the presentation changes, I'd be surprised. Is it disinterest?

No. My wife would be kind and just call it the "engineering mentality." Seventy-five percent of Circuit Cellar subscribers have been with us for more than 4 years, and 47% have been there since day one! Because we have maintained the same editorial focus, independent of names and colors, basically you don't care what we call it.

Should I be concerned? As an engineer, I subscribe to the collective mentality. But, as a publisher, I definitely blew it!

The last straw was a couple weeks ago. While at a Brentanos bookstore, I overheard a couple of techies talking. Staring point blank at the latest copy of Computer Applications Journal, one said to the other, "I wonder whatever happened to *Circuit Cellar*? I used to like the kinda stuff that Serseeah fella did."

Well, that did it. So, my fellow techies, put on your sunglasses for the next issue. We'll be there bigger and brighter than ever. There's only one name, and I guarantee you won't miss it!

Steve