

CIRCUIT CELLAR

INK[®]

THE COMPUTER APPLICATIONS JOURNAL

#64 NOVEMBER 1995

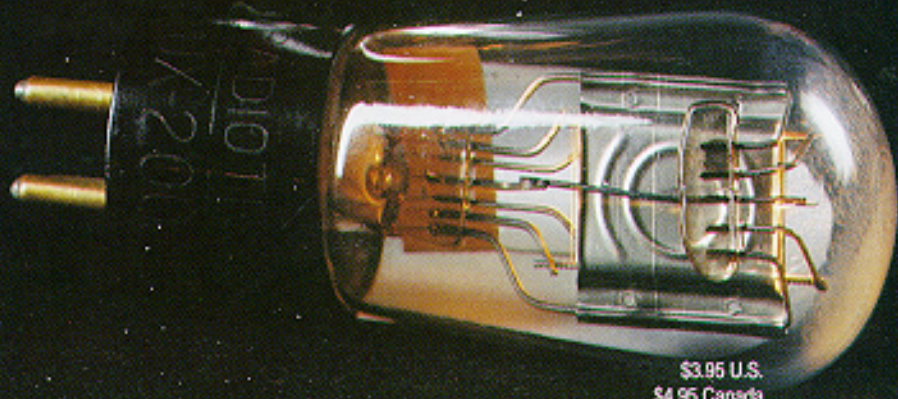
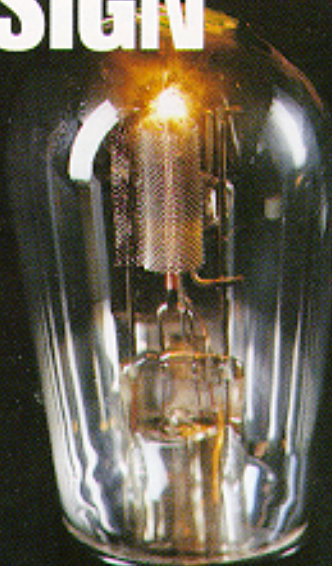
ANALOG DESIGN

Rediscovering
Analog Computers

A Transputer-based
Parallel Computer

Developing a
Virtual Hardware
Device

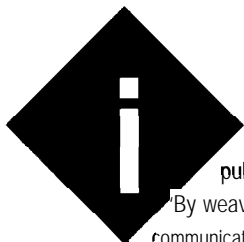
Power-Line
Modems



\$3.95 U.S.
\$4.95 Canada

EDITOR'S INK

Weaving the Web



Do you believe it was Charlotte the spider who first published on a web (remember *Charlotte's Web*?). By weaving words for all to see, a tiny creature communicates to the world.

Such is the case with the World Wide Web. The Web gives even the smallest of individual the capability to publish something for the world to see without the expense of printing and distribution.

The growth of the Web over the past year has been remarkable. Only a year ago, the first version of **Netscape** was released and immediately rocketed to the top of the popularity charts. During that time, I've received dozens of messages asking when we're going to put up a Circuit Cellar Web page.

I'm happy to announce that we've done just that. As with most Web pages, this one still needs work and will be continually evolving. Currently available is subscription information, an ftp link to many of the magazine-related files from the most recent issues, our author's guide, next year's upcoming themes and deadlines, results from this year's Design Contest (over a month before you'll see them in print), and more.

We first tied into the Internet over two years ago with an E-mail connection to the Circuit Cellar BBS. Our Web page marks our continued commitment to keeping in touch with our readers and providing up-to-date technical information.

How do you connect? Point your Web browser to <http://www.circellar.com/>. The pages are designed for Netscape, but look fine with other browsers. Let me know what you think.

Onto the **INK** at hand. Many engineers tend to think of amplifiers, filters, and wireless communications when asked about analog design. Our first article illustrates there is more to analog than signal processing. Analog computers have played an important role in the past, and their usefulness today can't be denied. You just have to train yourself to think analog.

Another computing tool that has been around for a while but just hasn't caught on is the transputer. If you've heard about them but don't know quite what they'll do, check out how to design a PC plug-in board using an array of transputers.

A big stumbling block in many embedded software design cycles is waiting for real hardware to test your code. Our next article shows some neat techniques for simulating hardware devices on the test bed.

In our last feature article, we wrap up the series on designing an engine-control system. This enormously popular series concludes with interesting accounts of the final testing and fine-tuning phases.

In our columns, Ed looks at interrupt processing in Virtual-86 mode, Jeff selects a low-cost power-line modem chip, and Tom follows up last month's video-processing chip overview by covering similar audio-processing silicon.

CIRCUIT CELLAR®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

PUBLISHER
Daniel Rodrigues

EDITOR-IN-CHIEF
Ken Davidson

PUBLISHER'S ASSISTANT
Sue Hodge

SENIOR TECHNICAL EDITOR
Janice Marinelli

CIRCULATION MANAGER
Rose Mansella

TECHNICAL EDITOR
Carole Boster

CIRCULATION ASSISTANT
Barbara Maleski

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

CIRCULATION CONSULTANT
Gregory Spitzfaden

WEST COAST EDITOR
Tom Cantrell

BUSINESS MANAGER
Jeannette Walters

CONTRIBUTING EDITORS
Rick Lehrbaum
Russ Reiss

ADVERTISING COORDINATOR
Dan Gorsky

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

PRODUCTION STAFF
John Gorsky
James Soussounis

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

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 (860) 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 Circuit Cellar INK Subscriptions, P.O. Box 696, Holmes, PA 19043-9613 or call (600) 269-6301. POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 698, Holmes, PA 19043-9613.

Cover photography by Barbara Swenson
PRINTED IN THE UNITED STATES

For information on authorized reprints of articles,
contact Jeannette Walters (860) 875-2199.

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

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

SOUTHEAST
Christa Collins
(305) 966-3939
Fax: (305) 985-6457

MIDWEST
Nanette Traetow
(708) 357-0010
Fax: (708) 357-0452

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

Circuit Cellar BBS—24 Hrs. 300/1200/2400/9600/14.4k bps. 6 bits, no parity, 1 stop bit, (860) 871-1988; 2400/3600 bps Courier HST. (860) 871-0549. World Wide Web. <http://www.circellar.com/>

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

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

Entire contents copyright © 1995 by Circuit Cellar Incorporated. All rights reserved. *Circuit Cellar INK* is a registered trademark of Circuit Cellar Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.



14 **Rediscovering Analog Computers**
by *Do-While Jones*

20 **Parallel Processing with Transputers**
by *David Prutchi*

36 **Developing a Virtual Hardware Device**
by *Michael Smith*

46 **Developing an Engine Control System**
Part 3: Completing the System
by *Ed Lansinger*

54 **Firmware Furnace**
Journey to the Protected Land: Real Interrupts in Virtual-86 Mode
Ed Nisley

62 **From the Bench**
Carrier Current Modem
Part 1: Communicating at 1200 bps Around the House
Jeff Bachiochi

70 **Silicon Update**
ADSP4ME
Audio Processor Chips for the Masses
Tom Cantrell

INSIDE ISSUE 64

2 **Editor's INK**
Ken Davidson
Weaving the Web

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

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

78 **ConneCTime**
Excerpts from
the Circuit Cellar BBS
conducted by
Ken Davidson

96 **Steve's Own INK**
They Still Flip Hamburgers,
Don't They?

81 **Advertiser's Index**

READER'S INK

IT'S GREEK TO ME

The information Do-While Jones offers in "Digital Filter Alchemy" (INK 61) is useful, but the symbols need to be defined in a list I can refer to.

One example is that squiggly Greek letter for damping factor (I think). (I never learned the English names for most Greek letters.) Another is those subscripted omegas (e.g., Ω_n, Ω_p). What frequencies are these! Where's the list!

Sayre **Rodman**
via the Internet

PERPETUATING COLOR MYTHS

Mike Bailey's "The Use of Color in Scientific Visualization" (INK 60) is a fine tutorial with one exception: he perpetuates the myth of the trichromatic retina. This is no doubt a result of the Second Law of Societal Inertia: An idea once accepted tends to remain accepted even when proven wrong. (This is the same reason astrology never died.)

It is now accepted that there are not three, but two kinds of cones. And yes, rods do contribute to neural color information, not just to night vision.

Color vision in the fovea is dichromatic meaning there are two kinds of cones there. Also, all efforts over centuries have failed to identify and confirm a third type of cone anywhere in the retina. Finally, many experimenters have shown that the source of neural hue information is the cooperation of the rods with the cones. My article on that subject appeared in the November 1977 issue of the *Journal of the Optical Society of America*.

My upcoming article, "200 Years of Color Vision Theories" in the quarterly international journal, *Speculations in Science and Technology*, lays it all out. (Watch for it. It's been accepted but not yet scheduled.) It turns out that the three color variables in the human-visual system are not red, green, and blue as Thomas Young proposed in 1801. The variables are actually luminosity, hue, and saturation as Helmholtz proposed around 1850.

Consider your own experience. When you look at a colored object, do you see it in proportions of red, blue, and green? No, not at all. You see it as so bright, having a particular hue, and a certain richness or saturation from vivid to pastel.

In the classic, color-television technological battle between CBS, RCA, NBC, and NTSC in 1951-54, NTSC won because it coded color into luminance (luminosity) and chrominance (hue and saturation) signals and found that only $\frac{1}{3}$ the bandwidth was required compared to the CBS red, green, and blue system. In the human eye too, coding in terms of luminosity, hue, and saturation requires much less bandwidth (i.e., one gets more data transmitted to the brain for the limited number of fibers in the optic nerve [about 1 million]).

Luminosity and hue information are developed in the retina, but saturation information is developed higher up. This conserves optic nerve fibers. Also, the same optic-nerve fibers carry hue information at high (photopic) light levels and rod-luminosity information at low (scotopic) levels, further conserving optic nerve fibers.

It now seems that red, green, and blue information as such, do not appear anywhere in the human-visual system!

Homer B. **Tilton**
Tucson, AZ

Contacting Circuit Cellar

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

Mail: Letters to the Editor may be sent to: Editor, Circuit Cellar INK, 4 Park St., Vernon, CT 06066.

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

Contact our editorial offices at (860) 8752199.

Fax: All faxes may be sent to (860) 872-2204.

BBS: All of our editors and regular authors frequent the Circuit Cellar BBS and are available to answer questions. Call (860) 871-1988 with your modem (300-14.4k bps, 8N1).

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

WWW: Point your browser at <http://www.circellar.com/>

NEW PRODUCT NEWS

Edited by Harv Weiner

VGA-TO-NTSC CONVERTER

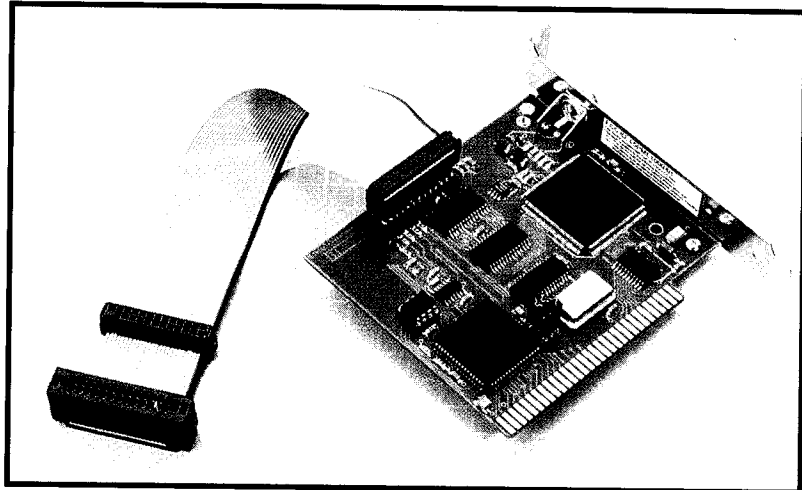
Methode Electronics announces **ME1000**, a new plug-in computer card. This card enables display of multimedia presentations on a standard television receiver or recorded on a VCR. The ME1000 (VGA2NTSC) converts VGA computer graphics, CAD, and multimedia presentations into standard NTSC video output.

The card automatically detects and switches between 640×480 , 320×200 , and text modes. The plug-in card works with any '386, '486, or Pentium computer using the ISA bus and a graphics card with a feature connector.

Additional features of the ME1000 include conformity to multimedia standards, automatic mode sensing and switching, flicker-free images, and compatibility with CAD, DOS, and Windows applications. The unit displays 256 colors and includes an internal cable.

Methode Electronics, Inc., dataMate Division
7444 W. Wilson Ave. • Chicago, IL 60656
(708) 867-9600 • Fax: (708) 867-3149

#500



SINGLE-CHIP PC SOUND SYSTEM

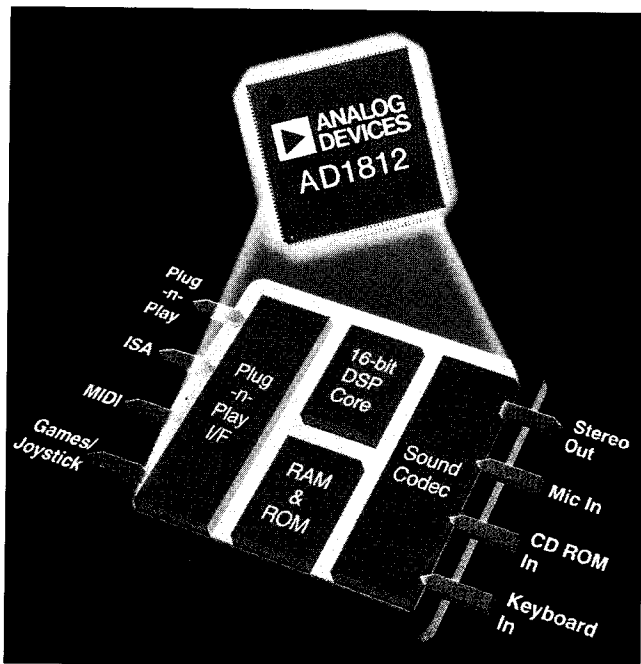
The **AD1812 SoundPort Controller** from Analog Devices enables developers to incorporate 16-bit, CD-quality sound into their PC products while building compatibility with various emerging audio standards. This single chip provides the functionality of five logical multimedia devices: 16-bit sound card, Windows sound-system CODEC, joystick port, MIDI sound, and music synthesizer. The chip offers quick and cost-effective integration of multimedia signal processing in PCs.

A 16-bit, fixed-point DSP core embedded in the AD1812 runs an audio-synthesis program which approxi-

mates the waveforms' output by FM synthesizer chips to provide compatibility with Sound Blaster software. In addition to the on-chip DSP, an integrated full-duplex stereo CODEC lets the chip be used as a signal-processing accelerator in PCs that support Intel's Native Signal Processing (NSP) initiative.

The AD1812's CODEC features a new signal-processing technology developed by Analog Devices, called *Continuous Time Oversampling* (CTO). CTO enables a variety of multimedia signals, including game audio, voice, music, video, and communications signals, to be converted from analog to digital and back to analog on a single chip. These signals are based on inherently different sample rates. Previously, conversion and synchronization required multiple chips, increasing design cost and complexity while decreasing performance.

The AD1812, targeted for PC and motherboard manufacturers, sells for \$25 in volume.



Analog Devices, Inc.
Three Technology Way
Norwood, MA 02062
(617) 937-1428
Fax: (617) 821-4273

#501

NEW PRODUCT NEWS

COMMUNICATIONS COPROCESSORS FOR BBS

GTEK announces the **SmartCard-8/128PC**, a high-speed communications card that lets a seven-node BBS run along with PlanetConnect service. Additional cards provide even more nodes on the same PC.

SmartCard-8/128PC has 128 KB of onboard buffer. Along with its built-in FOSSIL support, the card handles the normal load of a BBS and the continuous stream of data from a C- or Ku-band satellite receiver. SmartCard is transparent to PlanetConnect's software and requires no special drivers.

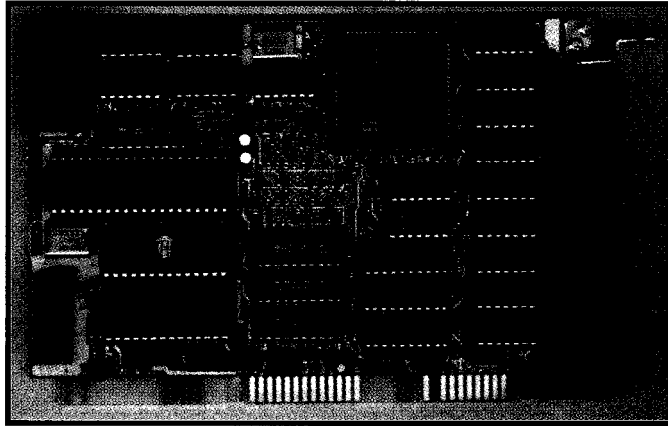
Standard features include eight high-speed communications channels, built-in FOSSIL driver and buffers (resulting in 43 KB more space for each BBS node), data rates up to 115,200 bps, full ten-conductor

RS-232C interface, and compatibility with all the popular BBS software that supports the FOSSIL command set.

The Guardian watchdog circuit constantly monitors the computer. If a problem results in a lock-up, the

Guardian hardware automatically resets the computer, enabling system operation in a walk-away, lights-out environment.

Installation is simple. The user inserts the card in the PC, configures the BBS software to work with FOSSIL, and sets up PlanetConnect. The SmartCard then takes over all communication, for PlanetConnect.



GTEK, Inc.

399 Highway 90 • Bay St. Louis, MS 39521

(601) 467-8048 • Fax: (601) 467-0935

#502

LIGHT-TO-FREQUENCY CONVERTER

Texas Instruments introduces two new light-to-frequency converters. The TSL235 is ideally designed for portable applications such as cameras, hand-held diagnostic equipment, and light meters. The TSL245 infrared (IR) converter is ideal for applications such as paper detection, proximity detection, and diagnostic equipment. Both devices directly convert light intensity to a high-resolution digital format.

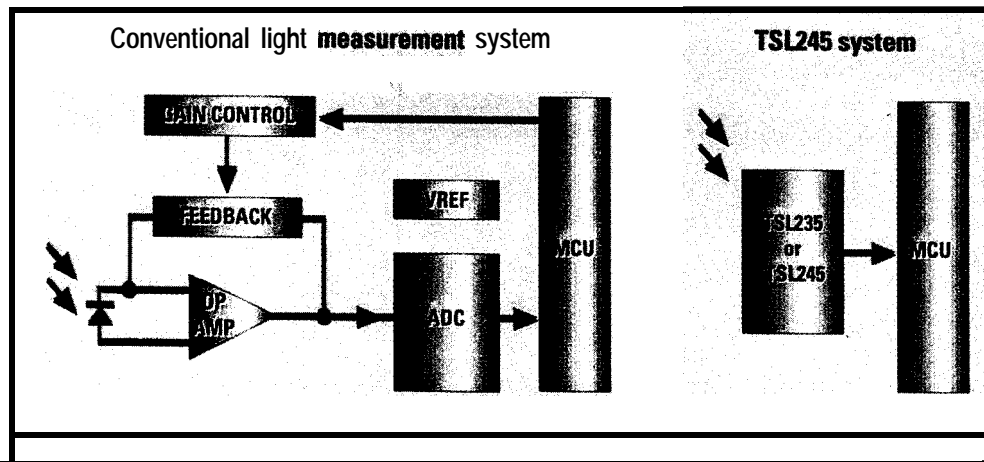
Interfacing directly to a microcontroller or DSP, the TSL235 and TSL245 combine a photodiode and current-to-frequency converter on a single chip. Both devices provide a simple way to process a wide dynamic range of light levels without external signal-conditioning circuitry or A/D converters.

The chips feature light-to-digital conversion with a pulse-train output, a dynamic range of 120 dB with 500-kHz full-scale output, a voltage operation of 2.7-5 VDC, and a TTL-compatible output with frequency directly proportional to light intensity on the photodiode.

The TSL235 and TSL245 sell for \$1.75 in 1000 quantity.

Texas Instruments, Inc.
Semiconductor Group
SC-95044
Literature Response Center
P.O. Box 172228
Denver, CO 80217
(800) 477-8924, Ext. 4500

#503

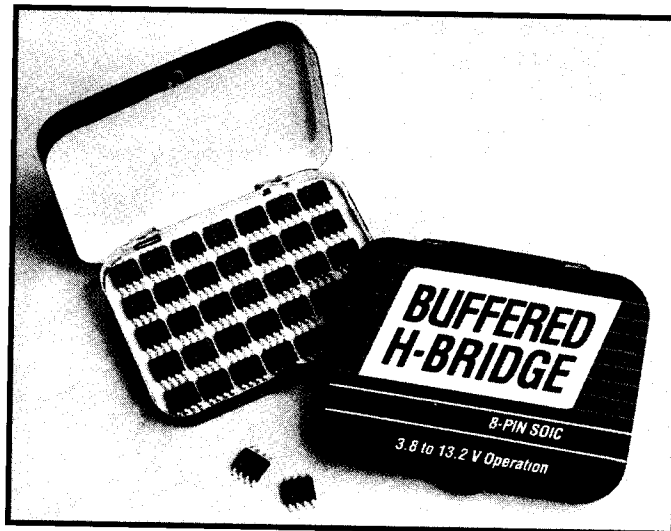


NEW PRODUCT NEWS

BUFFERED H-BRIDGE

Siliconix presents a buffered H-bridge that eliminates all the external discretes that prevent shoot-through in low-voltage brushed motor, stepper motor, and actuator applications. The Si9986CY integrates protection functions and MOSFETs onto a single chip. Target end products include optical disc drives, hard disk drives, scanners, printers, and other computer peripherals.

The Si9986CY delivers 1 A at switching rates up to 200 kHz in systems operating between 3.8



and 13.2 V. Packaged in a surface-mount 8-pin SOIC, the device dissipates up to 1 W, eliminating the need for heat sinks other than the

printed circuit board itself.

The chip integrates the logic control needed to prevent the upper and lower outputs of the internal half

bridge from turning on at the same time, which would cause a catastrophic failure of the circuit. Unique input codes force both outputs low for braking or to a high-impedance level.

The Si9986CY buffered H-bridge sells for \$1.86 in quantity.

Siliconix
P.O. Box 54951
Santa Clara, CA 95056-0951
(408) 970-4020
Fax: (408) 970-3995

#504

MPEG AUDIO ENCODER

Atlanta Signal Processors' **A1023 MPEG Audio Encoder Module** makes it easy to implement ISO MPEG audio compression. It provides high-quality digital-audio compression for mono and stereo signals in a completely self-contained module occupying less than 14 square inches of board space. The A1023 module has a balanced stereo-audio input. The audio goes to a stereo 16-bit A/D converter that can operate at 32, 44.1, or 48 kHz. It also has a glueless interface to digital-audio receiver chips. MPEG-encoded data output is either bit serial or through an 8-bit parallel FIFO. The module automatically handles SCR, PTS, and DTS timestamping for synchronization to a video bitstream.

The A1023 module offers external software control of operating parameters including sample rate, output bitrate, MPEG layer, audio-input source, and timestamping through an 8-bit bidirectional control interface. Default setups provide standard operation.

The A1023 module supports the Elementary Stream (ES) defined by ISO/IEC 11172-3 and the Packetized Elementary Stream (PES) defined by ISO/IEC 13818-1. The PES is directly compatible with the transport and program streams defined by ISO/IEC 13818-1 commonly referred to as MPEG-2. Either format may be selected through simple software control of the module. This feature easily merges the output of this module with other MPEG bitstreams to form a complete system stream of synchronized audio and video data.

Applications for the A1023 module include digital satellite transmission systems, video-on-demand systems, digital CATV systems, multimedia authoring systems, and other digital-audio applications.

The A1023 MPEG Audio Encoder Module costs less than \$400 in quantities of 250.

Atlanta Signal Processors, Inc.
1375 Peachtree St. NE, Ste. 690 • Atlanta, GA 30309-3115
(404) 892-7265 • Fax: (404) 892-2512 #505



Atlanta Signal Processors, Inc.

A1023

MPEG Audio Encoder

1375 Peachtree Street, NE, Suite 690 Atlanta, Georgia 30309-3115 USA
Internet: info@aspi.com

404 892-7265
Fax 404 892-2512
BBS 6 N 1 404 892-3200

NEW PRODUCT NEWS

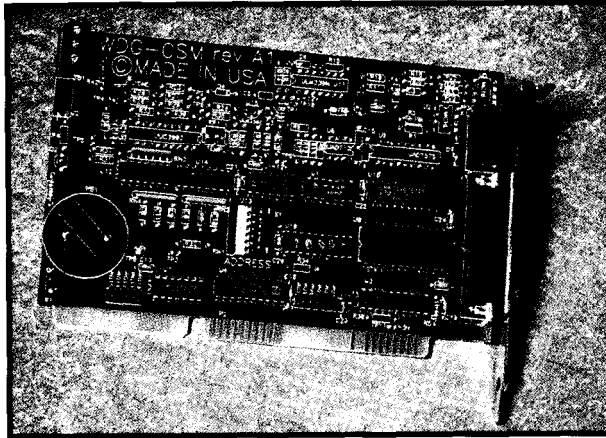
MULTIFUNCTION WATCHDOG TIMER

Industrial Computer Source introduces **WDT501-P**, a cost-effective multifunction watchdog timer card that includes a number of diagnostic functions relative to the host PC in which it resides. The WDT501-P offers the host computer excellent protection from temporary malfunctions.

In the event of a processor failure, program glitch, electrical noise, or component failure, a relay output and/or reset-line output may be used to reset the computer if reset does not occur through the user's application program. Timeout periods are software selectable from 10 ms to 80 min.

The WDT501-P augments its watchdog function with a number of internal host-computer diagnostics, including a

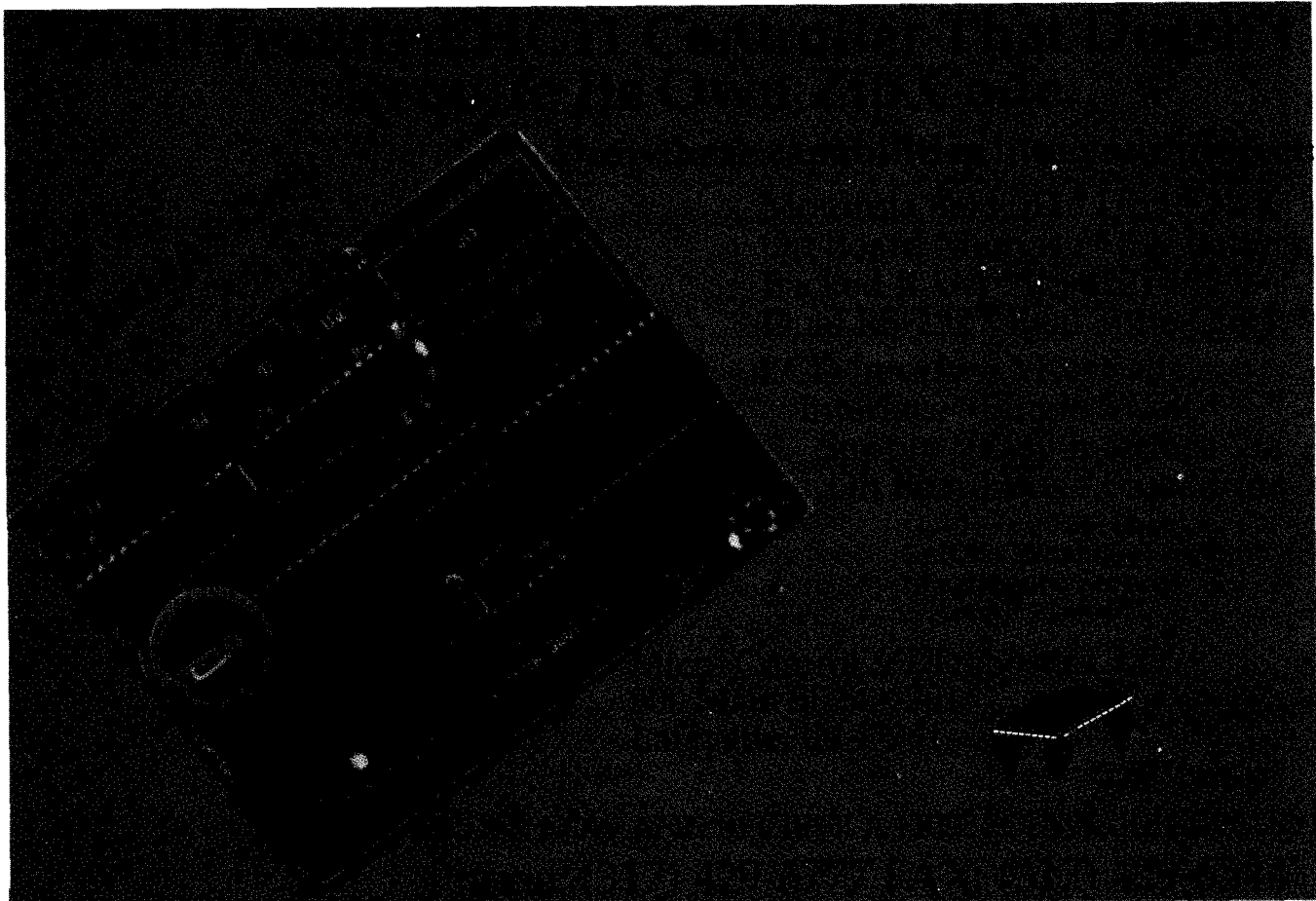
voltage monitor which verifies that all four computer voltages are within 7% of nominal, a temperature alarm that activates if the unit exceeds 50°C (adjustable), temperature monitor that can be read to within -14°C, watchdog timeout buzzer, fan-speed detector that triggers if fan-tachometer output drops below 50 Hz, and two isolated input and output lines for interrupts and computer reset.



The WDT501-P sells for \$195 and includes diagnostics, software, and manual. Model WDT500-P, a basic version without the diagnostics, sells for \$125.

Industrial Computer Source
9950 Barnes Canyon Rd.
San Diego, CA 92121-2720
(619) 677-0877
Fax: (619) 677-0895

#506



NEW PRODUCT NEWS

BATTERY-PACK TEMPERATURE MONITOR

Dallas Semiconductor introduces DS2434, a battery identification chip which monitors and reports battery temperature and charge status in portable electronics, hand-held instruments, and medical devices. The DS2434 is a digital-output temperature sensor that senses battery temperatures on-chip, eliminating the need for thermistors.

The DS2434 provides a convenient method of tagging and identifying battery packs by manufacturer, chemistry, or other identifying parameters. It stores battery-charge parameters, recharge history, and charge and discharge characteristics in its expanded 256-bit nonvolatile memory. Battery-pack temperature is monitored and reported digitally.

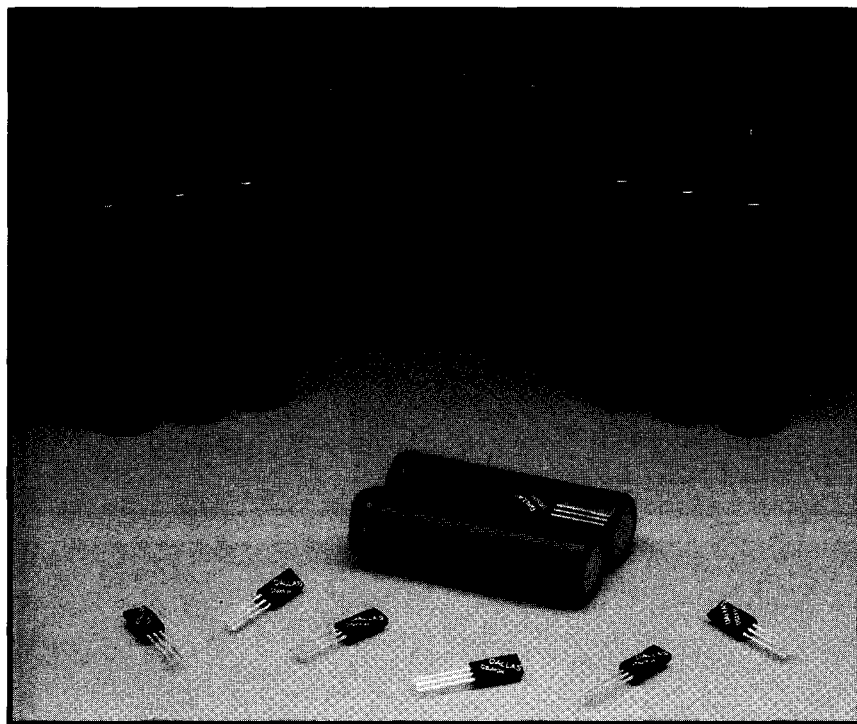
Battery manufacturers can assign an identification number to battery packs to aid in maintaining warranty information.

The expanded memory of the DS2434 offers sufficient storage capacity for user data such as gas gauge and manufacturing information and battery history. Information is sent and received via a one-wire interface so that the battery packs only need three output connections: power, ground, and the one-wire interface.

The DS2434 sells for \$2.98 in 1000 quantity.

Dallas Semiconductor Corp.
4401 South Beltwood Pkwy.
Dallas, TX 75244-3292
(214) 450-0448
Fax: (214) 450-0470

#507



PCMCIA READER/WRITER

Greystone Peripherals has introduced the GS-220F **CardDock**, a floppy combo that accommodates two PC card Types I, II, and/or III as well as a 3.5", half-height floppy disk drive in a single 5.25" drive bay on a desktop PC. The unit accommodates PCMCIA devices up to 15 mm and includes a patented eject mechanism, front-panel telephone jack, and convenient plug-and-play installation.

The two-socket CardDock floppy combo offers easy plug and play of, any memory and I/O PC cards designed to PCMCIA specifications. The CardDock is a totally compatible solution for reading and writing data or for exchanging data files and peripheral I/O functions between mobile and desktop PCs using PC cards.

CardDock software supports most flash-file software systems, which are provided with various flash cards, and protects critical data integrity, so PC cards can be moved from system to system and slot to slot. CardDock

software also manages system resources. It includes DOS- and Windows-compatible PCMCIA card and socket services software for configuring the system and each PC card. It also manages I/O ports, interrupt levels, and memory blocks. The software offers online PC card insertion and removal, an important feature for preserving data integrity and preventing data loss.

Each turnkey CardDock includes the two-slot docking bay hardware, a PC interface card (installs into any 16-bit AT-bus slot), cables, and software. The CardDock is easy to install and use and features LED read/write activity indicators.

Greystone Peripherals, Inc.
130A Knowles Dr. • Los Gatos, CA 95030
(408) 866-4739 • Fax: (408) 866-8328

#508

FEATURES

14

Rediscovering
Analog Computers

20

Parallel Processing
with Transputers

36

Developing a Virtual
Hardware Device

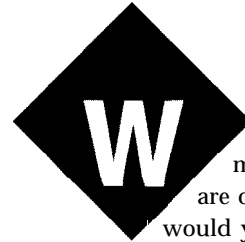
46

Developing an
Engine Control System

Rediscovering Analog Computers

FEATURE ARTICLE

Do-While Jones



When digital microcontrollers are on a chip, why would you use an analog computer?

Well...because there's still one thing that analog computers do better than digital computers—they solve differential equations.

And, since you can build an analog computer using just a few chips at a cost comparable to a microcontroller, there are times when an analog solution makes sense.

I rediscovered analog computers when I was writing "Cruising with Ada" [1], which demonstrated how to use Ada as an executable program design language. I illustrated my technique by designing an automobile cruise control. In the course of the article, I showed how to develop and test the cruise-control algorithm in Ada before translating the resulting program into assembly language for a microcontroller.

One of the points I wanted to make was that each project requires you to write a lot of support software. You need to take that into account when planning the project. In this particular case, the support software was a simulated automobile that tested how well the cruise control worked. It took more lines of code to simulate the automobile than it did to implement the cruise-control algorithm (i.e., it proved my pointed exactly).

Although I didn't say so in that article, I could have used an analog computer to simulate the automobile. An analog computer would have offered a more accurate simulation and would have been quicker to design and

In an increasingly digital age, it's easy to forget that the analog computer fills a special niche. Here Do-While shows us how an analog computer creates the best simulation for testing a cruise-control algorithm.

build. As well, I could have used the same analog computer to test both the Ada prototype and the microcontroller product. It would have let me verify the equivalence of two designs.

AN ANALOG EXAMPLE

The movement of an automobile body is determined by Newton's famous law: $F = MA$, where F is the total force acting on the body. In a simple simulation, F is the force applied by the engine (F_e) less the aerodynamic drag force (F_d). M is the mass of the body, which can be determined from the weight of the vehicle. A is the resulting acceleration of the body.

But, I really don't care about the acceleration. I want to know the velocity of the body because I am trying to control its speed. Since acceleration is the derivative of velocity, I can rewrite Newton's equation and solve for velocity:

$$\begin{aligned}
 F &= MA \\
 F &= M \frac{dv}{dt} \\
 \frac{dv}{dt} &= \frac{F}{M} \\
 v &= \frac{1}{M} \times \int F dt + V_0 \quad (1)
 \end{aligned}$$

So, to find the velocity, you have to integrate force with respect to time.

Figure 1 shows how to do this with an analog computer. A voltage representing the force applied to the body is multiplied by an inverting amplifier with a gain that represents the mass of the body. The resulting voltage represents $-F/M$. This voltage is applied to the input of an inverting integrator, which produces an output voltage that represents velocity. To give it the proper initial condition [i.e., the proper initial velocity], the integrator has an input that holds the output at the value V_0 until time = 0.

This crude model of an automobile was inadequate for my purpose.

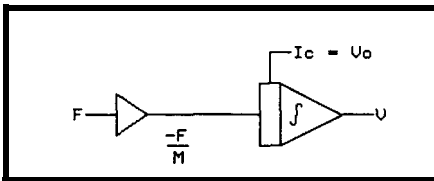


Figure 1—An amplifier and an integrator solve Newton's law as expressed in Equation 1.

Even the smallest engine eventually moves the car at an infinite velocity, which isn't very realistic. So, I added a simple drag model. I ignored friction and viscous drag, and assumed that all drag would be due to pressure drag, which is proportional to the square of the velocity. I assumed a top speed for the car, calculated the force the engine could produce at that speed, and picked a drag coefficient that equalled the engine force at top speed.

Figure 2 shows how drag can be added to the analog computer program. The velocity is applied to both the x and y inputs of an inverting xy multiplier, resulting in a voltage that represents the velocity squared.

To convert force to acceleration, the velocity squared has to be multiplied by a gain factor that represents the drag coefficient, and divided by a gain factor that represents the car's mass. Both of these gain factors can be combined in a single inverting amplifier. The drag acceleration has an opposite polarity of the engine acceleration, so the integrator combines the difference between the two accelerations to determine the output velocity.

This is such a simple program that one doesn't really need an expensive analog computer for it. No doubt, you could build it using an op-amp and an analog-multiplier chip. I'll bet you would use something that looks a lot like Figure 3.

If the program didn't have the drag (feedback) path, you would have to do something to keep the capacitor from integrating the input-offset voltage. But, since there is some negative feedback, you don't need to include the initial condition input. The output V goes to 0 in a few seconds if F is kept at zero.

SCALING VOLTAGES IS TRICKY

Conceptually, this is a very simple program [or circuit]. Determining the scale factors is the only tricky thing. The scale factors have to be chosen so their voltages don't exceed the supply voltages or be so small that they get lost in the noise. Furthermore, the time scale is a function of the amplitude scale factors. Skill is required to pick the scale factors.

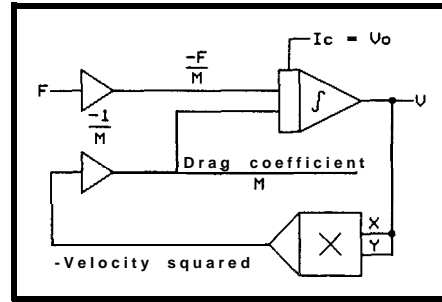


Figure 2—An analog computer program uses an analog multiplier to compute the effect of drag on the automobile's velocity.

To make it simple, ignore the drag portion of the circuit for a moment and consider just the portion of the analog computer shown in Figure 4. First, you need to know the range of values of the input force.

A 100-hp engine produces 54990 ft.-lb./sec. of power (i.e., 470 lb. of force at 117 ft./sec. or 80 MPH or 4,700 lb. of force at 8 MPH). Although theoretically it can produce 47,000 lb. of force at 0.8 MPH and infinite force when stopped, I am most interested in the performance of the cruise control when the speed is 40-80 MPH and the engine is producing 470-940 lb. of force.

So, if 1000 lb. equals 10 V and we are using a 15-V power supply, the engine force is limited to 1,500 lb. at low speeds. Similarly, if 10 V represents 100 MPH (146 ft./sec.), the supply voltage limits the car to 150 MPH. Both values should be acceptable.

In Figure 4, I need to pick resistor and capacitor values which correspond with these scale factors. If I pick a 0.1- μ F capacitor just because I have a drawer full of them, what's my value for R ?

If I apply a constant 1000 lb. of force to a car that weighs 2752 lb. (including the 200-lb. driver), the mass of 2752 lb. is 2752 divided by 32.2 ft./sec.² or 85.47 slugs. My acceleration is 1000 divided by 85.47 or 11.7 ft./sec.². So, after 10 sec. of acceleration, the car will be moving at 117 ft./sec. or 80 MPH.

I want a value of R so that if I apply 10 V (1000 pounds of force) to the input of Figure 4 for 10 s, the voltage across the capacitor is 8 V (80 MPH). The well-known equation for voltage across a capacitor is:

$$V = \frac{1}{C} \int i \, dt$$

Since the current is a constant (10 V / R Ω), I can pull the R value out of the integral:

$$8 \text{ V} = \frac{1}{0.1 \mu\text{F}} \times \frac{10 \text{ V}}{R \Omega} \times 10 \text{ s}$$

$$R = 125 \text{ M}\Omega$$

Since that's a larger resistor than I'm comfortable with, I'll let R = 1.25 M Ω and C = 10 μ F.

There is another way to make the component values more reasonable. If I scale the input so that 1 V = 1,000 lb. of force (instead of 10 V = 1,000 lb. of force), then R can equal 1.25 M Ω and C is 1 μ F. The normal input voltage then is 0.47-0.94 (rather than 4.7-9.4 V), which is well above the noise level. I could even let 1 V = 100 lb. of force, let R = 125 k Ω and C = 1 μ F. The input voltage at cruising speeds would be 47-94 mV.

It's difficult to select reasonable values for the capacitor, resistor, and voltage because the simulation has to run in real time. If I weren't restricted that way, I could let 10 V = 1,000 lb., R = 125 k Ω , and C = 0.1 μ F. The simulation would run 1000 times faster than real time, the capacitor would charge up to 80 MPH in 1/1000 of the time, and I could display the voltages on an oscilloscope and interpret milliseconds as if they were seconds.

Ironically, it is sometimes difficult to get digital simulations to run fast enough for real time, but with analog simulations, the problem is getting them to run *slow* enough. If you pick reasonable voltages and resistances, the capacitor values often have to be very large. It can be difficult (and expensive) to find precision large-value bipolar capacitors with low leakage. (Leakage decreases the voltage across

the capacitor, giving erroneous answers.)

If you pick reasonable voltages and capacitors, then the resistors often are very large. This problem leads to stray capacitance, leakage through dirt and condensation on the circuit board, and errors due to op-amp input-bias current and input-offset current.

If you pick reasonable resistors and capacitors, then the voltage levels have to be very small. You then have trouble with noise and DC offset.

I've been out of analog-circuit design for about 20 years, but I assume analog components have improved as much as digital components have. There must be op-amps today that have less noise and input current than the old Fairchild 741 had. Building an integrator that accurately simulates an automobile body in real time shouldn't be as hard with today's components as it was 20 years ago. But, with proper scaling, I could even use a 741 op-amp and still get results good enough for a real-time automobile simulator.

FINDING THE MISSING INPUT

The input to this analog computer is the force applied to the automobile body by the engine. However, the output from the cruise control is a throttle setting. How do you convert throttle setting to engine force?

Since this was a quick-and-dirty model to prove a concept, I assumed that engine power (not force) is linearly proportional to the throttle setting. This is almost certainly not true, but I'm not an automotive engineer so don't know what the relationship is.

It is more likely a curve with nasty discontinuities when the transmission shifts gears. However, over the cruising range, when the transmission stays in one gear, a linear approxima-

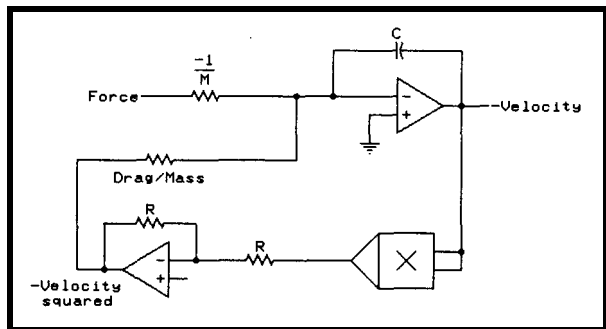
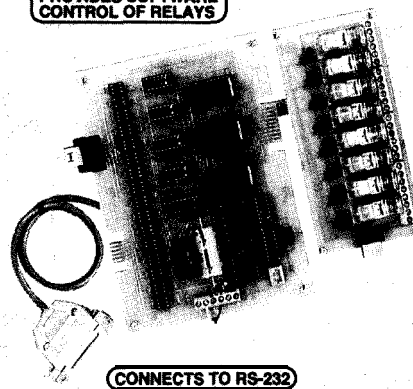


Figure 3—The analog computer program from Figure 2 can be built from two op-amps and an analog multiplier.

RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

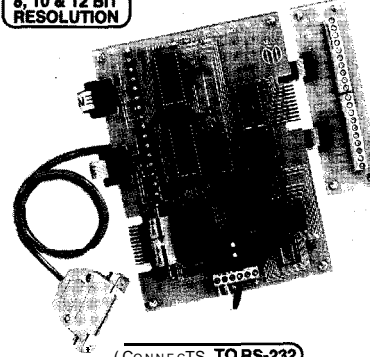


CONNECTS TO RS-232

AR-16 RELAY INTERFACE (16 channel).....\$ 69.95
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).....\$ 99.95
ADC-8G A/D CONVERTER (8 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 (8 ch).....\$ 139.95
Includes term. block & 8 temp. sensors (-40' to 146' F).
STA-8 DIGITAL INTERFACE @ channel).....\$ 99.95
Input on/off status of relays, switches, HVAC equipment, security devices, smoke detectors, and other devices.
STA-8D TOUCH TONE INTERFACE.....\$ 134.90
Allows callers to select control functions from any phone.
PS-4 PORT SELECTOR (4 channels RS-422).....\$ 79.95
Converts an RS-232 port into 4 selectable RS-422 ports.
CO-485 (AS-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 126 temperature inputs using the PS-4, EX-16, ST-32 & AD-16 expansion cards

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

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

CONNECTS TO RS-232, RS-422 or RS-485...use with IBM and compatibles, Mac and most computers. All standard baud rates and protocols (50 to 19,200 baud). Use our 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.
360 South Fifth Street, Suite 604
Columbus, Ohio 43215-5438

tion is probably good enough. [If it's not good enough, a look-up table in the cruise-control algorithm can compensate for the curve and make it linear.] So, I assume the cruise control output is power.

Force is power divided by velocity. However, analog-division circuits are much less common than analog multipliers. A multiplier does division by putting it in a feedback loop.

In Figure 5, I assume that the force voltage comes from a magic, unknown source (which I will create later). I apply force and velocity to the two inputs of an inverting analog multiplier. Since the product of force and velocity is power, the output of the inverting-multiplier circuit represents negative power.

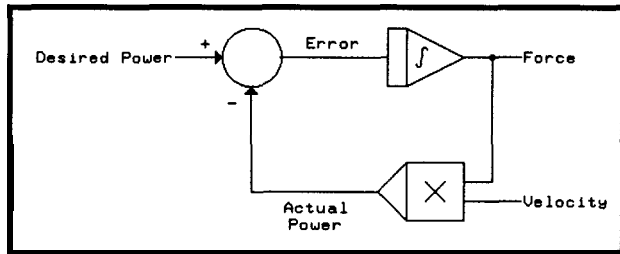


Figure 5—Feedback determines the force necessary to create the desired power.

I put the negative power into a summing junction that combines it with the desired power input. If the power exactly equals the desired power, they cancel each other. As long as there's no error, the integrator holds the force at the same correct value.

If the power and desired power don't exactly cancel each other out, an error voltage is produced. This error is applied to an integrator, which is the magic source for the force. (Use a small capacitor in this integrator so it's much faster than the rest of the time constants and doesn't affect the response of the circuit.) As it turns out, the polarity of the error is backwards, so an inverter has to be added before or after the integrator.

Figure 6 shows how this inverter can be added to the circuit in Figure 3. I could have built a crude automobile simu-

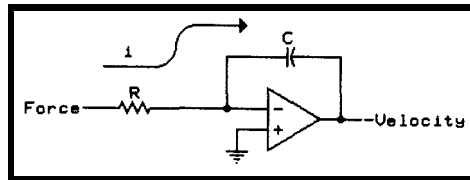


Figure 4—Values must be chosen for R and C to satisfy the scale factors being used.

lator using two op-amps and two analog multipliers instead of simulating it in software. Figure 6's crude analog computer is no worse than the digital simulation, which uses the same simplifying assumptions (i.e., drag is proportional to velocity, and power is proportional to throttle position), but was good enough for proof of concept.

If I need to have a better model, I must measure the transfer functions of the components of the body. When I do this, the open-loop transfer function is expressed in the frequency domain as a polynomial in seconds. I could write a digital filter with corresponding characteristics [see "Digital Filter Alchemy," INK 61] and put it in the digital simulation.

But, it would be just as easy, if not easier, to design an active analog filter with the same transfer function.

In general, as the requirements for a simulation become more severe, the digital solution gets more difficult at a faster rate than the analog solution does. This was particularly true in the cruise control example because of the time-frequency problem.

THE TIME-FREQUENCY PROBLEM

In "Cruising with Ada," one of the points I make is that software projects don't fail because of too much coupling, too little cohesion, improper indentation, or other things computer scientists worry about. It's true: bad software engineering practices do increase the number of defects, development time, and maintenance costs, so should be avoided. But, crummy code doesn't usually cause project failure.

A project usually fails because a fundamental problem isn't discovered until operational testing, which is generally at the end of the project. I encourage rapid prototyping because the sooner you get something operational, the earlier you discover the killer problem. In the cruise control, the problem was the time-frequency uncertainty principle.

Suppose you measure speed by putting a magnet on one of the axles and using a sensor that produces one pulse per wheel revolution. The outside diameter of the tires on my truck is 23.5", so the circumference is 73.8". Since 1 MPH is 17.6 in./sec., a vehicle moving 1 MPH produces 0.238 pulses/sec. If the cruise control is specified to work from 40 to 80 MPH, there's 9.5-19 pulses/sec.

I designed the control loop to measure speed every 100 ms. (I picked this value because I know guided missiles, which go 10 times faster than an automobile, can be controlled with a 10-ms cycle-control loop.)

During the sample period, 80 MPH produces 1.9 pulses, which isn't a very useful measurement of speed! Even if you increase the sample period

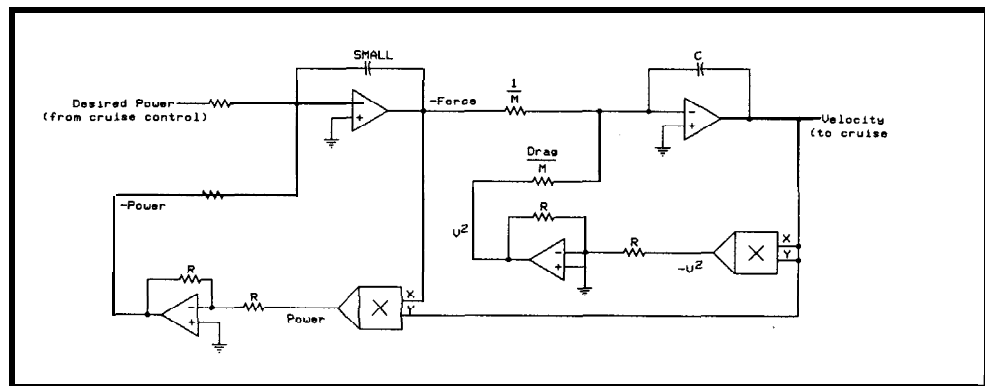


Figure 6—An analog computer circuit can be used to compute automobile velocity in response to cruise control commands. It uses four op-amps and two inverting xy multipliers.

to 200 ms, you only get 1, 2, or 3 pulses during the sample period, which translates to 21, 42, or 63 MPH. The control loop can't possibly give satisfactory performance with such a coarse measurement.

You can measure average speed more accurately if you count the number of pulses in a 10-s period. Then, 130 pulses corresponds to 54.5 MPH, 131 pulses to 55.0 MPH, and 132 pulses to 55.5 MPH. Although this gives the average speed to 0.5-MPH resolution, there is some uncertainty about the instantaneous speed. The control loop is likely to be unstable because vehicle speed changes too much in a 10-s period. The trick in getting a cruise-control solution is to make the proper tradeoffs between measuring speed quickly and accurately.

I'm sure you can think of several different ways you might do this (e.g., add more magnets, use the pulses to gate a high-frequency oscillator, etc.). Whatever you do, you have to simulate it accurately because it is critical to the performance of the cruise control.

I didn't carry the example any further in "Cruising with Ada" because of the difficulty of simulating an FM-modulated pulse train on a digital computer. With digital, the period of the modulation frequency can be less than the computational cycle of the cruise control. To convince myself that I had correctly simulated the response of the speed-sensing circuit, it would have required a lot of testing.

But, if I had used an analog computer, it would have been trivial. It would take most of you less than an afternoon to design a linear voltage-controlled oscillator that produces 9.5 pulses per second when the input voltage is 4.0 V (40 MPH) and 19 pulses per second when the input is 8.0 V.

DON'T OVERLOOK THE SIMPLE

I once heard a story about an incident that allegedly happened in Thomas Edison's research lab. The story may not be true, but its moral is.

Edison was improving his light-bulb design by trying different filaments, gases, and shaped bulbs. He asked his assistant to find the volume of an odd-shaped blown-glass bulb.

Several hours later, Edison asked what the volume was. The assistant showed him sketches of the bulb's outline, the function used to approximate the outline, and several pages of calculus used to find the volume of a solid of revolution by parts.

Without saying a word, Edison picked up the bulb, walked to the sink, filled the bulb with water, and dumped it into a measuring cup.

Obviously, a measuring cup does *not* eliminate the need for calculus. It simply illustrates how we can overlook simple solutions because we're accustomed to using the difficult ones.

Similarly, I'm not saying analog computers eliminate the need for digital computers. I just want to point out that the analog computer is a useful tool often neglected today. Many younger engineers assume they are obsolete.

So, it is worthwhile to remind everyone that the analog computer is still viable. It is a measuring cup that sometimes gives you the answer faster and more accurately than modern methods.

For problems that are difficult on a digital computer, especially if it involves differential equations or an analog simulation, consider using a general-purpose analog computer (or build a circuit that is a special-purpose analog computer). An analog solution could save you months of design time and give you more accurate results. ☐

Do- While [ones has been employed in the defense industry since 1971. He has published more than 45 articles in a variety of popular computer magazines and has authored the book Ada in Action. He may be reached at do_while@ridgecrest.ca.us.

REFERENCE

- [1] Do-While Jones, "Cruising with Ada," *Embedded Systems Programming*, 1994.

I R S

- 401 Very Useful
402 Moderately Useful
403 Not Useful

ATTENTION COMPUTER STORES!

PUT Circuit Cellar INK TO WORK FOR YOU!

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

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

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

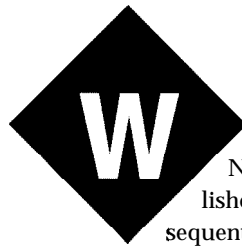
**(Circuit Cellar INK
Dealer Sales Program
4 Park St.
Vernon, CT 06066
(860) 875-2199**

FEATURE ARTICLE

David Prutchi

Parallel Processing with Transputers

Uniprocessor technology is at a limit. Greater speeds require new technology. Instead of paying the increased costs, David advocates parallel processing. Find out how to build your own transputer.



When John Von Neuman established the basics for sequential computer architectures in 1947, he simplified his solution to its very basics—a series of primitive numeric manipulations executed on data stored in a memory system. Each manipulation was carried out, one at a time, by a centralized

processor. Since then, major improvements in computational throughput have been achieved by using more ample instruction sets with wider data and address buses and by increasing system clock speed.

However, little can be done to speed up a system already running at full tilt. Current clock speeds bordering on 200 MHz already present difficult design and layout problems that heavily affect a system's flexibility, performance, and price. Further increases in clock speed may require sophisticated semiconductor materials (e.g., gallium-arsenide) and specialized interconnection technologies with prohibitive costs.

Ultimately, even if all other issues are solved, uniprocessor machines are limited by signals that can't travel faster than the speed of light across the finite dimensions of the processor.

Using multiple processors, tightly or loosely coupled, to share the workload achieves higher computational power without raw increases in speed. This strategy, called *parallelism*, can be exploited in three ways:

- Algorithmic—the algorithm is broken down into a pipeline of multiple processes

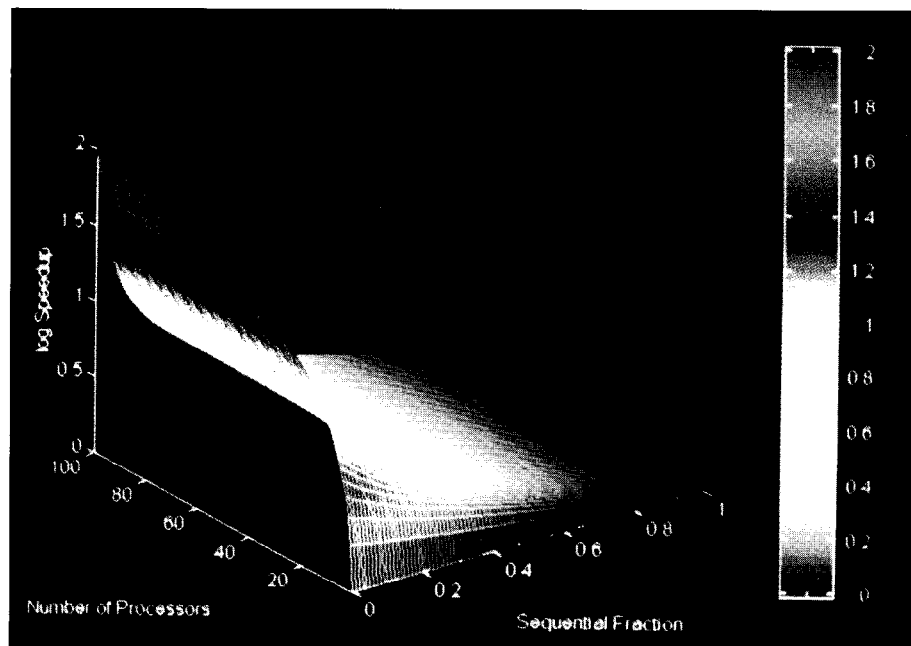


Photo 1—A plot of the logarithmic (10) maximum speedup that can be achieved through an ideal parallel computer demonstrates that effective use of a multiprocessor machine can only be achieved through a drastic reduction in the percentage of time spent executing sequential code. For large sequential fraction values, even an infinite number of processors only achieves modest performance improvements.

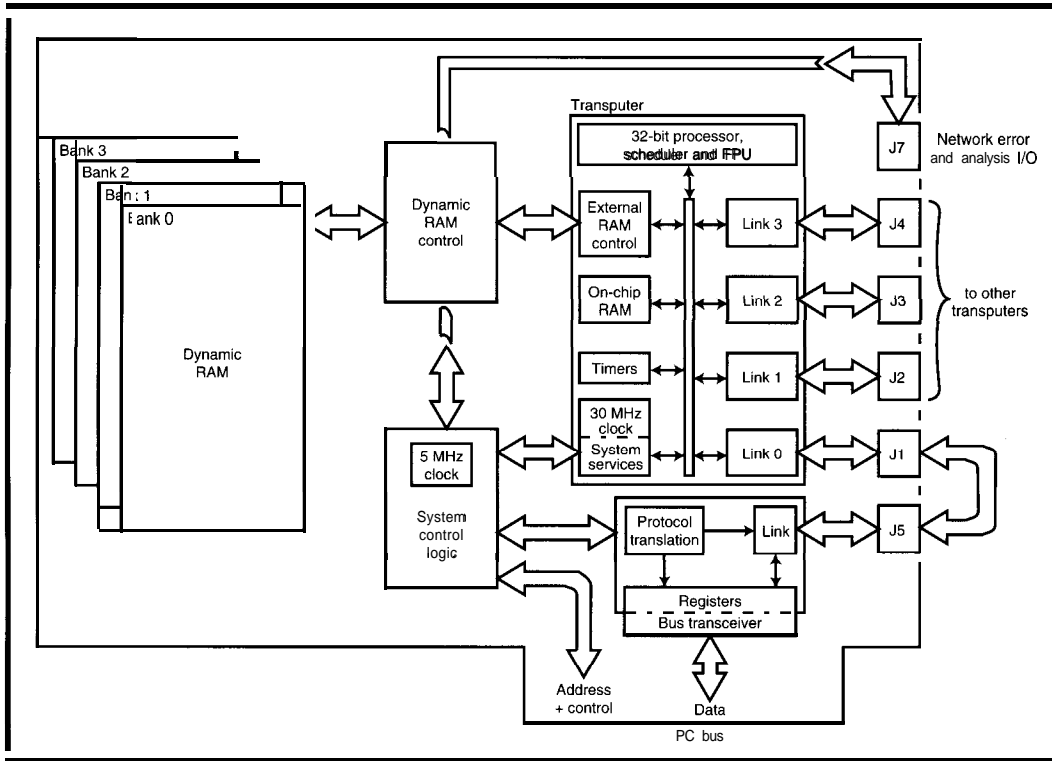


Figure 1—The PC add-in transputer card features a 32-bit transputer, up to 4 MB of DRAM, and three high-speed links for connecting to an external transputer network. The card also offers a hierarchical network analysis structure to aid the development of multitransputer systems.

as the Pentium, i860, or i960. By sharing system resources, these processors achieve performances in the hundreds of MIPS range. At the same time, companies such as Chips and Technologies are introducing multiprocessor chipsets that address many of the hardware design issues for shared-memory multiprocessor systems. The CS8239 chipset, for

- Geometric—the problem is broken down into a number of similar processes, each with a different subset of the total data to be processed. The processes communicate as they need access to data assigned to another process.
- Farming—the workload is “farmed out” by a master controller to other computing servers. The master dispenses new work to the servers as they become free.

Farming automatically balances the workload among the network servers regardless of network topology. It is limited only by the rate work can be dispensed and results handled. Unfortunately, farming is effective only when the problem can be divided into small, similar pieces, which represents only a small portion of the work demanded from a general-purpose computer.

Geometric parallelism introduces a major system design difference between Von Neuman sequential problem solving and parallel computation, namely the topology of the parallel processors’ network. In principle, the processors need to be connected in such a way that the network topology somehow models the structures inher-

ent to the problem. If a good model is found, the partitioning of the algorithm is simple to understand and implement.

Parallel machines based on specialized ICs have been developed experimentally for more than a decade. A number of parallel-processing computers and supercomputers are available, but only for large budgets. Lately, however, parallel processing chipsets have been designed that may launch the personal computer into affordable desktop supercomputing.

Intel and others offer parallel processing boards with a small number of scalable-architecture processors, such

instance, interconnects up to six ‘486 microprocessors to a fast, wide, non-multiplexed bus that permits multiple masters.

Most of these efforts are inherently limited because bus-based, shared-memory computers (regardless of the number of processors they use) are limited in practical scalability by memory contention and bus bandwidth. Implementing distributed memory systems instead of shared memory schemes avoids most scalability limitations. However, processors with distributed memories require communication to effectively exploit geometric parallelism. This shift in the

<p>lptr—This instruction pointer acts as a conventional program counter. It points to the next instruction to be executed.</p> <p>Wptr—This workspace or stack pointer points to a storage area of local variables.</p> <p>Areg, Breg, Creg—These general-purpose registers form a push-down stack (Areg on top), so the transputer can use zero- and one-operand instructions. Because the stack is not large enough to store variables of lengthy operations, the transputer’s assembly-language programming usually demands extensive use of load and store instructions, much like single-accumulator microprocessors.</p> <p>Oreg—The operand register assembles the operands used by direct instructions.</p> <p>Error Flag—This flag approximates a traditional overflow flag. Once triggered, however, it remains set until explicitly cleared. The state of this flag is presented to the transputer’s Error pin, and the location of an errant transputer may be located in a multitransputer network.</p> <p>Halt-on-error Flag—When set, this flag causes the setting of the error flag to be interpreted as a fatal error. The whole system comes to a complete stop.</p>
--

Table 1—The transputer has six internal registers and two flags which define the state of a sequential process.

architectural paradigm can block development of parallel processing desktop supercomputers.

Already, there's a battle to capture the desktop supercomputer market between Inmos (now owned by SGS-Thomson Microelectronics) and microprocessor industry giants. Since 1985, Inmos has offered a *transputer*, which is a microprocessor that gets its power from the radically different philosophy that underlies its design, although it barely performs a la pair with Intel and Motorola processors.

Transputers communicate with other transputers in parallel processing networks using minimal interconnection and communications overhead. The same level of parallelism can be executed on a network of devices as within a single transputer executing virtual concurrent processes through hardware-scheduled sharing of the processor time.

After looking at the principles of parallel processing hardware and soft-

ware, this article presents first-generation transputers in detail and a simple PC add-on implementation. A brief peek at Occam, the transputer's native language, is followed by a look at second-generation transputer products and the possibilities for desktop parallel computers that can stand up to the performance of multimillion dollar supercomputers.

PARALLEL PROCESSING

Different approaches to the design of parallel-processing computers have been identified to break the processing-speed limitations of sequential architectures. Essentially, these approaches aim to overcome the Von Neuman bottleneck of Single-Instruction Single-Data (SISD) computers.

Parallel architectures attempt to gain power by performing the same mathematical operation on a number of data elements simultaneously (Single-Instruction Multiple-Data—SIMD) or by assigning multiple unique

tasks to many processors in parallel (Multiple-Instruction Multiple-Data—MIMD). SIMD is typical for vector or array processors while MIMD is the basis for more flexible parallel computers because it can work efficiently over a wide range of granularity.

There is even a hybrid of these two architectures called Single-Program Multiple-Data (SPMD). A copy of the same program runs on each processor, even though the programs are not synchronized at the instruction level.

The efficiency of each approach may be estimated using Amdahl's law, a mathematical formula which assumes that a computing process can be divided into a sequential and parallel portion [1]. Besides the parallel operations which may be distributed over a number of processors, there remains a sequential portion, comprising at least the sequential communications between the processes. The sequential component limits the efficiency of a parallel machine.

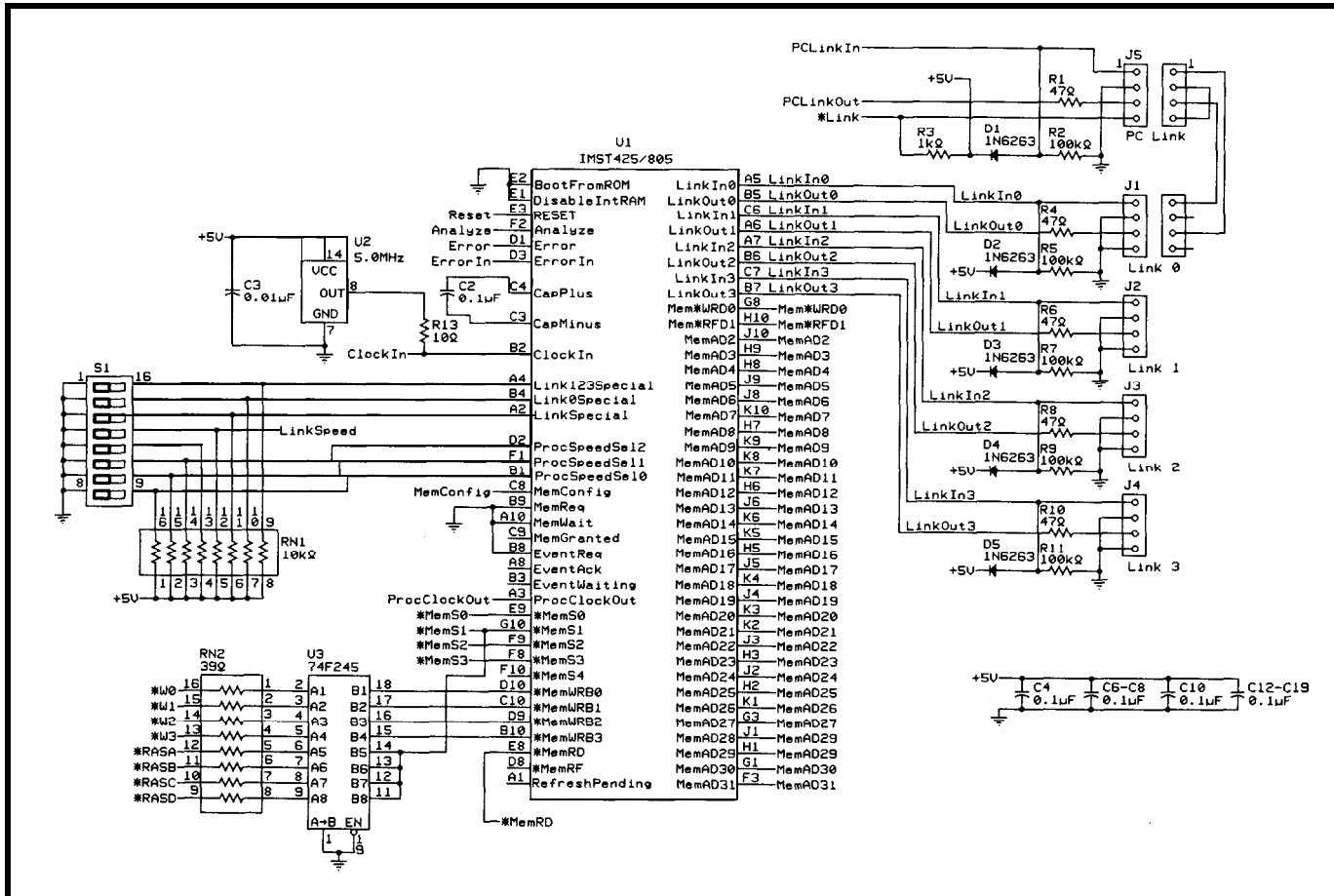


Figure 2—The heart of the add-in card is a T425 or T805. These transputers communicate with the host PC and other transputers via four high-speed bidirectional serial links. The processor clock is internally generated by a PLL oscillator locked to U2's 5-MHz clock. Processor and link speeds are selected through DIP switch bank S1.

Under ideal conditions where no overhead for communications and synchronization is required, the maximum speedup attainable by converting a sequential program to a parallel implementation is expressed by Amhdal's law as:

$$\text{Speedup} = \frac{T_1}{T_n} \quad (1)$$

where T_1 is the run time of the sequential version, while T_n is that of the parallel implementation, and n is the number of processors. T_n is determined by the fraction of the total time spent executing sequential code s and by the number of available processors as defined by:

$$T_n = sT_1 + \frac{T_1(1-s)}{n} \quad (2)$$

Photo 1 displays speedup under ideal conditions as a function of the number of processors and percentage of time spent executing sequential code. The point is obvious—the major enemy to speedup is sequential processing. For large values of s , time loss is so significant that an infinite number of processors only achieves modest performance increases.

In more realistic conditions where workload is not perfectly balanced and communications and synchronization requires overhead, there's an additional toll to the theoretical speedup.

In recent years, the interpretation of Amhdal's law has been frequently challenged. More conservative views contend that Amhdal's law applies to all processing systems and thus describes a more general limitation on the performance of any programmed system above a certain level of complexity. Under this view, a good sequential or parallel system design should loosen the bounds imposed by Amhdal's law to the point where the system becomes feasible [2].

A more radical view disputes the validity of Amhdal's classical basis by arguing that program execution time rather than problem size is constant. From this perspective, speedup is achieved by having the software design the hardware it needs. This idea recently led to the design of a massively reconfigurable logic computer in

which the hardware—676,000 gates in 52 FPGAs—is tailored through the software to exactly fit the algorithm. Virtual Computer Corporation, the developers of this machine, expect speedups well beyond those predicted by Amhdal's law.

In any case, Amhdal's law conveys the "catch" of parallel computing—different kinds of multiprocessing systems suit different kinds of applications. The designer must decide what is best for the problem at hand [3].

As a first step, assess optimal granularity. Although some algorithms run more efficiently at fine-grain level (simultaneously executing many different microinstructions such as move, add, compare, write, etc.) where multiple parallel processors can be accommodated with ease, the organizational overhead of communications and synchronization may consume the speed gains of parallelism.

However, using coarse-grain parallelism (simultaneously executing subroutines) doesn't ensure success. Tasks that could be executed in parallel may

remain locked within subroutines, so some processors remain idle for large amounts of the computing time.

To best develop a multiprocessor system, the designer needs to thoroughly understand the application and then develop a well-structured program that is highly modular and easy to partition. Many engineers experienced in parallel programming first develop and debug the algorithm in a single-task version before attempting to deal with communications, synchronization, and resource sharing.

The structured program should be carefully analyzed to identify how to best distribute the tasks. To achieve the desired speedup, optimization aims to balance the load between parallel processes, while reducing communications and synchronization requirements. If performed correctly, this step provides a clear view of the hardware and software topology and the level of granularity that best exploits parallelism for your application.

The designer should also identify how data and code distribution can be



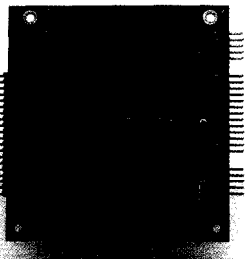


Sets the Pace in PC/104 CPU and DAS Technologies



33 MHz
486 SLC
\$496

- Features:**
486SLC
33 MHz
387SX
2MB DRAM
SSD
EEPROM
WDT
IDE
FDC
2 Serial
EPP
PS/2 mouse
Keyboard
Quick boot
Virtual devices



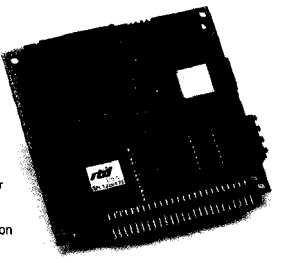
CM1486SLC-1 Fully Integrated PC-AT with Virtual Device Support

When placing your order, mention this ad and receive a 387SX math coprocessor FREE!



200 kHz
12-bit DAS
\$498

- Features:**
200 kHz
12-bit A/D
16SE/8DI
Scan
Burst
Multiburst
1K CGT
Triggers
FIFO buffer
Data marker
12-bit D/A
16-bit DIO
+5V operation



DM5408-2 200 kHz Analog I/O Module with Channel-Gain Table

Make your selection from:

9 cpuModules™

XT, SuperXT™, 386SX, 486SLC, 486SLC2, 486SX, 486DX, and 486DX2 processors. SSD, 8MB DRAM, RS-232/422/485 serial ports, parallel port, IDE & floppy controllers, Quick Boot, watchdog timer, power management, and digital control. Virtual devices include keyboard, video, floppy, and hard disk.

7 utilityModules™

SVGA CRT & LCD, Ethernet, keypad scanning, PCMCIA, intelligent GPS, IDE hard disk, and floppy.

18 dataModules®

12, 14 & 16-bit data acquisition modules with high speed sampling, channel-gain table (CGT), sample buffer, versatile triggers, scan, random burst & multiburst, DMA, 4-20 mA current loop, bit programmable digital I/O, advanced digital interrupt modes, incremental encoder interfaces, opto-22 compatibility, and power-down.

&Real Time Devices USA

200 Innovation Boulevard • P.O. Box 906
State College, PA 16804-0906 USA

Tel: 1 (814) 234-8087 • Fax: 1 (814) 234-5218
FaxBack®: 1 (814) 2351260. BBS: 1 (814) 234.9427

RTD Europa RTD Scandinavia

Budapest, Hungary Helsinki, Finland
Fax: (36) 1 212.0260 Fax: (358) 0 346.4539

RTD is a founder of the PC/104 Consortium and the world's leading supplier of PC/104 CPU and DAS modules

DIP Switch			Processor Speed (MHz)	
S1-8 (ProcSpeed Select0)	S1-7 (ProcSpeed Select1)	S1-6 (ProcSpeed Select2)	T400	T425/T805
0	0	0	20.0	20.0
0	0	1	22.5	22.5
0	1	0	25	25.0
0	1	1	30	30.0
1	0	0	35	35
1	0	1	Invalid	Invalid
1	1	0	17.5	17.5
1	1	1	Invalid	Invalid

Table 2-Processor-speed selection is available in discrete steps. Clock frequency is programmed through DIP switches S1-6 through S1-8 up to the maximum rated frequency for a particular device.

carried out independently. This often results in a network which can't be described as a pure processor farm, geometric array, or algorithmic pipeline. A simulation of a certain physical phenomenon may have a geometric processor array to model the physical system, each of which feeds a pipeline simulating a local physical process. These in turn may own a farm of processors which care for math operations requiring minimal interaction.

TRANSPUTER BASICS

The transputer is a RISC computer on a chip, complete with a high-speed CPU, memory, external memory controller, and four full-duplex communication links. Some models also include an on-chip floating-point unit (FPU). The transputer's architecture achieves optimal virtual and true concurrent processing under Occam.

Occam enables transputers to be described as a collection of processes operating concurrently and communicating through channels. It implements the Communicating Sequential Processes (CSP) model of parallel computing. It considers a parallel program to be the same as a finite number of sequential processes which execute concurrently while exchanging messages over message channels.

As a processing unit, the transputer's integer CPU calculates address information and presents the FPU with data. The CPU efficiently supports the Occam model of concurrency and communication. A reduced number of instructions form the transputer's instruction set and make concurrent processing as efficient as possible.

A stack-based architecture uses on-chip RAM as a conventional micro-processor uses its registers. Concurrent tasks map their "registers" to specific workspace in the on-chip RAM. Table 1 describes the internal registers and two flags which define the state of a running process.

Switching between concurrent tasks is as simple as switching a pointer to the correct workspace. An operating-system kernel is built into the transputer to execute multitasking under direct hardware control. Hence, any number of concurrent processes can be executed together, sharing the processor time. Hardware-controlled scheduling eliminates the need for external software kernels and speeds context switching to 0.6 μs.

The hardware-process scheduler automatically sleeps processes waiting for channel I/O and wakes them at I/O completion. The instruction set also includes an Alternative command (A LT) which makes a process dormant until it receives an alternative enabling event. Two interval timers and time-out support also keep processes dormant until they're needed.

Communication between concurrent processes takes place through channels when both the input and the output processes are ready. This message-passing channel construct lets processes share data and become synchronized. Communication between processes on the same transputer takes place through local-memory channels. When processes run on different transputers, communication takes place through channels implemented on the high-speed serial links. Each link is a

fast (20-Mbps), asynchronous, full-duplex channel.

In addition to the links, the built-in memory controller communicates with external memory and peripherals. The memory controller expands the address space off chip, and can directly control up to 4 GB of DRAM. It also maps I/O space for interfacing with other peripherals. Since these modules operate simultaneously, asynchronous communications between processes demands minimal overhead.

TRANSPUTER SPECIES

Transputers operate with clock rates of 15, 20, 25 and 30 MHz, and different transputer families fulfill the requirements of different markets and applications.

The T2xx comprises a family of 16-bit transputers. These "baby transputers" have 64 KB of memory space and lack many features of their 32-bit counterparts. Because of their low cost (-\$80 in singles for the 30-MHz model), they are especially attractive for parallel-processing embedded applications which do not require high-precision arithmetic or extensive memory.

The T2xx is often used as a signal-acquisition controller or preprocessor within systems that use more powerful transputers as their main processors. As you'll see later, integrating other types of transputers is possible because the internal register pointer architecture and link protocols work with different word lengths.

The T4xx transputer family contains 32-bit microprocessors, and implements all the features of the T2xx family. However, the T425 has four serial links and 4 KB of on-chip SRAM, while the T400 has only two serial links and 2 KB of on-chip SRAM.

The T8xx is essentially the same as the T4xx family, except it has an on-chip floating-point coprocessor. The T800, recently replaced by the improved T805, is pin-compatible with the T425. But, with the aid of its coprocessor, it is capable of delivering over 4.3 MFLOPS and 30 MIPS peak (@ 30 MHz) on its own.

notMemS0 (*MemS0)—Address latch enable
notMemS1 (*MemS1)—Row address strobe (*RAS), which is selected during all memory and I/O access operations
notMemS2 (*MemS2)—Row/column address multiplexing
notMemS3 (*MemS3)—Column address strobe (*CAS)
notMemS4 (*MemS4)—Not used
notMemRf (*MemRf)—Not used
notMemRd (*MemRd)—Not used for memory access; used to control the I/O PAL
notMemWr (*MemWr)—Write control to all memory chips

Table 3—Strobe lines are used for external memory interface.

The T9000 second-generation transputer is the latest addition to the transputer family. This new device sports numerous hardware enhancements which increase speed and support advanced operating systems while maintaining upward compatibility with the T805 instruction set.

More importantly, each of the T9000 links is connected to multiplexing hardware. Communications among processes in separate transputers takes place along as many channels as required. Shared physical links are software transparent. Much more flexible

parallel programs can be implemented to exploit the full power of the CSP model.

The design of the T9000 is a truly remarkable leap beyond the T8xx family. Its advanced features include a pipelined superscalar processor that delivers more than 150 MIPS and 20 MFLOPS, on-chip high-speed cache RAM, and 100-Mbps links.

However, its shortage of silicon and support hardware and software have been a problem. Although the T9000 was announced in 1991, it only recently started coming out of fab. Single 20-MHz units retail at \$450.

Inmos also offers a number of interesting support ICs that make it easy to design and interface flexible transputer networks. The IMS CO12 is an IC which converts the serial transputer protocol into parallel format and vice versa. The IMS COO4 link provides a crossbar switch between a maximum of 32 transputer links. It is cascable and enables reconfiguration



of the transputer network's topology under software control.

ICs with similar functions are available for the T9000's 100-Mbps links, including the IMS C104, a high-performance routing chip that interconnects T9000 transputers to form a full-blown packet-switching network. T9000s can be used in combination with any of the first-generation transputers by using an IMS C100 link converter IC, which translates 20-Mbps links to the 100-Mbps links of their second-generation counterparts.

TRANSPUTER ADD-IN BOARD FOR THE IBM PC

In 1987, Inmos introduced a T414 transputer add-in board for the IBM PC as the BOO4 model [4]. It was developed as an Occam engine hosted by the PC. In addition, it could accelerate computationally intensive tasks for the PC either alone or in conjunction with a transputer network. The BOO4 still supports transputer didactic and development environments for the PC.

Although the BOO4 performs well for unsophisticated applications, the introduction of the T8xx family made many T4xx-based products obsolete. Figure 1 offers an improved circuit, inspired by the simplicity of the BOO4 circuit, to illustrate the design of practical transputer systems. This simple circuit remains compatible with the original BOO4, but also accepts the powerful members of the T8xx family.

Based on this figure and some help from Inmos's transputer data book [5], it is relatively easy to develop more advanced systems. Figures 2-7 should enable you to build a fully functional transputer PC add-in card. Although it's a bare-bones approach, this design features compatibility with Inmos and third-party software; up to 4 MB of local DRAM; compatibility with T400, T414, T425, T800, and T805 transputers; DIP-switch selection of processor and link communication speeds; and DIP-switch memory configuration.

Ideally, transputer boards should be constructed using four-layer PCB

technology to ensure noise-free reliable operation. As with every board designed for high-speed operation, proper impedance matching and termination, extensive power-rail and ground-plane decoupling, as well as path-delay equalization are required [6].

THE T4xx/T8xx TRANSPUTER IC

As shown in Figures 2 and 4, multiple VCC pins minimize inductance within the IC, and all must be connected to a well-decoupled power rail. Similarly, all GND pins must be connected to the board's ground plane. C2, a 0.1- μ F ceramic capacitor, must be soldered directly between CapPlus and CapMinus to appropriately decouple the internal clock supplies. A 0.1- μ F ceramic decoupling capacitor between the VDD and ground planes of the PCB should be placed near the transputer's socket to aid in decoupling the power supply to this IC.

Clock design is simple since all first-generation transputers, regardless

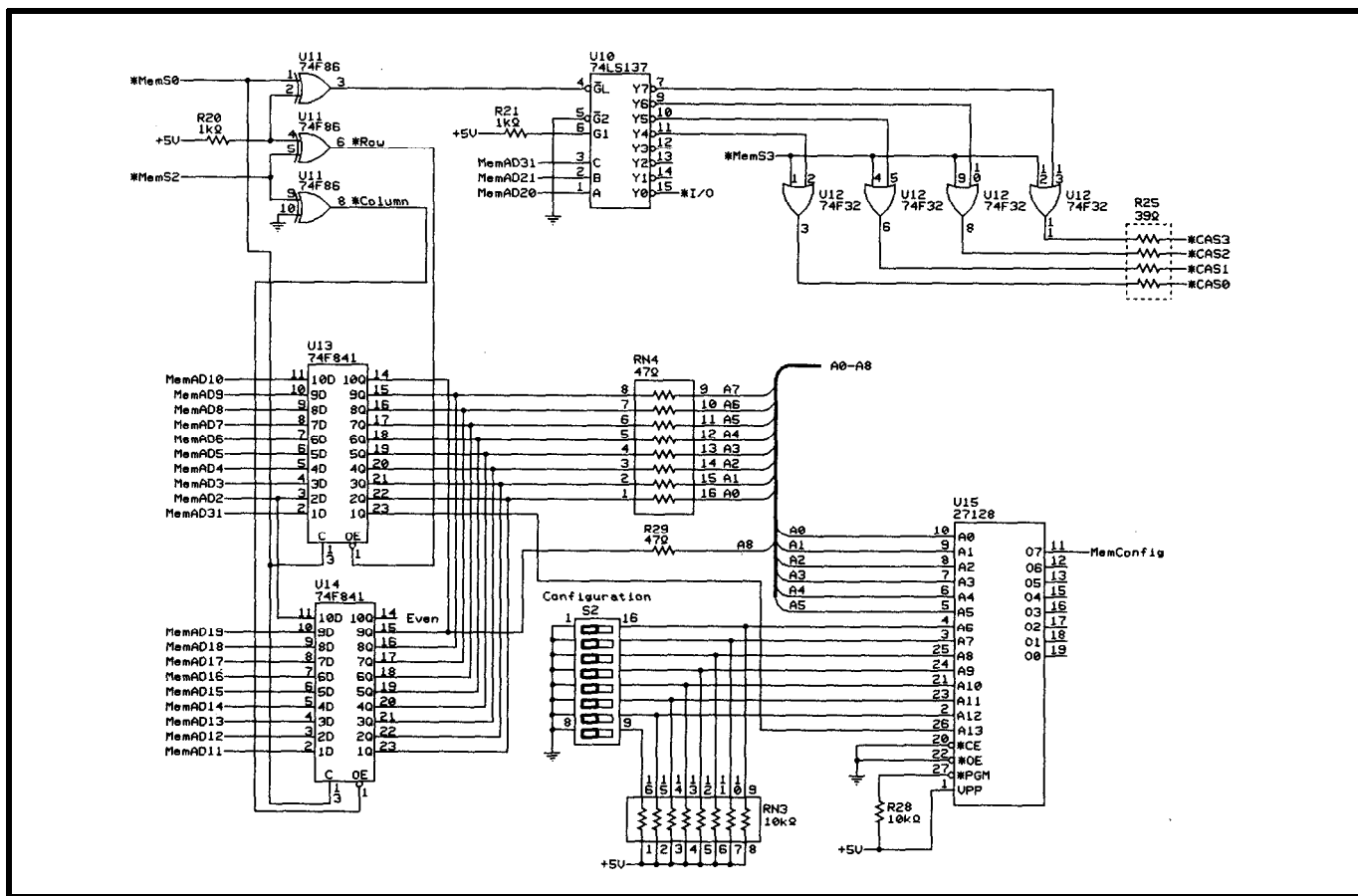


Figure 3—External memory address and control-signal decoding is accomplished through this circuit. The actual memory configuration (strobe use, timing, etc.) is automatically loaded from EPROM IU15 into the transputer on processor reset.

of device type, use a 5-MHz clock. The processor's high-frequency clocks are internally generated, which eases design and layout constraints so it's easier to construct a well-behaved board.

U2, a 5-MHz crystal oscillator, produces a stable 5-MHz clock signal. Processor speed, up to the maximum rated speed for a particular transputer, is selected through DIP switches S1-6, S1-7, and S1-8 as shown in Table 2.

MEMORY AND I/O SYSTEMS

The memory system implemented in the board is capable of supporting up to 4 MB of DRAM. This capacity is divided among four banks of 1 MB, each of which is implemented using four 256 KB x 9 DRAM SIP modules. For simplicity (unlike the original B004), parity-error checking has not been implemented.

The transputer has a built-in programmable memory interface which greatly simplifies the design of memory. This interface's programming determines the configuration of the external memory cycle required for a specific type of memory. In the board, a configuration corresponding to a given selection is entered through a DIP-switch bank and is translated into the appropriate memory configuration by U15, a 27128 EPROM (see Figure 3).

A detailed explanation regarding external transputer system memory configuration can be found in the transputer's data sheet. From this, you can configure almost every variation of the memory system you may want to implement in your card.

In addition, S706B ETA, the public-domain software tool used to design the EPROM code for this add-in card, helps you develop custom RAM configurations to be loaded in U15. S706-BETA and PROMLOAD, another freeware utility, help you develop bootable EPROM code for embedded applications.

In the add-in card, the EPROM is cycled by the decoded address bus A0-A5 and by the Scan/*Read line into two corresponding phases. During the Scan phase, all *MemS(i) strobes are held logic high. This state renders both 74LS841 latches transparent. When

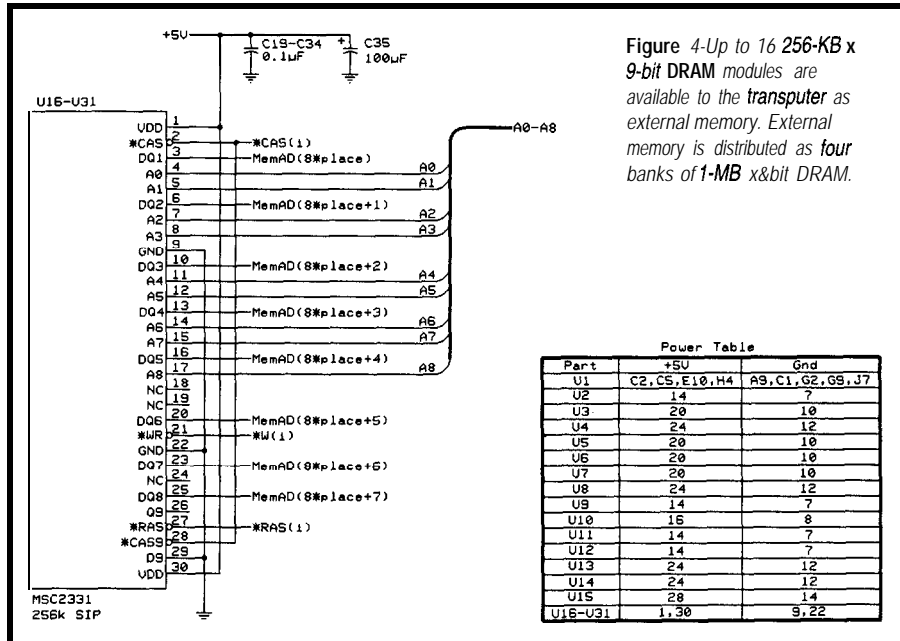


Figure 4-Up to 16 256-KB x 9-bit DRAM modules are available to the transputer as external memory. External memory is distributed as four banks of 1-MB x8bit DRAM.

this happens and MemS2 asserts the 'Row line, the logic high state of MemAD3 1 selects the EPROM's upper 8 KB. Zeros programmed in this area ensure that MemConfig is held low so the transputer loads an external configuration.

During the Read, MemAD3 1 remains low, selecting the EPROM's lower 8 KB. Different memory cycle configurations are stored in 128 segments, each of which is 64 bytes long. The desired segment number is user-selected through the configuration

Be a hero -- put *Team Paradigm* to work on your next embedded x86 design and reap the rewards you deserve. *Team Paradigm* has the ability to deliver all the embedded system pieces from Intel to AMD, C/C++ or assembly language, and all the Borland/Microsoft development tools. Plus Paradigm DEBUG for your favorite in-circuit emulator and real-time operating system.

So forget about making excuses and instead enlist *Team Paradigm* for your current or next x86 project. We deliver the finest embedded x86 development tools, backed by unlimited free technical support. No one is more serious about your success than Paradigm Systems. Call today and get the details before you waste any more of your precious time.

PARADIGM
Proven solutions for Embedded C/C++ Developers

1-800-537-5043

Paradigm Systems
3301 Country Club Road, Suite 2214, Endwell, NY 13760

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

Internet: 73047.3031@compuserve.com

©1995 Paradigm Systems, Inc. All rights reserved.

**TO BE
CONTINUED.**

DIP-switch bank S2. The correct memory-configuration sequence results through the cycling of lines AO-A5. Table 3 shows how to use the transputer strobe lines.

The transputer system's RAM is mapped into negative space. Therefore, RAM selection occurs during the logic-high cycles of

MemAD3 1. Selection of the appropriate bank is done through MemAD20 and MemAD21. When enabled by the address latch enable signal (*MemS0) and *CAS strobe (*MemS3), column-address selection signals (*CAS(i)) corresponding to each of the memory banks are decoded by U10, a 74LS137, which receives MemAD20,21,3 1 as inputs.

During refresh cycles, MemAD3 1 is logic low, thus disabling all *CAS(i) lines. RAM refresh-only is implemented on the board to take care of DRAM-refresh operations without disturbing I/O.

Only the subsystem port is mapped into the I/O area. This area resides at a positive address, which implies that *CAS(i) generation is disabled by the logic low of MemAD31 during I/O operations. I/O space is limited to the first megabyte of positive space because both MemAD20 and MemAd21 have to be low for I/O operations under the present implementation. The Even/* Odd selection line is the latched version of MemAD2, which appears during MemS2's falling edge. Figure 4 shows connection of the DRAM modules to the decoding logic and the transputer.

CONTROL SYSTEM

The transputer has a number of incoming and outgoing system-level control signals, which include processor reset (Reset), bootstrap control (BootFromROM), and facilities for error analysis (Error, ErrorIn, and Analyze).

The falling edge of Reset initializes the transputer, triggers the memory-configuration sequence, and starts the bootstrap routine. In this card,

DIP Switch			LinkSpeed(Mbps)	
S1-3 (LinkSpecial)	S1-1 (Link123Special)	S1-2 (Link0Special)	Link 0	Links 1,2,3
0	0	0	10	10
0	0	1	5	10
0	1	0	10	5
0	1	1	5	5
1	0	0	10	10
1	0	1	20	10
1	1	0	10	20
1	1	1	20	20

Table 4—Transputer link speeds are selected through switches S1-1, S1-2, and S1-3.

since the BootFromROM line is permanently grounded, the transputer waits for booting instructions to arrive from the host PC or from other transputers in a network. On powerup, U9e causes the logic in the 22V10 PAL (U8) to unconditionally assert the Reset line.

The error and analysis signals aid in developing and troubleshooting new designs. They become especially handy when debugging multitransputer networks. Asserting the Analyze line causes the processor to halt whenever it reaches specified breakpoint conditions, complete outstanding link transactions, and place special status values on the transputer's registers for debugging.

The Error pin conveys the state of the transputer's internal error flag ORed with the ErrorIn line. Internal errors are caused by conditions such as arithmetic overflow, divide by zero, array-bounds violation, or through direct software selection of the internal error flag. In a multitransputer network, the ErrorIn and Error pins of

a number of transputers can be daisy-chained to halt the network, making the status of each processor available for probing by a master transputer. In the transputer where the error originates, the error flag is not cleared by a processor Reset, so its location can be identified.

Executing the testerr instruction clears the flag and allows for normal system operation.

The add-in board's control signals are mapped to the PC's I/O space. By doing so, the PC can reset and analyze a network of transputers connected to the card's up or down subsystem ports. This feature prevents errors in the transputer network from flowing into the PC, so the PC can always be ready to reset and upload the add-in board.

Input control signals are Up-Reset (*UPR) and Up-Analyze (*UPA). These signals arrive from modules of higher hierarchy than that of the add-in transputer card while Subsystem Error (. SSE) signals arrive from modules of lower hierarchy. *UPR generates an unconditional Reset signal to the on-board transputer and link adapter. *UPA signals the Analyze input. SSE can be read by the onboard transputer at Occam address #200000000 (absolute address zero) as a logic 1 in the LSB of the word.

Output control signals generated by the board are Subsystem Reset (*SSR) and Subsystem Analyze (*SSA).

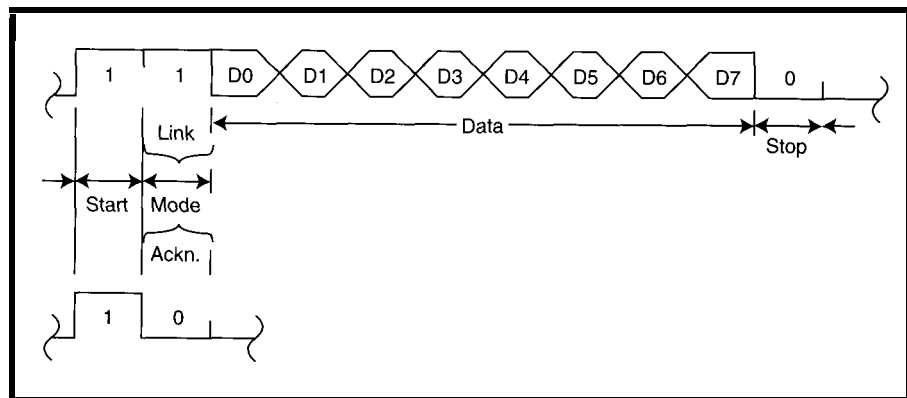


Figure 5—In the protocol for data transmission over high-speed serial links, each byte transmitted through the LinkOut line of a channel is preceded by a start bit and followed by a 1 bit and a stop bit. After each transmission of a byte, the sending device waits for an acknowledge signal. This acknowledge signal, received through the LinkIn line of the transmitting device, is formed by a start bit followed by a 0 bit.

These signals control transputers placed in lower hierarchies than those of the PC add-in board. Line *SSR can be asserted by the onboard transputer by writing a 1 to the LSB of absolute address zero. Similarly, *SSA can be asserted by writing a 1 to the LSB of absolute address 4 (Occam address # 20000001).

When the board is in the enhanced mode, the PC and onboard transputer can control an attached network of transputers. To do so, ● SSR and ● SSA are asserted through control signals PCUpReset and PCUpAnalyze generated by the PC. This approach prevents an error-generating network from interfering with the resetting and booting of the PC's add-in card. The status of the onboard transputer Error pin is placed in the up port as the Up-Error signal ● UPE.

If you want, the hierarchy arbitration can be changed by deasserting the *System line so that an error received from an attached transputer network through Down-Error signal ● DNE is forwarded to the PC as a *UPE.*UPR and *UPA signals received from the up port are buffered and forwarded to the down port as a Down Reset (*DNR) and Down Analyze (*DNA).

The system and subsystem control logic is implemented in US, a 22V10 PAL. The Subsystem, PC, Up and Down Reset, Analyze, and Error signals are brought out of the add-in card through edge connector J7 for easy connection to an external transputer network.

For normal operation, the *System line must be jumpered to ground (J7-2 to J7-1), and the PC control lines must be connected to the up port: ● PCR to ● UPR (J7-5 to J7-11), PCA to *UPA (J7-7 to J7-13), and ● PCE to ● UPE (J7-3 to J7-9).

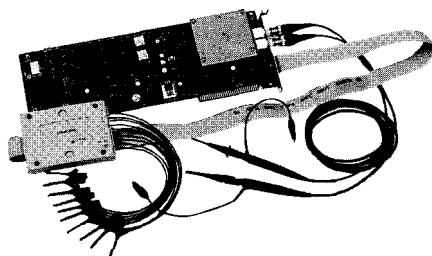
5-, 10-, AND 20-MBPS SERIAL LINKS

Communications through a transputer link is carried out through a simple protocol which supports the synchronized communication requirements of Occam. This protocol transmits an arbitrary sequence of bytes, which interconnects transputers with different word lengths.

PC-Based Instruments

200 MSa/s DIGITAL OSCILLOSCOPE

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



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

DSO Channels

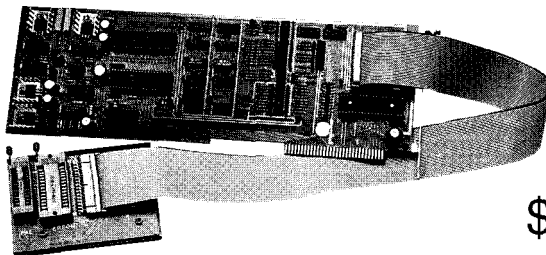
2 Ch. up to 100 MSa/s
or
1 Ch. at 200 MSa/s
4K or 64K Samples/Ch
Cross Trigger with LA
125 MHz Bandwidth

Logic Analyzer Channels

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

Universal Device Programmer

PAL
GAL
EPROM
EEPROM
FLASH
MICRO
PIC
etc..

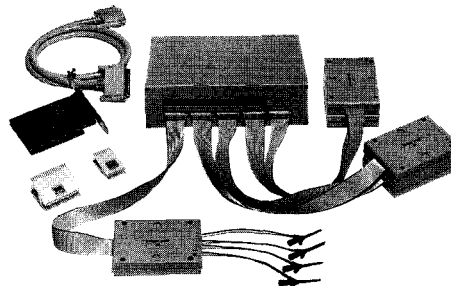


\$475

Free software updates on BBS
Powerful menu driven software

400 MHz Logic Analyzer

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



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

Call (201) 808-8990



Link Instruments

369 Passaic Ave, Suite 100, Fairfield, NJ 07004 fax: 808-8786

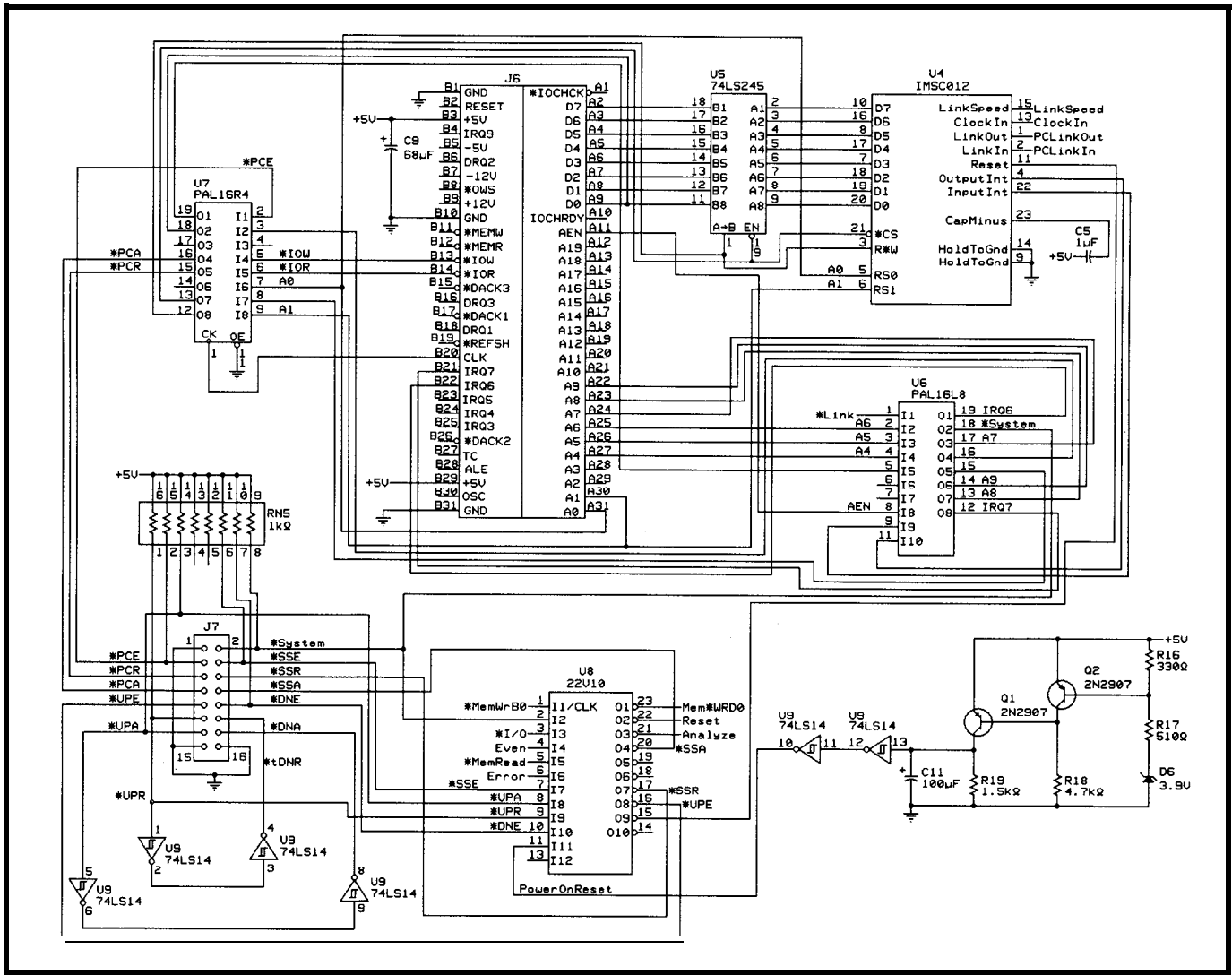


Figure 6—The interface between the transputer's link U and the PC's data bus is performed through U4 and bus transceiver U5. Glue logic, as well as error and analysis logic, is implemented in PALs U6, U7, and U8.

As shown in Figure 5, the eight bits of each byte transmitted through a channel's LinkOut line are preceded by a start bit and followed by a 1 bit. The data bits are then followed by a stop bit. After each byte's transmission, the sending device waits for an acknowledge signal from the receiving device before proceeding. This acknowledge signal, received through LinkIn on the transmitting device, is formed by a start bit followed by a 0 bit.

Transmitted bytes and acknowledgments are multiplexed on the various signal lines. As long as a process is waiting for the sent data and there is room in the data buffer, acknowledgment is sent immediately on data latching. The transmission is carried out without delays between successive data bytes.

Transputer links are designed for electrically quiet environments. Keep the direct interconnection between transputers to about 1'. The link lines are fully TTL-compatible, so industry-standard line drivers and receivers can extend the distance of these links.

Interface design is simple because all first-generation transputers use a 5-MHz oscillator with PLL for reference, and no phase reference is required. Separate transputers interconnected through the same link may thus operate from independent clocks, each of which may be running at a different frequency.

Link speeds can be selected between 5-, 10-, and 20-Mbps by programming switches S1-14, S1-15, and S1-16. Table 4 shows how link zero can be set independently from links 1,

2, and 3, which must be programmed to the same speed.

PC INTERFACE

The host PC communicates with the transputer through one of its serial links. As far as Occam is concerned, this maps the PC as a process connected through a channel implemented on that dedicated link.

As shown in Figure 6, translation between Inmos's serial-link protocol and the PC parallel data bus is performed through U4, an IMSC012 link adapter IC. Parallel data transfers between the adapter and the PC occur through U5, a 74LS245 bus transceiver, under the control of the logic programmed into PALs U6 and U7.

U6 and U7 decode the address bus, data direction controls, and address/

data bus demultiplexing, translating them into an appropriate timing sequence for the link adapter. The PALs' logic also arbitrates the master, slave, and subsystem hierarchies for the process, so Occam recognizes the PC.

Connection of the PC interface to the transputer's link 0 is done through jumper cables between the PC link and link 0 edge connectors (J1 and J5).

Asserting *Link low enables communications with the PC bus. The link speed for the IMS CO 12 must match that of link 0, which is set through switch S1-4. When this switch forces line LinkSpeed low, the link adapter IC operates at 10 Mbps. When high, it operates at 20 Mbps.

The logic in U6 makes software written for Inmos's BOO4 compatible with this board by placing it in the same location in the PC's I/O address space (150h-163h).

To experiment with true parallelism, additional processors can be added to the transputer system. Since additional transputers may be booted through the serial links, only one transputer in the network needs direct connection to the PC bus.

In additional cards, • Link must be high (achieved by removing the jumper between pins 2 and 4 of connector J5). Glue logic which interfaces the board to the PC may be omitted (or the ICs removed from their sockets). All links of the transputers are available for network interconnection by removing the connections between the PC link and link 0 connectors J5 and J1.

Extra cards use the PC only as a power source and do not need to share the same motherboard or power source. They may reside in an external chassis capable of supplying appropriate power and cooling or may occupy a slot in a PC.

TESTING THE BOARD

Testing your newly assembled transputer hardware is simple through PC-Check, a freeware utility package which determines the types, versions, functionality, and topology of all transputers in a network. As shown in Figure 7, PC-Check also identifies the speed of the link connected to the PC and performs a basic test of each

```

a)
A:\>check| mtest

Using 150 check 3.0
# Part rate Mb Bt [.link0 Link1 link2 Link31 RAM,cycle
0 T425b-20 0.17 0 HOST ... .. 4K,1+1022,4

b)
A:\>check| ftest

Using 150 check 3.0
# Part rate Mb Bt [.link0 Link1 link2 Link31 (Ftest)
0 T425b-20 0.17 0 HOST ... .. (OK)

```

Figure 7—(a) An example of a PC-Check test of the transputer PC add-in card shows that CHECK.EXE determines the types, versions, and topology of all transputers in a network, and reports on all available RAM. (b) FTEST.EXE uses the piped output from CHECK to establish the functionality of each transputer in the

transputer's internal and external memory.

In Figure 7a, CHECK.EXE identified a T425b as connected to the host through link 0, running at 20 MHz, and the sole transputer in the network. The pipe into MEMFEST.EXE reports 4 KB of 1-cycle memory (internal memory) and 1022 KB of 4-cycle RAM. Piping the output of CHECK.EXE into FTEST.EXE (Figure 7b) executes a full functional test of each network transputer. Each device which passes the battery of tests receives an "OK" message.

When CHECK.EXE runs, it leaves a vestigial routing trail at each node which sets a default communications path from the host to any transputer in the network. Through this path, LOAD.EXE, another program in PC-Check, can load programs on any transputer node. The package also includes a hex monitor and a simple method for configuring the topology of a network connected through one or more IMS COO4 software-controlled crossbar switch ICs.

To run your own programs, use IServer as an interface between the PC host and the transputer network. IServer, a public domain package, allows the PC to reset, boot, and analyze a transputer network. It also gives the transputer network access to various PC resources such as file I/O, keyboard input, and CRT output.

OCCAM PROGRAMMING

Transputers can be programmed in most popular programming languages, but its special properties are best ex-

ploited by Occam since the transputer directly supports the Occam model of concurrency and communication.

Occam syntax uses indentation to indicate program structure. Specialized folding editors are used to write a program. A folding editor represents a large text block in a single named fold line marked by three dots. Folds can be nested to any level and created before their contents are written, allowing the structure of a process to arise as the design's skeleton.

Occam describes a system as a collection of distributed concurrent processes that communicate with each other through channels. Occam consists of three primitive processes which are combined to create larger processes:

- Output-c !e outputs expression e into channel c
- Input-c? v inputs channel c into variable v
- Assignment-v :=e assigns expression e to variable v.

These primitives are joined by three types of constructors: sequential (SEQ!), parallel (PAR), and alternate (ALT). Statements or subprocesses contained within the context of SEQ execute in sequence, while those under PAR execute concurrently. In contrast, ALT has the processor execute whatever component process is ready first. Programs can be built in the conventional way by employing variables, assignments, mathematical and logical expressions, and with standard constructs such as IF, WHILE, and FOR.

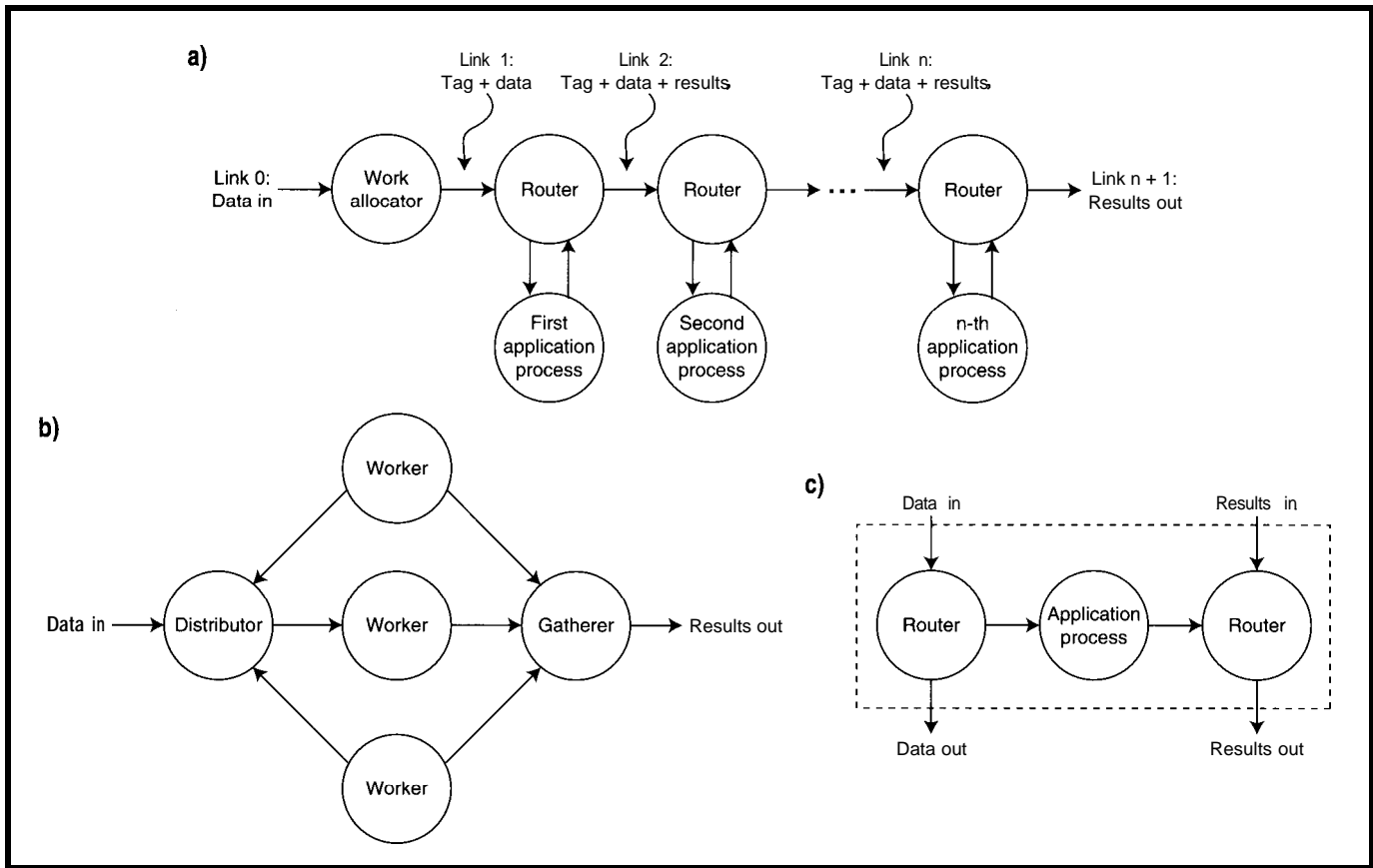


Figure 8—The most simple topology supported by the PAR replicator syntax is a pipeline (a). A spaceline system is slightly more complex, using a distributor process to hand out data to concurrent worker processes, and a gatherer process to collect and combine results (b). Input and output of these structures can be at one end by introducing an additional router to achieve bidirectional communication (c).

Each Occam channel provides a communication element between two concurrent processes. Information is sent via a channel in one direction only without message buffering. A channel is perceived as a read-only element to a receiving process and a write-only element to a transmitting process. Communication is synchronized and takes place when both the inputting and outputting processes are ready.

The *ALT* construct elegantly selects between executing parallel processes. It waits for input from a group of input channels and then executes the process attached to whichever one becomes active first. If more than one input arrives simultaneously, *ALT* chooses to act based on assigned process priority and the implementation of the program. Its choice ensures that no channel becomes permanently locked out.

PAR's power lies in its simple way of replicating arrays of similar processes. As shown in Figure 8, the most

simple topology supported by the *PAR* replicator syntax is a pipeline. Each stage performs data processing and passes data and/or results on behalf of other processes.

A spaceline system is slightly more complex. A distributor process hands out data to concurrent worker processes, and a gatherer collects and combines results. In addition, since channels are available in both directions, a router process can be introduced so input and output can be at one end of a pipeline.

The interconnection between processes is limited only by the number of available links. While this doesn't pose a problem for processes executing concurrently on a single transputer, it does when the transputer's physical links are used as Occam channels. Then, simultaneous connectivity between transputers is limited to the four high-speed serial links.

This arrangement still allows for many useful topologies. Binary trees

and H-trees provide almost unlimited extensibility. Two-dimensional grid arrays are also extensible and present improved communication and reduced path lengths between elements. The next level of architecture is the *n*-dimensional cube or binary *n*-cube. With $n = 2$ and 3, the topology is that of a square and a cube, respectively. A 4-cube or basic hypercube has 2^n transputers and achieves an almost ideal configuration for connectivity and channel-path length.

A hypercube can mimic other topologies. Processing can occur as a number of independent trees, 2-cubes, or 3-cubes just by preventing communications between selected nodes. Since it is topologically identical to a torus (an improved grid array), the hypercube is flexible, making it a favorite architecture for multitransputer networks.

By using multiple transputers within a single node to extend the number of links available to each node, higher-order hypercubes are

possible. A two-transputer supernode has six available links and can be used as a 6-cube. A four-transputer supernode forms an g-cube, and so on.

SECOND-GENERATION TRANSPUTERS

Despite a wide variety of topologies, implementation of universally parallel algorithms isn't possible using first-generation transputer hardware and software because they comply only partially with the CSP parallelism model. The discrepancy exists because message exchange between parallel processes in different transputers is limited by the transputer's four communications links.

The T9000 second-generation transputer provides a remedy. Besides hardware enhancements that increase operation speed and support advanced operating systems, the T9000 provides greater connectivity. Each link is connected to multiplexing hardware.

Communications among processes in separate transputers takes place along as many channels as required. Shared physical links are transparent to software, so true CSP parallel programs can be used.

This capability is achieved through a separate communications processor. It transmits messages that multiplex a large number of virtual links along each physical link. Virtual links support one Occam channel in each direction. Messages are transmitted as a sequence of 32-byte data packets. A header which precedes the packet routes the packet through the network and identifies the destination virtual link on the remote transputer.

The new packet communications hardware simplifies programming because software systems then don't need process allocation. Different allocations can be used for different networks, and they can change dynamically to optimize performance.

Moreover, the compiler can make the allocation, removing configuration details from the program. This is a vital step toward development of a general-purpose parallel computer.

To form a full-blown packet-switching network, one or more IMS C 104 high-performance routing chips interconnect T9000 transputers. Each 'C104 provides 32 bidirectional 100-Mbps links. The header of each packet arriving on a link input determines the link a packet should output to as soon as the physical link is free.

The basis for optimal packet routing is an algorithm called *interval labeling*. An interval is a continuous set of header values and is associated with an output channel. The header of an incoming packet is limited to within a single range, and the packet is directed toward the appropriate interval. Most common network topologies have stable and efficient labeling schemes.

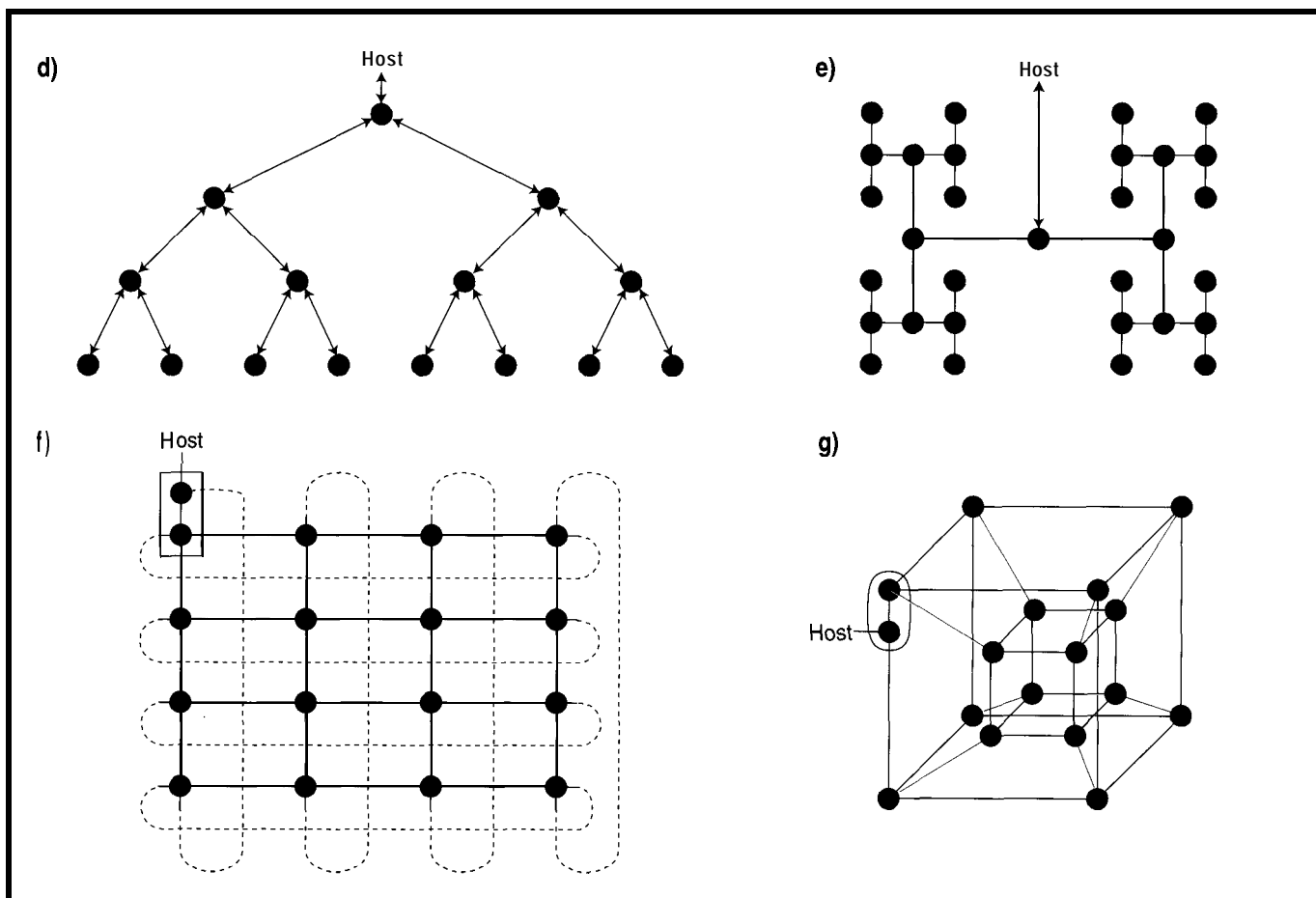


Figure 8 (continued)-Communications between processes running in separate transputers (shown as dots) are limited by each transputer's four high-speed links. Useful topologies such as a binary tree (d), an H-free (e), and a processor mesh (f) can be implemented with ease. Connecting the free links of the mesh (broken lines) results in the very powerful topology known as the basic hypercube (g).

In the T9000, parallelism starts to be exploited within the processor by using a pipelined superscalar architecture. The processor core executes up to eight instructions on each 20-ns clock cycle. Inmos even has dedicated on-chip hardware which controls the flow of multiple instructions through the pipeline, making it possible to run existing T8xx code at blazing speeds.

Another improvement is the move toward a cached architecture supported by 16 KB of on-chip cache RAM. This much RAM is often enough to support lightning-fast embedded applications without external RAM. The T9000 still supports external memory through a programmable memory interface, but it also has a 64-bit data bus with high data-transfer rates for cache-line refill. In addition, the new interface supports four independent banks of memory, each of which can be individually programmed.

The T9000 also provides trap handling and protected processes. The former enables error processing through software before control is returned to the process in which the error occurred. The latter supports secure programming and debugging in embedded systems, which protects processes and the operating system from other errant programs executing concurrently.

Second-generation transputers can be interfaced to previous-generation transputers by using the IMS C100 link converter IC mentioned earlier. In an advanced real-time embedded application, for example, T805 transputers can acquire and preprocess sensor-array signals, while a network of T9000s simultaneously executes computationally intensive processing, analysis, sensor fusion, and control portions of the program. Output post-processing and actuator control could use the less powerful T2xx or T4xx transputers as embedded controllers.

YOU-PART OF THE PC REVOLUTION?

Throughput of PCs and workstations has increased dramatically in recent years, and uniprocessors are quickly approaching the limits of cir-

cuit designs using current fabrication technologies. It is safe to assume that keeping up with increasing demands for computational speed will result in increased parallel processing.

To a certain extent, it is already happening. Take a look under the hood of your PC. Dedicated processors are on almost every add-on card, providing intelligent graphics, I/O, and cache controllers. New multimedia cards integrate DSPs, signal and image compression and decompression, and stand-alone media controllers. Some companies take this trend one step further by offering dual-Pentium motherboards, and multi-i860 add-in boards for the PC.

The shift toward truly parallel machines will be gradual—the industry-standard PC won't be massively parallel for a number of years. Yet, within the next two years, there will be more and more ads for small-scale superscalar and parallel PCs in which a small number of Pentium (or P6, etc.) processors run concurrently within a bus-based, shared-memory system.

The migration to distributed-memory systems may happen only when the scalability limitations of shared-memory systems are reached. Whose chips will dominate then? With the aid of today's low-cost parallel processors, you may well be one of the pioneers to exploit personal computing's next revolution. ☛

David Prutchi has a Ph.D. in Biomedical Engineering from Tel-Aviv University. He is an engineering specialist at Intermedics, and his main R&D interest is biomedical signal processing in implantable devices. He may be reached at davidp@mails.imed.com.

REFERENCES

- [1] G. Amdahl, "Limits of Expectation," *International Journal of Supercomputer Applications*, 2(1), 88-94, 1988.
- [2] B. Christianson, "Amdahl's Law and the End of System Design," *Performance Evaluation Review*, 19(2), 3032, 1991.

- [3] L. Kleinrock and J.H. Huang, "On Parallel Processing Systems: Amdahl's Law Generalized and Some Results on Optimal Design," *IEEE Transactions on Software Engineering*, 18(5), 434-447, 1992.
- [4] S. Ghee, *IMS B004 IBM PC Add-in Board*, Inmos Technical Note 11 (72TCH01100), 1987.
- [5] Inmos, *The Transputer Data-book*, 3rd. Ed., 1992.
- [6] D. Prutchi, "Designing Printed Circuits for High-Speed Logic," *INK*, 42, 38-43, 1994.

SOURCES

INMOS
1000 East Bell Rd.
Phoenix, AZ 85022
(602) 867-6100

Transtech Parallel Systems
20 Thornwood Drive
Ithaca, NY 14850-1263
(607) 257-6502
Fax: (607) 257-3980

Computer System Architects
{CSA}
100 Library Plaza
15 North 100 East
Provo, Utah 84606-3 100
(801) 374-2300
Fax: (801) 374-2306

Transtech and CSA offer transputer hardware and distribute many software packages. These include n-parallel Prolog interpreter by Paralagic; Parallel C, C++, Fortran, Pascal for the Transputer by 3L; a C transputer toolset from Logical Systems; and the Professional OCCAM Toolset, ANSI C Toolset, ANSI Fortran 77 Toolset, and Inquest Debugger from INMOS.

Parallel-processing software is also freely available through the Internet. Contact ftp://unix.hensa.ac.uk and ftp://ftp.inmos.co.uk.

IRS

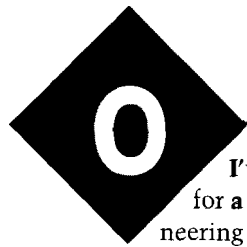
404 Very Useful
405 Moderately Useful
406 Not Useful

FEATURE ARTICLE

Michael Smith

Developing a Virtual Hardware Device

Mike finds out what it takes to simulate virtual devices on the buses of MC68332 CISC and AM29200 RISC microcontroller evaluation boards. His concern: how the simulators deal with exception handling of bus errors.



Over the last year, I've taken courses for a Software Engineering Management certificate, which has refreshed my knowledge of how the "real" world works. I now know that an industrial software project's time is divvied out like this:

- 30%-requirements analysis
- 25 %-coding
- 15 %-review process
- 15%—unit testing
- 215%—waiting for hardware

The cost of waiting has created a need for device simulators. A virtual device shares the characteristics of a real device but is built entirely in software. You can move and read values or cause actions with the device registers just as if the device was attached to the processor bus!

With such software devices, you can integrate and test before the hardware arrives, making it easier to schedule development time and people.

Suppose you design a pseudo-device to speed a software project's development. What do you need to simulate? If you use a virtual device with a stand-alone evaluation processor board, what resources on that board support your simulation?

As a Motorola course book points out, "The quality of a simulation is not measured by how accurately it simulates a device, but instead by how effectively it trains the programmer to do the right thing when dealing with an actual device." [1] Keeping this in mind, here's one way to simulate the operation of virtual devices.

Let's consider the differing requirements for simulating virtual devices placed on the buses of the Motorola MC68332 CISC and the Advanced Micro Devices Am29200 RISC microcontroller evaluation boards. The simulations use the exception handling and I/O capabilities of the evaluation boards' operating systems to implement the virtual device. We'll look into the different ways interrupts are handled with RISC and CISC processor architectures.

To minimize the simulation's programming requirements, I'll use a virtual coffee pot. The device's code is placed on the Circuit Cellar BBS.

SIMULATION REQUIREMENTS

Listing 1 shows how a real coffee pot, physically attached to a controller on an evaluation-board bus, might operate. A virtual coffee pot should run the same code using the same device-register accesses.

From the pseudocode, it's clear that a virtual coffee pot must be ca-

Listing 1--The pseudocode for controlling the operation of a real coffee-pot device is the same no matter what the target processor.

```
InstallCoffeePotOnBus();
control register = RESET;
AddWater();
while (temperature < HOT-ENOUGH) Heat(FAST); /* Heat water */
while (timer < READY) Heat(SLOW);           /* Perk coffee */
CoffeeReady();
```

pable of handling a number of operations. It must:

- recognize that the pseudodevice exists (InstallCoffeePotOnBus())
- set the rate for heating the water. Heat () corresponds to writing a control value to a device register.
- access and change the timer and temperature values
- handle other register accesses that may produce multiple simulation operations

Now let's examine how to use an evaluation board's resources to implement a coffee-pot simulation that supports these operations.

DEVICE DEFINITION

In defining a virtual device's functionality, first specify the registers that control and manipulate it. Table 1 gives the register description for the coffee-pot device.

Offset	Mnemonic	Description
0x00	C_CONTROL	Initialization of device
0x10	C_HEAT	Power-level control on heater
0x20	C-TEMPERATURE	Current water temperature
0x30	C-TIMER	Time water has been heated
0x40	C-WATER	Opens and closes valve to add water

Table 1--The registers defined for a COFFEEPOT device controller include all the necessary information about the state of the device.

In simulating the actual device's operation, COFFEEPOT must:

- store a value in C_CONTROL that resets C-HEAT control, C-TEMPERATURE sensor, and C-TIMER
- store a value in the C_HEAT register that modifies C-TEMPERATURE's rate of change
- simulate heat transfer (i.e., pot cooling) when a heating element is not high enough
- access C-TEMPERATURE and display a status message
- measure the time since the coffee pot was plugged in.

Using these register definitions and requirement statements, let's program the RISC and CISC evaluation boards to operate on the coffee-pot device. Listings 2 and 3 show the necessary, but inefficient, assembly language code for the Am29200 RISC and MC68332 CISC processors.

To show the equivalence of the base device operation for these very different processors, Listing 4 defines Am29200 LOADindexed and STOREindexed macros equivalent to the MC68332 indexed instructions.

The similarity of the code at this level contrasts with the different coding practices needed for the underlying simulation. It becomes apparent later why the full range of MC68332 CISC MOVE instructions can't be used to access the virtual device. The same code should be required to operate COFFEEPOT, whether the device is real or virtual.

But, how can we change, check, or read the heating, timing, or temperature registers on a virtual device when none of them exist!

Listing 2--The virtual COFFEEPOT device can be implemented using Am29200 assembly language.

```
.set value,1r3 ; register definitions
.set base_address,1r2
.set rtnaddress,1r0
.set boolean,gr96

main: Code here to set register and memory stacks
CALL rtnaddress,InstallPot ; InstallPot();
NOP

LADDR base_address,COFFEEPOT
CONST value,RESET ; C-CONTROL = RESET;
STOREindexed base_address,C_CONTROL,value
CONST value,FILLPOT ; C-WATER = FILLPOT;
STOREindexed base_address,C_WATER, value

CHECKBOILING:
LOADindexed value,base_address,C_TEMP
CPGE boolean,value,HOTENOUGH ; while (C_TEMP < HOTENOUGH)
JMPT boolean,BOILED
CONST value,FAST ; Heat(FAST);
STOREindexed base_address,C_HEAT,value
JMP CHECKBOILING
NOP ; (Could fill this delay slot)

BOILED:
LOADindexed value,base_address,C_TIMER
CPGE boolean,value,READY ; while (C_TIMER < READY)
JMPT boolean,COFFEEREADY
CONST value,SLOW ; Heat(SLOW);
STOREindexed base_address,C_HEAT,value
JMP BOILING
NOP ; (Could fill this delay slot)

COFFEEREADY:
```

CONNECT THE VIRTUAL DEVICE

It's easy to simulate the basic operation of reading and writing the virtual device registers—simply dedicate some RAM space for those registers. However, the simulation is rather unrealistic because the temperature- and timer-register contents never change. And, writing to one device register doesn't affect other registers.

For useful values, a background process must perform operations such as changing the pseudotimer and temperature values. Adding a true multitasking environment to produce a real-time simulation is not a five-minute job, especially on top of the board's existing operating system. Since we don't need this much realism for a coffee pot, we can use a standard programmer's trick: we can fake it.

Suppose a series of pseudodevices is sitting at locations not occupied by anything valid on the evaluation board's bus. For the SA29200 board, the addresses can be in the DRAM memory range of 0x50000000 to

0x50FFFFFF. For the MC68332 board, they can be in the range 0x20000 to 0x2FFFFFF. Accessing these memory locations immediately introduces a bus error, causes an exception, and crashes the program. The processor is taking notice of the virtual devices.

If the vector for the bus error's exception-handling routine is adjusted, it can be steered to our own exception-handling routine. In this handler, I can implement any operation I want when the virtual device registers are accessed. Listing 5 gives the `InstallPot()` routines for the Am29200 and MC68322 microcontrollers.

I'm sticking to virtual device operations only during device-register accesses. Combining the bus error exception-handler form of register modification with one performed at specific intervals under the on-chip timer's control offers better event simulation when accurate timing is important.

THE AM29200'S VIRTUAL DEVICE

Let's assume a virtual coffee pot is attached to the bus of the Am29200 evaluation board. When it reads or writes to one of the virtual device registers, a bus-error trap occurs. This catapults us to the exception-handling routine `PSEUDODEVICE`.

As in any processor, the bus-error trap causes a context switch on the Am29200. Thus, the state of all previous activity must be preserved until after the exception has been handled. No user or system resources can be modified during the exception until after they've been saved. The exception handling is then transparent to the processes that use it.

These requirements produce different results on the register-rich, highly pipelined RISC Am29200 than they do on the CISC MC68332. I'll explore some of the basic differences between exception handling on these processor architectures.

Table 2 shows the various Am-29200 registers dedicated for exception handling with their mnemonics. These dedicated registers can speed exception-handling on a RISC processor, eliminating much of the slower memory-stack storage needed to handle

Listing 3—The virtual `COFFEEPOT` device can also be implemented in MC68332 assembly code

```

value equ D0 ; register definitions
base-address equ A0

main: ; Code here to set stack
    JSR InstallPot ; InstallPot();

    MOVE.L #COFFEEPOT,base_address
    MOVE.L #RESET,value
    MOVE.L value,C_CONTROL(base_address); C-CONTROL = RESET;
    MOVE.L FILLPOT,value
    MOVE.L value,C_WATER(base_address) ; C-WATER = FILLPOT;
CHECKBOILING:
    MOVE.L C_TEMP(base_address),value
    CMPI #HOTENOUGH,value ; while (C_TEMP < HOTENOUGH)
    BGE BOILED
    MOVE.L #FAST,value ; Heat(FAST);
    MOVE.L value,C_HEAT(base_address)
    JMP CHECKBOILING
BOILED:
    MOVE.L C_TIMER(base_address),value
    CMPI #READY,value ; while (C_TIMER < READY)
    BEQ COFFEEREADY
    MOVE.L #SLOW,value ; Heat(SLOW);
    MOVE.L value,C_HEAT(base_address)
    JMP BOILED
COFFEEREADY:

```

Listing 4—Macros for the Am29200 RISC processor that provide equivalents for the MC68332 CISC processor indexed-memory operations ease parallel code development.

```

.macro LOADindexed,dest,address,offset
    ADD address,address,offset ; (Index address)
    LOAD 0,0,dest,address ; (Get Value)
    SUB address,address,offset ; (unIndex address)
.endm

.macro STOREindexed,address,offset,src
    ADD address,address,offset ; (Index address)
    STORE 0,0,dest,address ; (Store Value)
    SUB address,address,offset ; (unIndex address)
.endm

```

Mnemonic	Register	Register Function
CHA	SR4	Frozen memory CHannel Address register
CHD	SR5	Frozen memory CHannel Data register
CHC	SR8	Frozen memory CHannel Control register
PC0	SR10	Frozen old PC value
PC1	SR11	Frozen old PC value
PC2	SR12	Frozen old PC value
ALU	SR132	Frozen ALU status register
useindirect	gr0	Marker to indicate using indirect register pointers
rsp	gr1	Current Register Stack Pointer
traptemp/funcaddress	gr121	Dedicated General Register
traptemp2/devaddress	gr122	Dedicated General Register
msp	gr127	Current Memory Stack Pointer
OPS	SR1	Current Old Processor Status
CPS	SR2	Current Processor Status
IPsrc	SR129	Current Indirect Pointer A register

Table 2—Use of registers in exception handling on the Am29200 eliminate slower memory-stack storage necessary on CISC processors.

Listing 5—It takes just a handful of instructions to install the bus error handlers on the Am29200 and MC68332 microcontrollers.

```
; Am29200 microcontroller bus error handler set-up
.equ COFFEEPOT,0x50000000 ; virtual device
.equ P_COFFEEPOT,0x40007000; real RAM locations
InstallPot:
LADDR gr96,PSEUDODEVICE
LADDR gr97,VECTORLOCATION ; Bus error vector
JMPI rtnaddress ; (In delay slot, as we return)
STORE 0,0,gr96,gr97 ; (store addr in vector location)

; MC68332 microcontroller bus error handler setup
COFFEEPOT equ $20000 ; virtual device
P-COFFEEPOT equ $A000 ; real RAM locations

InstallPot: ; (Store addr in vector location)
MOVE.L #PSEUDODEVICE,VECTORLOCATION ; Bus error vector
RTS
```

exceptions on CISC processors. When the exception handler is entered, the disable-interrupt bits in the current processor status register (CPS) are activated and the original status is saved in the old processor status register (OPS). Of the 192 general and local registers on Am29200, registers gr121 and gr122 are typically dedicated as temporary registers during exceptions. They needn't be saved unless multiple levels of exceptions are handled.

A unique feature of exception handling on the Am29200 is that the processor is said to be "frozen" when it enters the exception handler. This state is signified by the Freeze bit in the CPS register.

To get the most from a pipelined processor, data and memory accesses must be handled in parallel with instruction execution. For best performance, data access may end up far removed from its initiating instruc-

Listing 6—A bus error exception handling routine can be used to implement LOAD and STORE operations on Am29200 virtual device registers.

```
.equ NOTNEEDED, 0x2
.set traptemp, gr121 ; register definitions
.set func_address, gr121
.set dev_address, gr122

PSEUDODEVICE:
MFSR dev_address,CHA ; (Grab address fault info)
LADDR traptemp, (P-COFFEEPOT COFFEEPOT)
ADD dev_address,dev_address,traptemp; (Conv to physical space)

MFSR traptemp,CHC ; (Grab control info)
MFSRIM CHC,NOTNEEDED ; (Tell processor 'JUNK I/O OPERATION')
MFSR IPsrc,traptemp ; (Remember register indirection)

; The following operation shifts the LOAD/STORE bit from the
; Channel Control register into a BOOLEAN TRUE/FALSE
; position to control a conditional jump
SLL traptemp,traptemp,16
JMPF traptemp,HANDLESTORE ; if (WASLOAD)
NOP ; (delay slot)

HANDLELOAD:
LOAD 0,0,useindirect,dev_address ; reg = *dev_address;
IRET

HANDLESTORE:
STORE 0,0,useindirect,dev_address ; else *dev_address = reg;
IRET
```

BIG THINGS COME



DOMINO

Microcontroller




DOMINO
Microcontroller
MODEL _____ SN _____

Starting at
\$79

Micomint's
Domino-52
microcontroller is a
"supercomputer" in less than
0.75 cubic inches. We've packed
the most essential elements into
one tiny package. Domino is a
plug-and-go module, just attach
+5V and a terminal or network.
A simple keyed sequence saves
an autostarting program in
nonvolatile memory.

SPECIAL FEATURES

- 80C52 with ROM-resident, full floating-point BASIC
- 32K bytes SRAM and 32K bytes EEPROM
- Two PWM outputs, I²C bus
- Serial I/O: (up to 19,200 bps) RS-422, RS-485 & RS-232A
- Two interrupts and three timers
- Parallel I/O: 12 bits, 3 shared with ADC and I²C
- Power: +5V @ 15 mA;
Size: 1.75" x 1.062" x 0.4" potted
- A/D converter: 2 channels, 12 bits, 10k samples/sec.
- Connections: via 2 x 10, 0.1" dual-row header
- -20°C to 75°C operating temperature
- Industrial temperature available



4 Park Street • Vernon, CT 06066

Call 1-800-635-3355

(860) 871-6170 • Fax (860) 872-2204

tion. In fact, the instruction code may be flushed from the processor before data access is complete. This flushing action can cause complications if an interrupt occurs before memory access finishes.

On the AMD 29k processors, memory access is controlled by a dedicated data channel and several channel-support registers. If an exception occurs, the processor updates the channel registers to reflect the status of the current memory access.

To avoid overwriting the registers, further interrupts are blocked, and the Freeze bit is set to reflect the use of the channel registers. A quick snapshot of the instruction memory's access pipeline is taken and stored in the dedicated registers PCO-PC2. These buffer registers act as program counters to restart the Am29200 instruction pipeline after the exception.

The three memory-channel registers are used to indicate the state of memory activity before the exception. Information about the type of operation and register in use when the bus error occurred is stored in the channel-control register (CHC). The location of the error is stored in the channel-address register (CHA). Information in the CHC register and the indirect register pointer (IPsrc) can be used to implement pseudodevice operations. This register has a NotNeeded flag which can abort the memory operation causing the bus-error exception trap.

The architecture of the AMD 29k processors means that all 192 local and general registers have direct access to the ALU and memory. However, the special registers (SR) must be moved into a local or general register via a fast register-to-register operation before they can be manipulated.

Listing 6 shows exception handling of LOAD and STORE values on the pseudodevice registers. First, the virtual COFFEEPOT address in CHA is translated into physical memory space (PCOFFEEPOT). Next, CHC is modified to abort the old memory access. Information about the register used during the memory operation is then saved into IPsrc, which controls the indirect use of any register as a source register for a later instruction.

Listing 7-A bus error exception handling routine can be used to implement limited READ and WRITE operations on MC68332 virtual device registers. Some decoding of the faulting instruction and modification of the stack frame is necessary.

```

CLEAR-RR      equ    $FDFF ; Clear NotNeeded flag
READ-WRITE    equ    $40  ; Read/write* flag
SRC_MODE      equ    $38  ; Mask for determining READ mode
HAS-OFFSET    equ    $28  ; MOVE.L OFFSET(Ax),D0 type instruction

FAULT-ADDRESS equ    $14  ; Address on stack after register save
SSW           equ    $22  ; Special Status Word
ORIG_INSTR    equ    $1C
ORIG_RETURN   equ    $E

dev_address   equ    A0          register definitions
instr_address equ    A1
func_address  equ    A1
traptemp     equ    D1

PSEUDODEVICE: ; only move INDIRECT using D0 register!
    MOVEM.L D1/A0-A1,-(SP); (Create temp. registers)
    MOVE.L #FAULT_ADDRESS,dev_address;(Do memory translation)
    ADD.L (PCOFFEEPOT COFFEEPOT),dev_address

    MOVE    SSW(SP),traptemp
    AND     #READ_WRITE,traptemp ; (Was it a READ or a WRITE?)
    BNE    HANDLEREAD ; if (WASWRITE)

HANDLEWRITE:
    ANDI.W #CLEAR_RR,SSW(PC); (Tell processor, we'll do writes)
    MOVE.L D0,(dev_address) ; *dev_address = D0;
    MOVEM.L (SP)+,D1/A0-A1 ; (Recover registers)
    RTE

HANDLEREAD:
    MOVE.L (dev_address),D0 ; else D0 = *dev_address ;

; The rest of the READ gets trickier as we must decode the faulting
; instruction and adjust the RETURN PC on stack accordingly
; 1 word if we SEE it's indirect, 2 words if with outLANDish offset
    MOVE.L ORIG_INSTR(SP),instr_address;(Get instruction addr)
    MOVE.W (instr_address),traptemp ; (then orig instruction)
    ADD.L #2, instr_address ; (Get over instruction word)

    AND.L #HAS_OFFSET,traptemp
    CMP.L #HAS_OFFSET,traptemp
    BNE    READCONTINUE ; (If OFFSET, inc RETURN PC)
    ADD.L #2,instr_address ; (over the constant)
READCONTINUE: ; (Restore fixed RETURN PC)
    MOVE.L instr_address,ORIG_RETURN(SP)
    MOVEM.L (SP)+,D1/A0-A1 ; (Recover registers)
    RTE

```

Finally, the channel-control register is examined to see whether a LOAD or STORE operation is needed. This section of code makes use of another unique feature of the 29k architecture. COMPARE instructions set a boolean flag that can be stored in any register. This construct speeds the operation of the pipeline. The LOAD or STORE indicator bit in CHC is shifted by an S L L instruction into this boolean flag position to control the conditional jump that follows.

After the necessary operation to the pseudodevice registers, an I RET instruction initiates recovery. It "un-freezes" the processor, refills the instruction pipeline, and enables the interrupts again, thereby servicing the pseudodevice. No memory operations are required to recover registers or other status information during the exception return.

For more detailed information on the 29k exceptions, read the Am29200 user manual or Dan Mann's book [2].

Listing 6-The full COFFEEPOT Am29200 virtual device description includes JMP table information to add functionality into the device's operation. The MC68332 virtual device description is similar.

COFFEEPOT:

```
.equ C_CONTROL,0           ; C-CONTROL offset
.word 0                    ; C-CONTROL register value
.word STANDARD            ; C-CONTROL LOAD function
.word FUNC_CONTROL       ; C-CONTROL STORE function
.word 0

.equ C_HEAT,0x10          ; C_HEAT offset
.word 0                    ; C_HEAT register value
.word STANDARD           ; C_HEAT LOAD function
.word FUNC_HEAT          ; C_HEAT STORE function
.word 0

.equ C_TEMPERATURE,0x20  ; C-TEMPERATURE offset
.word 0                    ; C-TEMPERATURE register value
.word FUNC_TEMPERATURE  ; C-TEMPERATURE LOAD function
.word STANDARD          ; C-TEMPERATURE STORE function
.word 0
etc.
```

Listing 9-A JMP table provides functionality on the Am29200 pseudodevice.

HANDLELOAD:

```
LOAD 0,0,useindirect,dev_address register = *dev_address;
JMP JUMPTABLEACTION ; (In belay slot)
ADD traptemp,dev_address,4 ; (prepare for LOAD func access)
```

HANDLESTORE:

```
STORE 0,0,useindirect,dev_address ; *dev_address = register:
ADD traptemp,dev_address,8 ; (Prepare for STORE func access)
```

JUMPTABLEACTION:

```
LOAD 0,0,func_address,traptemp ; (Grab JUMP address)
JMPI func_address ; (and do it)
NOP
```

STANDARD:

```
; When any register accessed
LADDR dev_address,P_COFFEEPOT; (Get base address)
LOADindexed dev_address,C_TIMER,traptemp ; C_TIMER++
ADD traptemp,traptemp,1
STOREindexed dev_address,C_TIMER, traptemp
LOADindexed dev_address,C_TEMPERATURE,traptemp ;C_TEMPERATURE--
SUB traptemp,traptemp,1
STOREindexed dev_address,C_TEMPERATURE,traptemp
IRET
```

FUNC_CONTROL:

```
; When C-CONTROL reg accessed
CONST traptemp,0
STOREindexed dev_address,C_TIMER,traptemp
STOREindexed dev_address,C_TEMPERATURE,traptemp
STOREindexed dev_address,C_HEATER, traptemp
JMP STANDARD
NOP ; (Could be filled)
```

FUNC_HEAT:

```
; when C-HEATER register accessed
LADDR dev_address,PCOFFEEPOT ; (Get base address)
LOADindexed dev_address,C_TEMPERATURE, traptemp
; C-TEMPERATURE += C_HEAT
ADD traptemp,useindirect, traptemp ; NOTE ORDER
STOREindexed dev_address,C_TEMPERATURE, traptemp
JMP STANDARD
NOP ; (Could be filled)
```

IN SMALL PACKAGES

BLACKJACK TELECONTROLLER



Starting at **\$139**

Micromint's

BlackJack-552 is a true "telecontroller" incorporating both microcomputer and FCC-certified Xecom modem in a single package. BlackJack comes preprogrammed with firmware utilities which are optimized for assembly language, C, and BASIC programs. Like Domino, BlackJack is easy to use. Attach power and a terminal, then upload, store, and execute your program.

SPECIAL FEATURES

- 80C552 processor with firmware monitor, 14.7456 MHz
- Serial I/O; (up to 38,400 bps); full-duplex TTL, I²C bus
- Connection: 1.2" dual-row 48-pin DIP header
- Parallel I/O: 20 bits, 4 bits shared with RTC
- Two interrupts: three timers, watchdog timer, two PWM outputs
- Hardware real-time clock
- A/D converter; 8 channels, 10 bits, 20k samples/sec.
- 2400 bps data modem (data/fax available)
- Power: +5V @ 55 mA;
- Size: 2.75" x 1.375" x 0.5" potted
- 32K bytes SRAM, 32K bytes ROM and 128K bytes EEPROM
- Industrial temperature available



4 Park Street • Vernon, CT 06066

Call 1-800-635-3355

(860) 871-6170 • Fax (860) 872-2204

THE MC68332'S VIRTUAL DEVICE

Differences between the architecture of the MC68332 CISC and Am29200 RISC microcontrollers show up immediately when exception handling is examined more closely.

Listing 7 shows how reading and writing of the MC68332 virtual device registers are controlled. Before the bus error's exception-handling routine is entered, any register altered by the fault instruction is automatically restored to its original value. A lot of status information (see Table 3) must also be stored to and later recovered from the external-memory stack. This feature contrasts with the faster register-to-register operations in the Am29200 exception handler.

In the MC68332 exception-handling routine, you first free up some temporary registers. This step requires further memory accesses since registers D1, AO, and A1 are saved to the external stack. As we'll see later, register DO plays a critical role in the device simulator and can't be used as a temporary register.

The Special Status Word (SSW) stored on the stack plays the same role as some of the bits in the Am29200's CHC. The SSW contains information of the size and type of data transfer, key cycles, and function code. The RR bit in the SSW is the MC68332 write operation's NotNeeded bit. The RW bit describes whether a read-from or write-to memory operation accessed the virtual device.

The CISC processor's virtual-device operations present new problems. For a start, there is no equivalent to the Am29200's IPsrc register that enabled us to determine which data register or addressing mode is being used.

To overcome this problem, we need to use the information on the stack to find the instruction accessing the device registers. The effective address then has to be decoded. Such a complete simulation is, for present purposes, probably not necessary.

It is far simpler to permit only virtual-device memory operations to

Stack Address	Contents
SP	copy of Status Register
SP + \$2	return PC high word return PC low word
SP + \$6	vector info
SP + \$8	faulting address high word faulting address low word
SP + \$C	DBUF high word DBUF low word
SP + \$10	faulting instruction PC high word faulting instruction PC low word
SP + \$14	internal transfer count register
SP + \$16	Special Status Word

Table 3—MC68332 stack information is shown for a bus error exception.

use the DO register, which is not as limiting as it might seem. It is common programming practice to use DO as a temporary register in code accessing external devices. However, it is important to ensure that this practice is followed in compiler-generated code.

A bigger problem is determining which MC68332 CISC instruction accesses the device. On the Am29200

RISC processor, you LOAD or STORE any register using an address stored in any another register. On the MC-68332, however, you can access memory with many instructions. A lot of code would be needed to simulate all possible instructions and addressing modes.

Support for ADD and SUBtract instructions is not a normal feature with device registers, so the simulation is not limited significantly if we require that only indirect move-write instructions are used on the virtual-device registers. Thus, operations like these are permitted:

```
; 2-byte instruction
MOVE.L DO,(Ax)
; 4-byte instruction
MOVE.L DO,CONST(Ax)
```

These operations use one of the MC-68332 address registers (Ax) and a possible fixed offset (CONST) from the device's base address.

Listing 10—Functionality for the MC68332 pseudodevice is provided via the JMP table of Listing 8

```
HANDLEREAD:
    MOVE.L (dev_address),DO    ; DO = *dev_address ;
    ....
READCONTINUE:
    MOVE.L instr_address,ORIG_RETURN(SP)
    MOVE.L dev_address,func_address
    ADD.L #4,func_address    ; (Set up function pointer)
    JMP     JUMPTABLEACTION

HANDLEWRITE:
    MOVE.L DO,(dev_address)    ; *dev_address = DO;
    MOVE.L dev_address,func_address
    ADD.L #8,func_address    ; (Set up function pointer)

JUMPTABLEACTION:
    MOVE.L (func_address),func_address    ,(Grab JUMP address)
    JMP     (func_address)    ; (and do it)

STANDARD:    ; (When ever a device register is manipulated)
    MOVE.L #P_COFFEEPOT,dev_address    ; (Set base address)
    ADD.L #1,C_TIMER(dev_address)    ; C_TIMER++
    SUB.L #1,C_TEMPERATURE(dev_address); C_TEMPERATURE--
    MOVEM.L (SP)+,D1/A0-A1    ; (Recover temp. registers)
    RTE

FUNC_CONTROL:    ; when C-CONTROL register accessed
    MOVE.L #0,C_TIMER(dev_address)
    MOVE.L #0,C_TEMPERATURE(dev_address)
    MOVE.L #0,C_HEATER(dev_address)
    JMP     STANDARD

FUNC_HEAT:    ; when C-HEATER register accessed
    MOVE.L #P_COFFEEPOT,dev_address    ; (Set base address)
    ADD.L DO,C_TEMPERATURE(dev_address); C_TEMPERATURE += C_HEAT
    JMP     STANDARD
```

Listing 11—Am29200 registers must be saved, the processor unfrozen, and the interrupts released prior to operating system calls in the middle of a bus error interrupt handler.

```

FUNC_TEMPERATURE:
    SAVEREGISTERSANDUNFREEZE ; macro

; HIF(WRITESERVICE, STDOUT, MESSAGEPT, MESSAGELENGTH
    CONST gr121, 20 ; WRITESERVICE
    CONST lr2, 1 ; STDOUT
    LADDR lr3, MESSAGE ; MESSAGEPT
    CONST lr4, 15 ; MESSAGELENGTH
    ASNE 0x45, gr1, gr1 ; HIF0;
    FREEZEANDRECOVERREGISTERS; macro
    IRET

MESSAGE: .ascii "COFFEE PERKINGn"
    .align

.macro SAVEREGISTERSANDUNFREEZE
    MFSR traptemp, PC0 ; save FROZEN PC0
    SUB msp, msp, 4
    STORE 0, 0, traptemp, msp
; Save also SR registers PC1, ALU, CHA, CHD, CHC. OPS
; and standard register gr96
    CONST traptemp, 0x400 ; FREEZE BIT
    MFSR traptemp2, CPS ; being cleared
    ANDN traptemp2, traptemp2, traptemp
    MFSR CPS, traptemp2

    SUB msp, msp, 12 ; Burst save of lr2, lr3, lr4
    MFSR CNT, (3 1) ; registers
    STOREM 0, 0, lr2, msp
    MFSR traptemp2, CPS
    ANDN traptemp2, traptemp2, 0x3 ; Reenable interrupts
    MFSR CPS, traptemp2
.endm

.macro FREEZEANDRECOVERREGISTERS
    MFSR traptemp2, CPS ; Disable interrupts
    OR traptemp2, traptemp2, 0x3
    MFSR CPS, traptemp2

    MFSR CNT, (3 1) ; Burst recovery of lr2, lr3, lr4
    LOADM 0, 0, lr2, msp ; registers
    ADD msp, msp, 12

    CONST traptemp, 0x400 ; REFREEZE processor
    MFSR traptemp2, CPS
    OR traptemp2, traptemp2, traptemp
    MFSR CPS, traptemp2

; Recover SR registers PC1, ALU, CHA, CHD, CHC, OPS and gr96
; registers in the reverse order were pushed onto the stack
    LOAD 0, 0, traptemp, msp
    MFSR PC0, traptemp ; recover FROZEN PC0
    ADD msp, msp, 4
.endm

```

Even with these restrictions, simulating the following READ instructions gets more complicated:

```

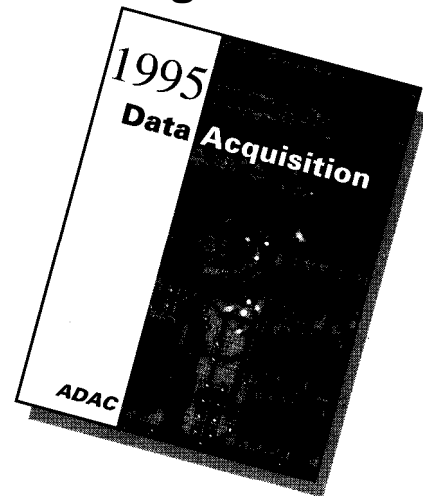
; 2-byte instruction
MOVE.L (Ax),DO
; 4-byte instruction
MOVE.L CONST(Ax),DO

```

Listing 7 indicates the extent of this complexity.

After a bus error produced by a write operation, the exception-stack frame has a return-PC value that corresponds to the instruction after the faulting WRITE instruction. In contrast, the same return-PC value for the

FREE Data Acquisition Catalog



PC and VME data
acquisition catalog
from the inventors of
plug-in data acquisition.
Featuring new low-cost
A/D boards optimized
for Windows,
DSP Data Acquisition,
and the latest
Windows software.
Plus, informative
technical tips and
application notes.

Call for your free copy

1-800-648-6589

ADAC

American Data Acquisition Corporation
70 Tower Office Park, Woburn, MA 01801
phone 617-935-3200 fax 617-938-6553
info@adac.com

read bus-error fault points to the instruction that caused the fault.

This address must be modified by 2 or 4 bytes, depending on whether the indirect **MOVE** has an offset. The faulting instruction must be fetched from memory and its effective addressing mode must be determined.

At the end of the MC68332 exception handler, the original values of the temporary registers must be recovered from memory. Issuing **RT E** recovers other status information from the stack. The processor restarts program execution after the memory access that produced the bus error. Again, the pseudodevice has been serviced.

I gained information on bus-error handling from Thomas Harman's book on the MC68332 microcontroller [3], the CPU32 CPU reference manual, and trial-and-error techniques.

GETTING ACTION

Up to this point, we have provided the virtual device with registers that can be written to or read from. However, it still has no functionality.

One way to introduce functionality is to place **J M P** table information directly into the device description. Listing 8 shows the format for part of this new device description and Listing 9 shows how to introduce functionality into the Am29200 simulation.

Information about the virtual device register is contained in `dev_add_registers`. If this register is incremented by 4 or 8, it points to a location containing the address of the function to perform following a read or write operation on a device register. An indirect **J M P** with the appropriate start address switches the exception handler to the required function.

Each function is completed by jumping to **STANDARD**, which modifies **TIMER** and **TEMPERATURE** registers to better model device properties.

Listing 10 enables the functionality of the MC68332 virtual coffee pot. Note the similarity with Am29200's code. Register **DO** doesn't need saving since its functions don't change. Functionality on other devices may require **DO** and/or additional data registers that need saving to the stack so access remains transparent.

Listing 12—This is a technique for making calls to the MC68332 board's operating system during an exception handler. The saving of important status registers to external memory is handled automatically by the processor.

```
FUNC_TEMPERATURE:
MOVE.L DO, -(SP) ; (Used as temp register during SYS call?)
MOVE SR, -(SP) ; (Not necessary for this example?)
; (Monitor did I/O without needing changes to interrupt levels)
PEA MESSAGEPT
TRAP 15 ; SYS(WRITESERVICE, MESSAGEPT)
DC.6 WRITESERVICE
MOVE (SP)+, SR ; (Recover status register and DO)
MOVE.L (SP)+, DO
JMP STANDARD
MESSAGEPT:
DC.6 14, "COFFEE PERKING"
.align 4
```

FOR FLAVORED COFFEE

A simulation should also display status or debug messages. These features involve using the evaluation board's operating system to perform system calls to communicate with the PC providing a keyboard and screen. This communication typically occurs over a serial communication link between the PC and evaluation board.

To use this link, registers used during the call must be saved. Also, the interrupts for the boards must be reactivated so serial communications can be reestablished. This reconnection is done by the operating system on the MC68332 evaluation board, but by the user on the SA29200 board.

Listing 11 demonstrates how to make use of calls to the Am29200 evaluation board's high-level interface (HIF) during a bus-error exception handler. A status message about the coffee-pot operation displays each time the **C-TEMPERATURE** register is read.

Two steps reactivate the interrupts and unfreeze the processor during the current exception handling routine. First, save the frozen registers and any other necessary registers.

Dan Mann [2] describes a fast context-switching approach that sets aside Am29200 registers gr68–gr79 for a register cache that stores the frozen special registers. To simplify this simulation, all the registers are stored to external memory using the Memory Stack Pointer (MSP). MSP must be distinguished from the Register Stack Pointer (RSP), used to control the RISC register window cache (lr0–lr 127).

After saving the Am29200 special registers, the processor must be unfrozen and the interrupts reenabled. Because of the way the pipeline buffer (PCO-PC 1) is reestablished, two instructions must occur between the processor unfreezing and enabling the interrupts. After the **H I F** call, the inverse operations must occur before an **I RET** call completes the bus-error exception-service routine.

Since the **H I F** call uses registers lr2–lr4 and gr96, they must also be saved on the memory stack. The channel registers become active again once the processor is unfrozen. Registers can then be fast-stored using a store-multiple instruction and recovered with a load-multiple instruction. **STOREM** and **LOADM** take advantage of the faster burst-access memory mode.

Listing 12 demonstrates a **SY S** call to the MC68332 board's monitor to display a status message. The code appears simple because most operations occur internally to the **SY S** call, not externally as with the SA29200 board's **H I F** call.

It is not clear whether local registers **DO**, **D1**, **AO**, and **A1** are modified during the **SY S** call, so the safest approach is to save them onto the stack. Only **DO** must actually be saved, since the others already have been placed on the stack. The other status information needed to restart the user program was stored to the stack when the original bus-error trap occurred.

It is not necessary to modify the status register in order to reenable the interrupts and reestablish the commu-

nications link. This happened automatically during the SY S call.

WHERE TO NEXT?

For more comprehensive memory operations on the Am29200 and MC68332 pseudodevices, a small amount of additional code could offer memory operations with 8-, 16-, or 32-bit access. However, this doesn't solve the problem that MC68332 pseudodevice access is limited to one data register (DO) and indirect MOVes.

Advanced Micro Devices has both an architectural and an instruction set simulator for the Am29200. Software Development Systems provides an MC68332 simulator in their C/C++ 68k starter kit. A really caffeine-free next step would be to hang COFFEEPOT on pseudoprocessors!

I hope to examine other techniques to get fuller functionality out of virtual devices. I'd like time-accurate simulations and calls accessing replay files on the PC driving the evaluation board. The pseudodevice could then be tested with real data. □

Special thanks to the University of Calgary for a sabbatical to follow up on research and teaching ideas.

Mike Smith is a professor in the Electrical and Computer Engineering Department at the University of Calgary, Canada. He teaches courses on Ctt, microprocessor interfacing, and comparative processor architecture and does research into digital-signal-processing applications for magnetic-resonance image reconstructions. Mike may be reached at smith@enel.ucalgary.ca.

language programming and interfacing, Prentice Hall, 1991.

CONTACTS

Advanced Micro Devices
901 Thompson Pl.
Sunnyvale, CA 94088
(408) 732-2400
Hotline: (800) 2929-AMD

Motorola
2100 East Elliot
Tempe, AZ 85284
(602) 244-6900
Fax: (602) 952-4067

Software Development Systems
815 Commerce Dr., Ste. 250
Oak Brook, IL 60521
(708) 368-0400
Fax: (708) 990-4641

REFERENCES

- [1] Motorola, "Gathering Requirements," *Motorola University Course Notes*, 1993.
- [2] Dan Mann, *Programming the 29k RISC Family*, Data Book AMD-19243BD, 1995.
- [3] Thomas Harman, *The Motorola MC68332 Microcontroller-product design, assembly*

IRS

407 Very Useful
408 Moderately Useful
409 Not Useful

TIRED OF WAITING FOR THE PROMPT ?

Speed up your ROM DRIVE! Also DOS and programs instantly. Also used to replace mechanical drive completely. Control light of diskless workstations. Easy installation. Size card.

MVDISK1 128k 575
MVDISK2 1 44m 5150
MVDISK3 5.76m 5195

\$75

Quantity discounts!

DOS IN ROM!



\$95 EPROM PROGRAMMER

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



WORLDS SMALLEST PC !!!

ROBOTS ALARMS RECORDERS DOS

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

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

FREE SHIPPING IN U.S.

5 YEAR LIMITED WARRANTY

MVS Box 850
Merrimack, NH
(508) 792 9507

8088 SINGLE BOARD COMPUTER

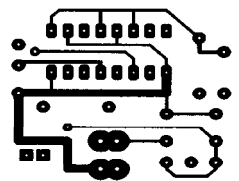


CADPAK & PROPAK for Windows

PROFESSIONAL SCHEMATIC CAPTURE AND PCB SOFTWARE FOR WINDOWS *NEW!*

LOW PRICES!

STARTS AS LOW AS \$299



Professional schematic & PCB design Software for Microsoft Windows 3.1

- Automatic wire routing & dot placement.
- Exports diagrams to other applications.
- Full control of drawing appearance.

PROVIDES all the Convenience of Windows & the POWER of CAD Software

- Advanced routing
- Libraries, including SMT
- Includes Gerber Viewing
- Fast & Easy to use.

R4 SYSTEMS INC.
1100 GORHAM ST. SUITE 118-332
NEWMARKET ONTARIO
CANADA L3Y 7V1
905.898.0665 FAX 905.898-0683

FREE DEMO
WRITE OR CALL
TODAY

Internet r4system@astral.magic.ca

FEATURE ARTICLE

Ed Lansinger

Developing an Engine Control System Part 3: Completing the System

Ed finishes his series on the engine control system by looking at the last critical parts: the crankshaft position detector, automatic shutdown, and power supply. He ends with how to tune the complete system.

Ohis month, I wrap up my series on the electronic fuel-injection system I developed for Rensselaer's race car. I created the Motorola 68HC16-based system while a member of the Formula SAE team. In the past two articles (*INK* 62 and 63), I covered the fuel- and spark-delivery subsystems.

Three simple but important circuits round out the hardware: crankshaft-position detection, automatic shutdown, and power supply. In software, the `Distributor` and `AlarmClock` objects catch the interrupts that initiate all software activity, and the `MAP` object reads the air-pressure sensor. I finish by describing the basic process of testing and tuning the complete system.

HARDWARE

To synchronize control events with crankshaft position, the engine (from a Yamaha FZR600 motorcycle) has teeth cut in its metal flywheel. As the engine turns, the teeth pass close to a variable-reluctance pickup called the *crankshaft-position sensor* (CPS). Figure 1 shows this arrangement.

The sensor produces a current pulse as the teeth pass. The pulse's direction is determined by whether the

edge of the tooth is leading or trailing. The pulse is detected by the crankshaft position-sensing circuit in the Engine Control Module (ECM). Figure 2 shows the crankshaft position-sensing circuit.

The circuit's core is the full-wave bridge rectifier formed by D1-D4 that is open at one end. A current pulse loops up through D2, into the base of Q2 (turning it on), to ground, through D3, then back to the sensor. A current pulse traveling in the opposite direction makes a similar loop but through D1, Q1, and D4.

Q1 and Q2 pull low two separate lines that are tied high to +5 V. These lines are interrupt lines to the processor. In this way, the processor is interrupted on both leading and trailing edges and can tell them apart. The ability to differentiate between the two is important for detecting the unique Top-Dead Center (TDC) tooth [explained in "Software"].

Q1 and Q2 must be high-gain transistors because the crankshaft position sensor doesn't push a lot of current at low RPMs. TIP120s work fine and are ubiquitous, although their power-dissipation rating is unnecessarily high for this circuit.

More subtle is the requirement that D1-D4 be germanium diodes like the 1N34A. At cranking speeds, the CPS produces output voltages barely more than a couple of volts. The forward voltage drop of two silicon diodes (0.7 V each) and the transistor's base-to-emitter voltage drop (about 1.4 V) blocks this signal.

By using germanium diodes (forward voltage drop of only 0.3 V), the overall drop reduces enough to ensure reliable starting. This problem goes away at higher engine speeds. In fact, the CPS develops open-circuit voltages of several hundred volts near the red-line!

The Automatic Shutdown (ASD) watchdog circuit controls the power supply to the ignition coils, fuel pump, and injectors. If the software hangs or the car stalls, the ASD circuit (see Figure 3) shuts off power to these devices within 0.5 s.

This shutdown protects driver, engine, and electronics if the ECM fails. If the system conks out, you

cannot have the injectors remaining on, filling the engine with fuel while the coils get toasty and the drive circuitry overheats. The ASD circuit turns on the relays supplying power directly. It is kept active by a line from the CPU that the software must toggle rapidly.

Rapid toggling of the line coming from the 68HC 16 keeps voltage low on C1 and C2 to prevent the relay from being shut off. When the input is a logic high, transistor Q1 discharges capacitor C1 and transistor Q2 is off, which allows C2 to charge. If we're lucky, the CPU changes the state of the input line before the voltage on C2 gets above a reference set by R1, R2, and R3.

When the input changes from high to low, C2 is discharged by Q2, and C1 begins to charge. However, if either capacitor gets sufficiently charged, comparators U2a or U2b detect it and turn off the relay that gates power to the coils, injectors, and fuel pump.

The software toggles the line when it receives a pulse from the CPS, so the ignition and fuel subsystems turn off in the event of a stall. The values of C1, C2, R8, and R9 set the minimum toggle rate. Since the voltage reference is $\frac{2}{3}$ the supply to the capacitors, the reference voltage is reached in about $R \times C$ seconds. In this case, the line must be toggled at least once every 0.5 s.

This time must be slower than the rate that crankshaft position pulses

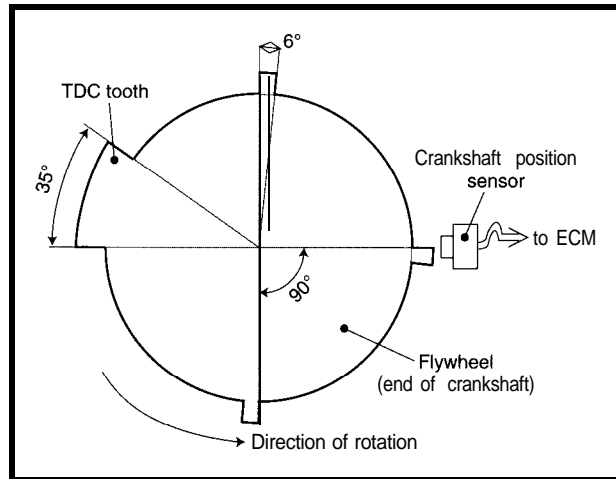


Figure 1-A variable-reluctance sensor detects teeth on the flywheel to determine engine position.

generated when the starter motor turns off.

Current for the starter motor comes directly from the battery. For simplicity, it is controlled by a dashboard switch, rather than the ECM.

occur at cranking speeds. Our engine typically cranks at around 200 RPM. There are four teeth on the crankshaft, so the time between pulses is generally no longer than 75 ms. When the engine is cranking very slowly during cold starts or when push-starting the car, the extra time allowed by that 0.5 s toggle helps, and it's still a short enough time for protection.

The power supply (see Figure 4) is straightforward. The car battery supplies +12 VDC, which is filtered and passed to 7805 and 7808 voltage regulators. The 7805 needs a large heatsink because the total current drawn by the ECM is near capacity. The 7808 needs no heatsink.

Diode D1 provides reverse-voltage protection by conducting and blowing the fuse. Diode D2 is a transient-suppression diode which clamps any potentially harmful voltage spikes

The switch turns on an automotive-starter relay, sending current to the starter motor.

Starting the engine draws about 100 A from the battery, which is enough to lower the battery output voltage. The ECM's performance degrades as battery voltage drops mainly because the ignition coils don't charge sufficiently and the engine doesn't crank quickly. At 10 V, the engine runs poorly, and at 9 V, it won't run at all, so the battery must be well charged before starting. A booster battery helps during long periods of cranking.

Shock and vibration are the biggest physical dangers to the ECM. If components are not properly mounted and flat to the board, solder joints fatigue and crack. For extra durability, I used silver-based solder. I mounted the boards rigidly to their enclosure with short standoff. Rubber shock mounts attached the enclosure to the vehicle's frame.

I then secured all wiring leading to the box, to ensure that the box wasn't supporting the weight of the wires. If you're really confident in your circuit, potting it in epoxy is the way to go.

SOFTWARE

To keep injector and ignition software synchronized with engine position, I created the **Distributor** software object. Figure 5 shows its data members and messages.

Distributor mimics the action of a mechanical distributor in determining which cylinder should fire next. It also supplies information to

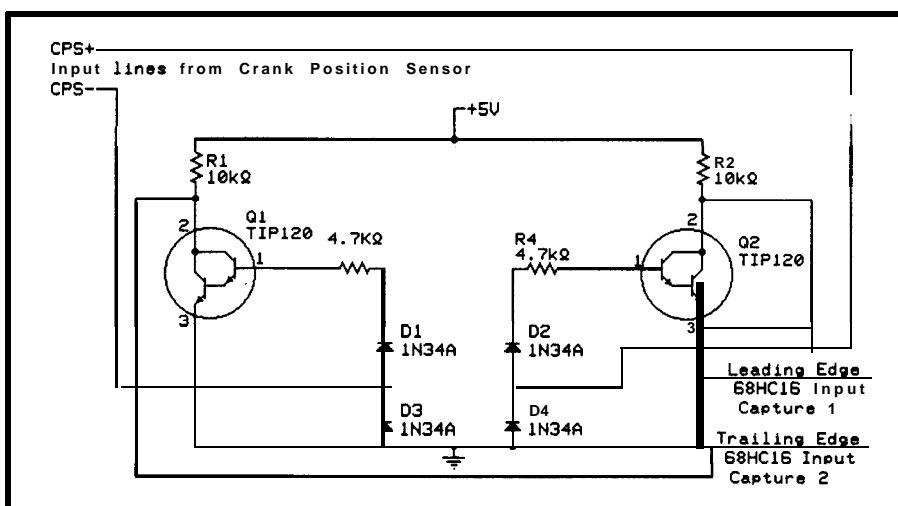


Figure P-Germanium diodes help this circuit detect small current pulses from the crank position sensor at low speeds.

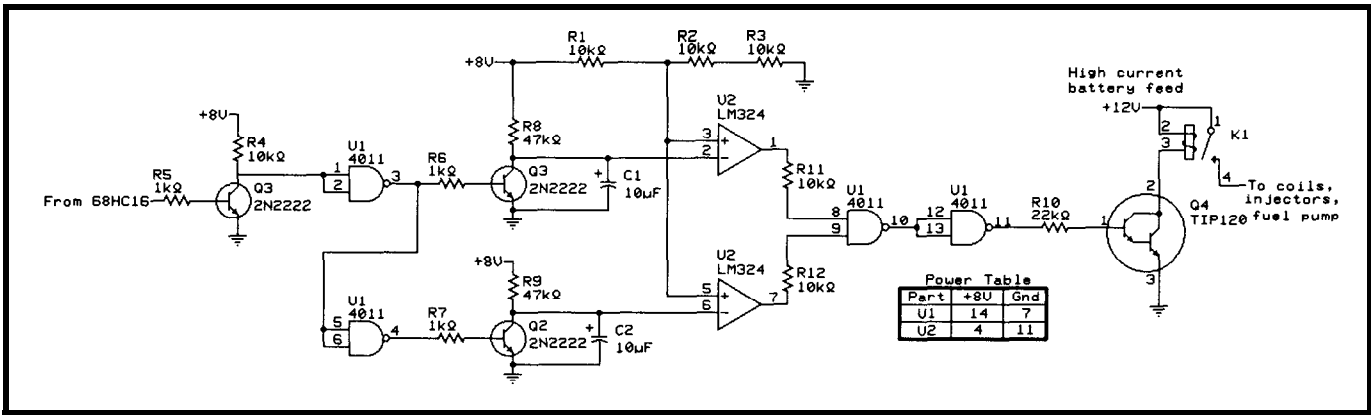


Figure 3-The automatic shutdown circuit turns the ignition and fuel off in the event of a malfunction.

other objects about engine speed and manages the rev limiter.

Determining engine position is simple because of the shape of the teeth passing the CPS. Engines have different sense-tooth arrangements. Position sensing typically amounts to detecting a variation in a tooth-and-gap pattern. This engine has three short teeth, each covering about 6" of crankshaft rotation, and one long, covering about 35". Leading edges are spaced evenly at 90". The long tooth's

leading edge passes the sensor before the piston in cylinder 1 reaches TDC.

The piston in cylinder 4 is in phase with the piston in cylinder 1. Pistons in cylinders 2 and 3 are 180" out of phase, so when 1 and 4 are at TDC, 2 and 3 are at bottom-dead center. This is a common arrangement for inline, 4-cylinder engines. (When an event occurs n° before or after TDC, that is the position of the specific cylinder's piston. When I refer to a timing tooth, such as the TDC tooth,

that tooth's position is relative to the position of cylinder 1's piston.)

As mentioned, the teeth's leading and trailing edges produce unique interrupts. These interrupts send the `LeadingEdgeInterrupt()` and `TrailingEdgeInterrupt()` messages to `Distributor`, which allows `Distributor` to determine engine position easily.

I used the Motorola 68HC16's Input Capture inputs to capture each signal, record the time when it oc-

RTC-HC11

PROCESSOR BOARD

Offering an exceptional value in a single-board embedded controller, Micromint's RTC-HC11 combines all of the most-asked-for features into a compact 3.5" x 4.5" package at a reasonable price. Featuring the popular Motorola MC68HC118-bit microcontroller, the RTC-HC11 gives you up to 21 lines of TTL-compatible I/O; an 8-bit, 8-channel analog-to-digital converter; two serial ports; a real-time clock/calendar with battery backup; 512 bytes of nonvolatile EEPROM; and up to 64K of on-board RAM or EPROM, 32K of which can be battery backed.

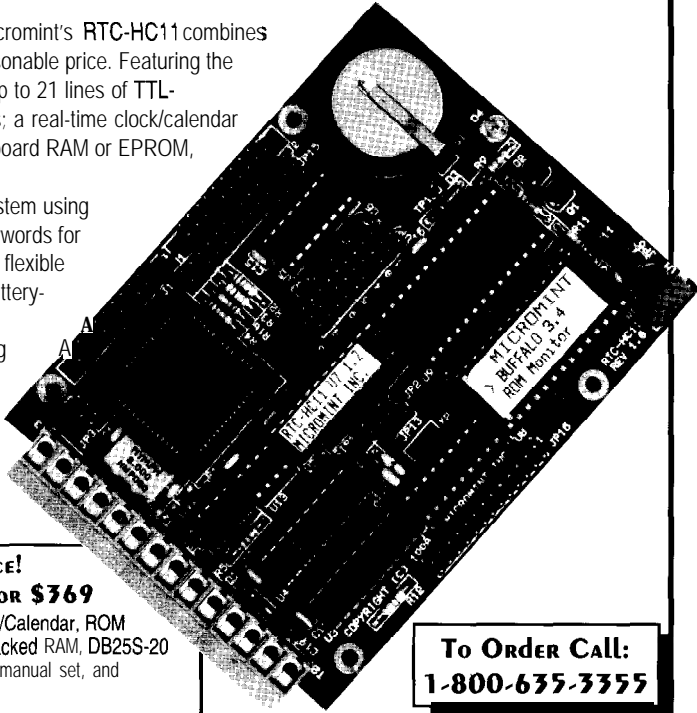
Software development can be done directly on the RTC-HC11 target system using BASIC-11, an extremely efficient integer BASIC interpreter with dedicated keywords for I/O port, A/D converter, timer, interrupts, and EEPROM support. In addition, a flexible configuration system allows a BASIC program to be saved in the on-board, battery-backed static RAM, and then automatically executed on power-up. Micromint also offers several hardware and software options for the RTC-HC11 including the full line of RTC-series expansion boards as well as an assembler, ROM monitor, and a C language cross-compiler.


Additional features include:

- Asynchronous serial port with full-duplex RS-232 and half-duplex RS-485 drivers
- 1-MHz synchronous serial port
- CPU watchdog security
- 5-volt-only operation
- RTC stacking expansion bus

Special Development System Price!
RTCHC11-DEV A \$414 value for \$369
 Board w/8-bit ADC, EEPROM, 8K RAM, Clock/Calendar, ROM monitor, BASIC-11 in EPROM, 32K battery-backed RAM, DB25S-20 serial cable, utilities diskette (PC compatible), manual set, and HCTerm software.
 Other configurations starting at \$199

To Order Call:
1-800-635-3355





MICROMINT, INC. 4 Park Street • Vernon, CT 06066 • (860) 871-6170 • Fax (860) 872-2204
 in Europe: 1285-658122 • in Canada: (514) 336-9426 • in Australia: (3) 9467-7194 • Distributor inquiries invited!

curred, and cause an interrupt. The `TrailingEdgeInterrupt()` message just records the time when the edge occurred. The `LeadingEdgeInterrupt()` message starts the processing.

Distributor tracks the time of the last leading and trailing edges. When it receives the next tooth's leading edge, it figures out the previous tooth's length and the length of the gap between teeth.

Instead of comparing tooth widths, I compare the gap's width to that of the tooth before the gap. When a leading edge passes, the previous gap's length equals the difference between the timestamps of the current leading-edge and the last trailing-edge.

The length of the previous tooth is the difference between the last trailing-edge timestamp and the last leading-edge timestamp. If half the gap's width is shorter than the tooth before it, I know the previous tooth was the TDC tooth, so the current tooth must be the one 90° after TDC.

Otherwise, I assume that the engine has rotated another 90° from its previous position. Although engine speed can vary between teeth, this algorithm allows large fluctuations in engine speed to occur between teeth without confusing the pattern.

Once it has determined position, `Distributor` runs through a schedule of things that must happen at that position. First, it toggles the line to the ASD circuit to keep the fuel and ignition systems up. Next, it computes engine RPM and other useful quantities. Finally, it selectively fires injectors and ignition coils based on engine position and RPM.

Four `Injector` objects represent the four physical injectors. The `In-`

jector for a given cylinder receives an `Open()` message from `Distributor` when the piston for that cylinder is at TDC. The `Injector` objects turn themselves off at the right time with the help of the `AlarmClock` object.

The two `Coil` objects are activated somewhat differently. Each `Coil` must receive `Charge()` and `Fire()` messages. The `Charge()` message simply tells a `Coil` that a spark is to be fired sometime in the future, so start charging the coil now. `Fire()` tells the `Coil` that its cylinder pair is starting the last 90° of travel before TDC, so schedule a spark from the (already charging) ignition coil. This spark generally occurs 10–45° before TDC, depending on engine RPM.

`Distributor` sends `Charge()` far enough in advance to guarantee that the coil has enough time to charge. Below 3000 RPM, the message is sent 90° before TDC. Between 3000 and 6000 RPM, it is sent 180° before TDC. Above 6000 RPM, it is sent 270° before TDC. This keeps the coils from being on longer than necessary, reducing heat build-up.

Two issues with spark timing are not readily apparent. First, engine speed varies greatly while cranking, even between teeth. A spark scheduled to happen before TDC might fire after TDC if the engine speeds up in the interim. So, below about 300 RPM, `Distributor` sends `Charge()` messages at the usual time, but it also sends a special `FireNow()` message right at TDC. `FireNow()` tells `Coil` to fire the spark immediately, synchronizing it with the leading edge of a tooth and piston position. This technique greatly eases starting.

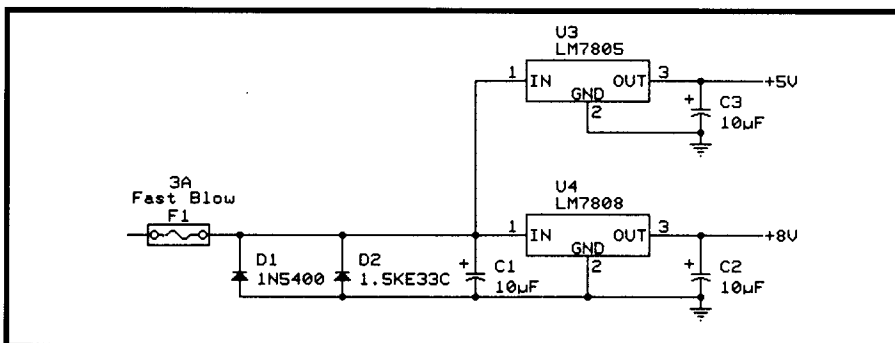


Figure 4—The power supply provides protection from reverse voltage as well as voltage spikes.

Smart Solutions

for Analog to RS-232/485 Conversion

MEC232-08*

Single chip analog to RS-232

- 4 8-bit input channels
- 6 bi-directional I/O lines
- Configurable sampling rates
- Low power
- Low cost

Express Fax Document #8002

MEC232-14*

High accuracy analog to RS-232

- 2 bipolar 14-bit analog inputs
- 6 bi-directional I/O lines
- Full scale accuracies to 0.015%
- Uses less than 12mW
- Minimal external components

Express Fax Document #8004

MEC485-14*

High accuracy analog to RS-485

- 2 bipolar 14-bit analog inputs
- 6 digital I/O lines
- On-board EEPROM stores address and I/O status
- Very simple to use

Express Fax Document #8005

*PATENT PENDING

MARTEL
ELECTRONICS

P.O. Box 897, Windham, NH 03087
800-821-0023 Fax: 603-898-6820

For more information call our Express Fax 800-821-6820 and request any of the above #'s.

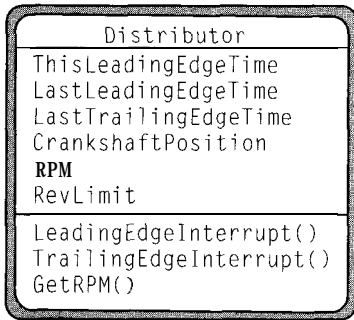


Figure 5—Injectors and ignition coils are synchronized by the *Distributor* software object.

Second, I'd like to start charging a coil at TDC at the highest RPMs to maximize charge time. Unfortunately, I can't. The spark duration for this ECM can be as high as 1.5 ms, which can easily extend past TDC at speeds beyond 11,000 RPM. (TDC as measured by the sensor is actually about 13° before mechanical TDC.) Therefore, charging the coil at the TDC signal interrupts the previous spark, preventing reliable ignition. Setting the earliest charge point at 270° before TDC works well for this system, even

though the coils do not charge to their fullest.

Distributor limits engine speed by cutting out cylinders above a certain RPM. If a cylinder is cut out by the rev limiter, its *Coil* and *Injector* stop receiving *Charge()* and *Open()* messages. *Coil* continues to receive *Fire()* messages to ensure it is fired safely if it has started to charge instead of remaining on indefinitely.

After *Distributor* sends the appropriate messages to *Coil* and *Injector*, it cleans up and returns from the interrupt, which lets foreground processing run. You can choose

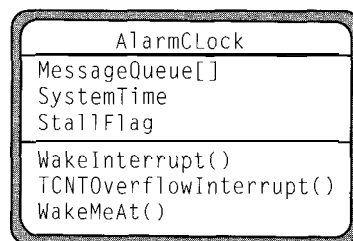


Figure 6—Software objects that need to "go to sleep" for a while call *AlarmClock* so they are reactivated at the right time.

what to do in foreground (e.g., data logging, an LCD display, traction control, etc.). The 68HC16 is fast enough that the engine control software uses only a fraction of the available CPU time, even at high engine speeds.

As *Distributor* synchronizes *Injector* and *Coil* with their mechanical equivalents, *AlarmClock* (see Figure 6) facilitates asynchronous processing. It provides wake-up calls to other objects so they can schedule future events.

Frequently, an object needs to perform an action at a future time. For example, *Injector* needs to turn off its output after a certain time elapses. Waiting in a busy loop won't work because other objects may need to run. Instead, *Injector* asks *AlarmClock* to wake it up at a future time by sending it the message specified in the wake-up request.

AlarmClock tracks which messages to send to different objects. It uses a dedicated interrupt to wake itself up when the first message in the queue is to be processed.

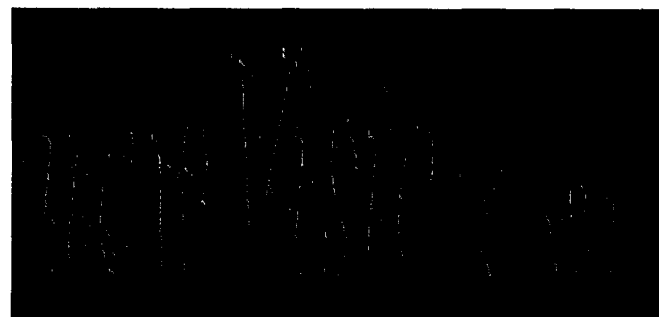
flash memory

Products start at \$79 qty 1

Now get C-programmable miniature controllers with non-volatile "flash" memory. Our **Dynamic™** development system makes it easy—only \$195. Call our **AutoFAX** today! Dial 916.753.0618 from your FAX and request catalog #18.

ZWORLD ENGINEERING

1724 Picasso Ave.
Davis, CA 95616
916.757.3737
916.7535141 FAX



newVision™ is a C/C++ object library containing over 1000 floating point and 8-bit tools for full-color image processing, rectification, enhancement, image analysis, object segmentation and classification, neural network applications and 2-D math/transform operations, etc. etc. It is designed to provide a complete set of functional tools for the development of 32-bit protected mode DOS, WINDOWS and OS/2 applications (A SUN based version is also available); all of which are supported in single greyscale mode or using 3-band full color processing, of RGB, HSI, and YIQ true-color. *newVision™* also supports the use of region-of-interest (ROI), area-selective application processing.

newVision™ comes with over 5000 lines (>100 programs) of sample codes demonstrating the use of all aspects of the libraries. No royalties. Supports most C/C++ protected mode compilers.

TARDIS Systems, P.O.Box 125 I, Los Alamos, NM 87544
(505) 662-9401, (505) 662-6780 fax

Listing 1—The MAP software object uses AlarmClock to trigger itself at a periodic rate

```
* Object MAP "Manifold Absolute Pressure" sensor
* Messages:
* Init: constructor
* Query: returns current manifold abs pressure quantized into 0-255
* MAPUpdate: called periodically via AlarmClock to do filtering

cwMAPSampleTime equ 5244      ;10 ms at 1 tick = 1.907 μs

* Reserve memory for MAP data members
  section bRAM

oMAP:
wFilteredMAP equ *-oMAP
  ds.w 1      ;current filtered MAP value

* Define MAP messages
  section bBankOROM
mMAPInit:  clrw >wFilteredMAP,z ;initial filtered MAP value = 0

* Ask AlarmClock to send the MAPUpdate message a bit later
  pshm z,k      ;save the "this" pointer
  tzy          ;Alarm Clock wants pointer to object
  tzkb        ;to wake up (us) in YK:Y
  tbyk
  ldab #page(mMAPUpdate) ;and message
  tbxk        ;to pass
  ldx #>mMAPUpdate ;in XK:IX
  ldab #page(oAC1k) ;set "this" pointer
  tbzk        ;for call
```

(continued)

After sending the message, it finds the next message and checks its due time. If that time has passed, it sends the message and checks the next one. Otherwise, it resets its interrupt for the time of the next message and goes back to sleep.

A AlarmClock also tracks global system time. In the ECM, time is based on the 68HC16 TCNT register, a 16-bit free-running counter which increments once every 1.907 μs. As a 16-bit counter, it wraps back to zero every 125 ms, so it can't time long intervals.

A AlarmClock responds to an interrupt that occurs every time TCNT wraps around and increments another 16-bit counter, which extends the range to over two hours. Using 32-bit time, an object can request a wake-up call many minutes in the future with microsecond accuracy.

What happens after two hours!

Good question. Our race car never runs for more than 15 min. because it only holds one gallon of gas. But, longer-lived applications might need to

ALL ELECTRONICS CORP.

QUALITY PARTS . DISCOUNT PRICES . HUGE SELECTION

EXPERIMENTERS DELIGHT 20 MB TAPE BACK-UP SYSTEM

Interdyne Model # 6025
Designed to back-up a" IBM PC/XT/AT, DOS 2.0 or later. These units were designed to do back-ups on a small roll of 1/4" magnetic tape. The units are 14 6" x 4.75" x 2.9" and weigh 10 lbs. The heavy-gauge metal chassis box contains a tape drive, power supply with fan and other components. Also includes IEC power cord, data cable and 3 rolls of back-up tape. Although these are new units which include instructions and software on 5.25" floppies we do not wish to represent these as useable units.

We are selling them "AS-IS"

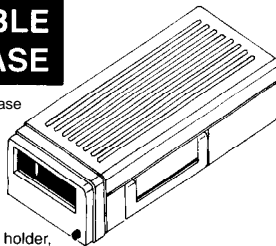
for parts and experimentation.



CAT# TBS-1
\$12.00 each

PORTABLE SCSI CASE

Nearly complete case for SCSI device. Includes 40 watt power supply (5 Vdc @ 3 amps, 12 Vdc @ 2 amps) with cooling fan, power switch, fuse holder, LED socket, power input receptacle, two standard 50 pin SCSI Interface lacks, molex-type power connectors, interior cable with 50 pin socket connector and interior mounting rails. Front face plate is an aftermarket piece that doesn't quite go with the box but serves the purpose pretty well. Absent from the case is the switch in back which specifies the device's position in the chain. This can be easily bypassed with a few 0.1" shorting jumpers (our CAT# SJ-1). Also not included is a standard IEC power cord (our CAT# LCAC-60). Off-white case is 14.25" X 7.38" X 3.26" and has a folding handle for easy carrying.



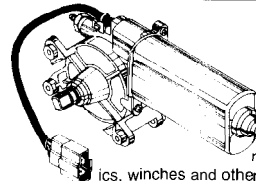
\$19.95 each

2 for \$36.00

CAT# SCSI-1

Visit Our World Wide Web Site...
<http://www.allcorp.com/allcorp/>

12 Vdc GEAR MOTOR



ASMO brand
12 Vdc. reversible gear-head motor. Right-angle motor for automotive applications, power windows, sun-roofs etc. Great for robots,

lifts, winches and other hi-torque applications
175 RPM @ 12 Vdc, 1.5 amps (no load). Works well from 6 to 24 Vdc. 10mm diameter X 15mm long dual flatted threaded shaft (thread pitch 1mm) Overall dimensions: 6.5" long X 2.75" X 3.5" 7" wire leads with molex-type connector

CAT# DCM-61

917%

FORCE SENSING RESISTANCE PAD



Interlink 7.5" X 11.5" x 0.03" thick plastic pad with two separate force sensing resistor circuits embedded in the pad. We have no factory specs or information on the intended application for the product. It was probably designed for some sort of touch sensitive apparatus. The resistance circuits read between 1.5 Meg and 0.5 Meg when nothing is touching the pad. The resistance decreases to as low as 1.5K ohms when pressure is applied to the pad and appears to be dependent upon force and the amount of surface area acted upon. An interesting device for experimentation.

\$4.50 each

CAT# FSR-1

CALL, WRITE, FAX or E-MAIL for a FREE 64 Page CATALOG Outside the U.S.A. send \$2.00 postage.

MAIL ORDERS TO:
ALL ELECTRONICS CORPORATION
P.O. Box 567
Van Nuys, CA 91408
FAX (818)781-2653
E-Mail - allcorp@allcorp.com

ORDER TOLL FREE
1-800-826-5432
CHARGE ORDERS to Visa, Mastercard or Discover

TERMS: NO MINIMUM ORDER. Shipping and handling for the 48 continental U.S.A. \$5.00 per order. All orders including AK, HI, PR or Canada must pay full shipping. All orders delivered in CALIFORNIA must include local state sales tax. Quantities Limited. NO COD. Prices subject to change without notice.

extend the timer or at least check the system's operation at the time limit.

AlarmClock also detects stalls. Each time Distributor detects a tooth edge, it sends a message to AlarmClock that the engine is turning. If AlarmClock receives no such messages between two successive TCNT overflows, it assumes the engine has stalled. Objects register special stall routines with AlarmClock that are executed when this happens. Distributor uses this feature to set RPM to zero and resets engine position to unknown.

Aside from the CPS, the only sensor in the ECM is the Manifold Absolute Pressure (MAP) sensor. Along with RPM, it measures intake manifold air pressure, which indicates how much air flows into the engine. The MAP object represents the physical sensor. This object reads the A/D converter attached to the sensor every 10 ms and filters the reading using a simple first-order digital filter. The result is available to any object that needs it.

Filtering the MAP value is desirable because the air vibrates inside the manifold due to pressure pulses from the valves. But, too much filtering reduces system response time, causing poor throttle response. Listing 1 demonstrates the filtering code and use of AlarmClock for wake-up calls.

TESTING

The ECM was tested carefully and rigorously. My most important rule was to verify with an oscilloscope that everything else was operating properly, before I connected the fuel pump. I made sure that the ASD subsystem worked, the code did not hang, the coils fired at the right time, and the injectors turned off when they should.

I did this to convince myself that the engine would not be damaged, nor gasoline spill all over the place. Just in case, I also installed a kill switch affecting all electronics, wore safety goggles, and had another person handy with a big fire extinguisher.

I first checked the code to make sure it worked as expected. Single-stepping through every code path took a long time but was well worth it.

Listing 1-continued

```

ldz #>oAClk ;to the AlarmClock
ldd #cwMAPSampleTime ;load snooze time
clr
jsr mAClkWakeMeAt ;request wake-up call
pulm z,k ;restore the "this" pointer
rts
mMAPQuery:
ldd >wFilteredMAP,z
rts

mMAPUpdate:
* To filter MAP, apply the formula:
* NewFilteredMAP = (OldFilteredMAP * 0.75) + (CurrentMAP * 0.25)
ldd >wFilteredMAP,z ;grab OldFilteredMAP
lsrd
;E = OldFilteredMAP * 0.5
lsrd ;D = OldFilteredMAP * 0.25
ade ;E = OldFilteredMAP * 0.75
ldd >wMAPADC ;grab CurrentMAP from ADC
lsrd ;multiply
lsrd ;by 0.25
ade ;E = (Old * 0.75 + Current * 0.25)
ste >wFilteredMAP,z ;save as new FilteredMAP
* Ask AlarmClock to send the MAPUpdate message again
pshm z,k ;save the "this" pointer
tzy ;Alarm Clock wants pointer to object
tzkb ;to wake up (us) in YK:IY
tbyk
ldab #page(mMAPUpdate) ;and message
tbxk ;to pass
ldx #>mMAPUpdate ;in XK:IX
ldab #page(oAClk) ;set "this" pointer
tbzk ;for call
ldz #>oAClk ;to the AlarmClock
ldd #cwMAPSampleTime ;load snooze time
clr
jsr mAClkWakeMeAt ;request wake-up call
pulm z,k ;restore the "this" pointer
rts

```

Because the code is real-time, if it screws up, it is virtually impossible to backtrack to the error without an expensive logic analyzer.

I then put the engine on a dynamometer to spin it and check the CPS signal. If you don't have a dynamometer, you can do a pretty good job with a starter motor and big battery. Make sure neither gets too hot. Take the spark plugs out so the engine spins freely.

I checked that the CPS was interrupting the ECM reliably and the code was finding TDC. With all fuel and spark hardware disconnected, I then used an oscilloscope to verify that the coil, injector, and ASD output signals worked, and that the system was not hanging. I also checked the current-limiting feature of the ignition-coil driver circuitry.

Next I checked spark timing. I hooked up the coils and spun the engine at a low RPM near idle. Using an automotive timing light, I checked that cylinders 1 and 4 fired as programmed in the spark-advance table. Then, I made sure that cylinders 2 and 3 fired 180° after 1 and 4. I did this at successively higher RPMs to verify that the advance changed according to the table.

I then tested the limits of AlarmClock by having objects request extra wake-up calls. It is vital that AlarmClock work because it sends the InjectorClose() messages. I had already checked all message combinations: one, more than one, past-due, early, and so on. This extra test was a confidence check. It is especially useful if you can't spin the engine fast enough to let messages stack up.

When I felt confident that the injectors would turn off properly, I hooked them up, leaving the fuel pump disconnected. I spun the engine at various RPMs and throttle positions, watching the spark advance with the timing gun and the injector pulse width with the oscilloscope. I also cranked the engine using its distributor sent messages using the proper low-speed firing algorithm.

Comfortable with what I saw, I turned on the fuel. I must confess that things did not go quite as smoothly as this narrative suggests. It was all the more exhilarating when the engine finally roared to life.

Then, I began tuning for maximum power.

TUNING

For racing engines, wide-open-throttle (WOT) operation dominates all other modes. A race car spends roughly 70% of its time at WOT, 20% at closed throttle, and only about 10% at part throttle. So, I spent most of my time tuning for Maximum Brake Torque (MBT) at WOT and a little time making sure the engine holds an idle. I tuned the part-throttle range at the track when I got the chance.

Simply put, MBT at WOT for a given RPM is achieved with a unique spark-advance value and a unique injector pulse width. These values are recorded in a look-up table. Fortunately, I had access to a dynamometer with constant-speed control which maintained a set speed regardless of engine output. I measured MBT spark advance and injector pulse width for WOT operation over engine speeds from 4000 to 11,000 RPM in increments of 1000 RPM.

I had guessed injector pulse width using the ideal gas law, as described in *INK 62*. To tune pulse width, I simply set an RPM in the area I expected to run most of the time, varied the initial guess until I found a pulse width that generated MBT, and recorded it. MAP remained very close to 1 atm at WOT regardless of RPM, so one pulse width could be used for all RPMs.

For spark advance, I started with the manufacturer's original timing

curve from the service manual, but found it too conservative for MBT. At each RPM, I increased the advance until I found the maximum or until the engine began to knock. If it knocked, I recorded an advance value 2° retarded from when it knocked.

I recorded these values in the advance table for Coil 1. As I mentioned in *INK 63*, it is important to know what knock sounds like for a given engine before tuning it. Heavy knock can damage internal parts.

Both spark advance and injector pulse width can be successfully tuned in a vehicle. The ideal gas law and the OEM advance curve get the engine running well enough to power the vehicle. From there, you can use the seat of your pants, timing equipment, or an in-car accelerometer. Of course, you won't be able to hold constant RPM, so just look for the best acceleration.

Also, don't forget to use your eyes. Soot coming out the exhaust is a sign of too much fuel as are soot-covered spark plugs. Be careful of running too little fuel, though, which causes excessively high-combustion temperatures that can damage the engine.

If you have enough testing time, it's good to map advance and pulse width for many more combinations of MAP and RPM. With limited time, though, you can achieve good results by following these steps:

- use the WOT MBT spark advance curve for part-throttle operation at RPMs above idle
- near idle, use the OEM spark curve
- linearly scale the WOT injector pulse width by MAP for part-throttle conditions (*INK 62*).

CONCLUSION

The effort spent on this system paid off. The Rensselaer team won several trophies at local racing events and an award from Delco Electronics at the international Formula SAE competition in 1994.

It was once feared that if computers replaced carburetors, only big engineering firms would build racing engines. Thanks to better electronics and information access, the opposite

has occurred. Racers can now choose off-the-shelf or do-it-yourself products. Magazines cover high-tech engine controls, and a few Internet mailing lists are available for engine-control developers.

There's no time like the present for getting on track! □

Ed Lansinger is a computer and systems engineer who worked on the Cadillac Northstar powertrain control software, cofounded an industrial software company, and does consulting. He has returned to Rensselaer Polytechnic Institute for graduate studies and is forming a team there to build an electric race car. He may be reached at lansie@rpi.edu.

CONTACTS

Society of Automotive Engineers
400 Commonwealth Dr.
Warrendale, PA 15096-0001
(412) 776-4970
Fax: (412) 7765760

Turbo & Hi-Tech Performance
Mag-Tee Productions, Inc.
9952 Hamilton Ave.
Huntington Beach, CA 92646
(714) 962-7795
Fax: (714) 965-2268

Racecar Engineering
Eric Waiter Associates
369 Springfield Ave.
Berkeley Heights, NJ 07922
(908) 665-7811
Fax: (908) 665-7814

There are two Internet mailing lists for those interested in building fuel-injection systems (diy_efi) and on developing a Motorola 68332-based system (efi332). You can subscribe by sending a no-subject message to majordomo@coulomb.eng.ohio-state.edu with "subscribe diy_efi" or "subscribe efi332" in the body of the message.

IRS

410 Very Useful
411 Moderately Useful
412 Not Useful

DEPARTMENTS

54 Firmware Furnace

62 From the Bench

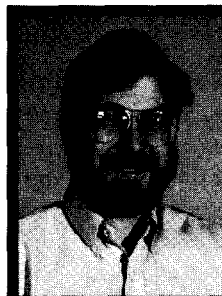
70 Silicon Update

78 ConnectTime

FIRMWARE FURNACE

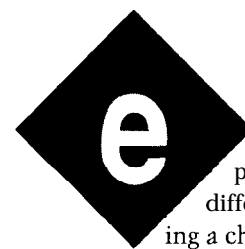
Ed Nisley

Journey to the Protected Land: Real Interrupts in Virtual-86 Mode



For an
'x86
processor
running in

Virtual-86 mode,
processing an external
interrupt is no small
feat. If your DOS comm
program has ever
dropped characters
while running under
Windows or OS/2, you'll
soon understand why.



Embedded programs are different. Try explaining a chunk of your code to a GUI application coder. Talk about culture shock!

Recently, I wrote a magic DOS TSR that linked a 3D digitizer to a virtual-reality program. C++ code didn't have moxie enough for the job. Everything came down to precise bit timings, high-speed I/O, and, yes, a few hundred lines of assembler code.

High-level design and languages are Good Things. Knowing *precisely* what happens when the bits hit the silicon remains essential, though. A friend once observed that it's easier to turn an engineer into a programmer than the converse, perhaps because engineers are more familiar with the real world. Your mileage may vary, but it seems to me more programmers should read *INK*.

This month, we'll see what happens when an external interrupt occurs with the CPU in a Virtual-86 task. If your DOS communications program occasionally drops characters while running under Windows, OS/2, or (shudder) a DOS extender, you'll understand why.

For you embedded systems folks (ab)using a '386 or '486 CPU as a fast 8088, none of this applies—you're not using protected mode. Should you

have the luxury of pure 32-bit PM code and device drivers, you don't need V86 mode. For the rest of us, who want 32-bit performance without rewriting BIOS disk drivers, here's how it works.

If you're looking for light reading, flip on by. This column rings the "most dense writing" bell and ranks up there with C++ folks explaining the latest X3]16 and WG31 inventions. If it were easy, it'd be done now!

MYSTERIOUS GAPS

We've measured interrupt response time under a variety of conditions (INK 50 and 57). The latency, which is the delay from an interrupt signal to the start of its handler, is typically a few tens of microseconds, even when performing a protected-mode task switch.

With that in mind, Photo 1 may come as a surprise. A V86 task created the bottom trace by toggling a parallel port bit. The output of a pulse generator applied to pin 10 of LPT1 appears

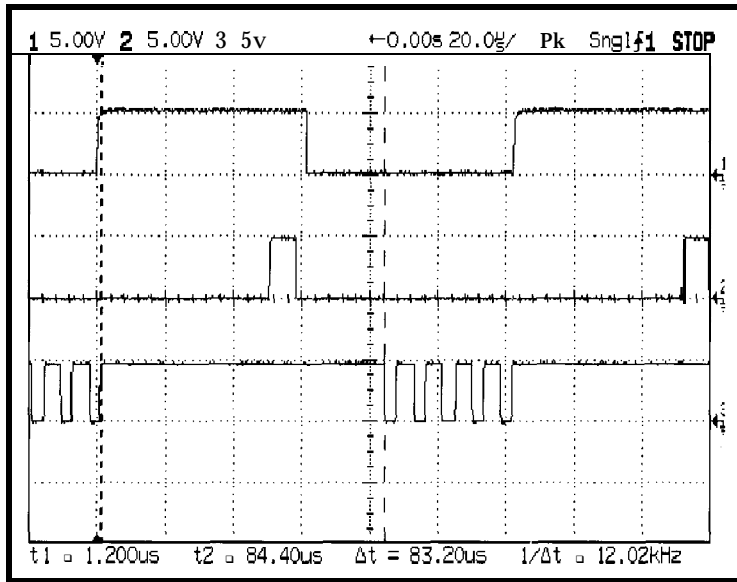


Photo 1—Relaying a hardware interrupt to a V86 task requires more time than you'd expect, even if the V86 task is executing when the interrupt occurs. Trace 1 shows the pulse-generator signal applied to IRQ 7. The V86 interrupt handler produces the blip in Trace 2 more than 50 μ s later. The V86 task produces the pulses in Trace 3 when it's not interrupted. The CPU is an 80-MHz '486DX2, so the response time isn't directly comparable with the results shown previously.

in the top trace. Each rising edge triggers a V86 interrupt handler which, in turn, blips the output port pin shown in the middle trace.

Horrid as it may seem, waking up the V86 interrupt handler requires over 50 μ s. It runs for a mere 8 μ s, then 20 μ s passes before the square wave resumes. These times are even more surprising when you realize that the CPU is an 80-MHz '486DX2, not the

33-MHz '386SX starring in my earlier columns.

Last month, the V86 task required two routines: the 16-bit V86 code itself and a 32-bit protected-mode V86 monitor. The V86 code runs much as it does in real mode and remains blissfully unaware that the monitor is active. The monitor handles GPFs and other protection exceptions produced when the 16-bit code attempts to execute privileged or forbidden instructions.

All that protection makes little sense if an external interrupt can go directly to V86 code.

Thus, in addition to its other duties, the V86 monitor must handle hardware interrupts. Those mysterious gaps in Photo 1 show the V86 monitor in action, mediating the V86 code's access to the outside world. An obvious tradeoff rears its ugly head: don't plan on high-speed interrupts with V86 mode handlers.

FAMILIAR SURROUNDINGS

A V86 task activates an external interrupt in the usual way: install the handler's vector address, tweak the 8259 controller, and enable the I/O hardware's interrupt line. Listing 1, an excerpt from the complete code available on the BBS, should be entirely familiar from your real-mode experiences. The behind-the-scenes tricks (the ones that don't show up in the listing) bear some examination.

Remember that the FFTS task setup code loads the task's I/O permission bitmap with zeros, thus granting unlimited access to all I/O ports. A more defensive system limits access to only a few, carefully chosen ports. The GPF resulting from an attempt to touch a prohibited port would probably terminate the task to prevent interference with the rest of the system.

The V86 monitor can even restrict access to specific I/O bits by setting the I/O permission bit for that port.

Listing 1—Setting up an IRQ 7 interrupt handler within a V86-mode task requires the same steps as in real mode. FFTS places its protected-mode IDT above 1MB, eliminating any conflict with the V86 vectors stored near address zero.

```
XOR  AX,AX                ; aim at int vector
MOV  FS,AX
MOV  SI,4*(8+IRQNUM)     ; . . .using real-mode values!

MOV  AX,OFFSET IntHandler ; insert our handler
MOV  [FS:SI],AX
MOV  AX,SEG IntHandler
MOV  [FS:SI+2],AX

MOV  AX,0040h           ; BIOS data segment
MOV  GS,AX
MOV  DX,[GS:0008h]     ; pick up port base address

ADD  DX,2               ; aim at control port
MOV  AL,14h            ; enable int, raise control pins
OUT  DX,AL

IN   AL,I8259A+1       ; read mask register
AND  AL,NOT INTMASK    ; 0 = enable interrupt
OUT  I8259A+1,AL       ; shazam!
```

When the task attempts to read or write the port, the monitor can examine the offending instruction and either supply the actual I/O port value or make up whatever it likes before resuming the task. We won't get into that level of detail, but it's just a (not quite so) simple matter of software.

As far as the V86 code knows, it reads and writes I/O ports as usual. Because of our lax setup, the instructions execute normally and don't cause any additional overhead. Whew!

The V86 code installs the address of its IRQ 7 interrupt handler in the vector at address 0000:003C. Unlike standard real-mode code, it doesn't need to save the existing interrupt vector because the task never ends and doesn't pass control to the old handler. FFTS boots directly after the BIOS setup, which means the default handler is a simple `I RET` buried in the BIOS. Some BIOSs include a snippet of code that disables the offending interrupt input line, which isn't particularly useful in our situation.

In both real and V86 modes, the 256 interrupt vectors form a table of four-byte entries starting at address 0000:0000. The CPU's memory-paging hardware can relocate each V86 task's addresses to separate locations in physical memory. Because we haven't activated that hardware yet, the code in Listing 1 changes the contents of RAM near physical address zero.

In protected mode, the CPU uses eight-byte interrupt descriptors located in the IDT. The Interrupt Table Register specifies both the starting address and size of the IDT, which can reside anywhere in storage and may have fewer than 256 entries. The FFTS setup code creates an IDT above the 1-MB line and, thus, prevents conflict between the protected-mode IDT and the V86-mode interrupt vectors.

After its simple setup, the V86 code enters an endless loop. Each pass around the loop toggles the parallel-port bit that creates the bottom trace in Photo 1. We'll look at the loop in more detail next month, since it holds the key to a subtle problem.

The PC's interrupt hardware doesn't know about real or protected modes. When a rising edge appears on

Listing 2—This V86-mode interrupt handler uses familiar 16-bit, real-mode techniques. It counts both normal IRQ 7 interrupts and default interrupts that occur when the input signal doesn't meet the 8259's timing specs.

```

UseV86TaskCS
PROC   IntHandler
PUBLIC IntHandler

    PUSH    AX                ; save bystanders
    PUSH    DX
    PUSH    DS

    MOV     DX, SYNC_ADDR     ; send a blip
    IN      AL, DX
    OR      AL, 40h
    OUT     DX, AL

    MOV     AX, SEG IntCounter ; aim seg reg at our data
    MOV     DS, AX
    INC     [IntCounter]      ; count this interrupt

    MOV     AL, 00001011b     ; write OCW3 to read ISR
    OUT     I8259A, AL

    IN      AL, I8259A        ; is it a valid interrupt?
    TEST    AL, INTMASK      ; ISR bit = 1 is normal
    JNZ     @@GoodInt

    INC     [UnExIntCtr]     ; ISR bit = 0 means default int
    JMP     @Done            ; skip EOI command!

@@GoodInt:
    MOV     AL, NS_EOI       ; reset the controller
    OUT     I8259A, AL

@@Done:
    IN      AL, DX           ; remove blip
    AND     AL, NOT 40h
    OUT     DX, AL

    POP     DS                ; restore bystanders
    POP     DX
    POP     AX
    IRET

    ENDP   IntHandler
EndTaskCS

```

the parallel port's -ACK line, the 8259 generates an interrupt request. As far as the V86 code is concerned, the interrupt handler shown in Listing 2 gets control precisely as it would in real mode.

The handler pulses a bit on the parallel port, ticks a counter, and then determines whether the 8259 issued a valid IRQ 7 interrupt or a default interrupt. The latter occurs when the input-pulse timings don't meet the 8259's specifications. A second counter accumulates their occurrence.

As you might expect, the handler sends an `EOI` to the 8259 controller on

each valid interrupt, restores the CPU registers, and executes an `I RET` that returns to the mainline V86 code. Just by looking at the code, you can't tell when or where the V86 monitor gains control. For that, we must examine the CPU documentation and pore over the list of restricted instructions.

Some unfamiliar territory peeks through Photo 1's mysterious gaps!

SWITCHING MODES

The smallest part of the missing time occurs while the CPU switches between V86 and PM operation. With a little help from the V86 monitor

program, this happens automatically at the beginning and end of the handler. As with all protected-mode programming, you must map out the whole process and set up a variety of tables and code before executing the first instruction. Slapdash style just doesn't work any more!

Figure 1 diagrams the CPU's response to an external hardware interrupt during a V86 task. The first step occurs when the CPU acknowledges the interrupt and vectors through the corresponding gate in the IDT. The sample code this month uses the printer port's IRQ 7 for the sake of convenience, although the principle applies to any interrupt.

The CPU automatically extracts the interrupt handler's code segment, offset, and privilege level from the gate's descriptor. It issues a protection exception if the handler is less privileged than the interrupted code.

The restriction is more explicit for V86 mode. Handlers for any interrupts that may occur when the CPU is in V86 mode must run in Ring 0 because the IRET instruction cannot set the VM bit if it's executed at any lower privilege. This means you can get out of, but not into, V86 mode. So, unless you disable interrupts in V86 mode, your handlers are kernel routines.

Because our V86 task, like all V86 tasks, runs with the least privilege, the CPU automatically switches stacks to the Ring-0 SS:ESP defined in the task's TSS. It pushes a variety of information on the stack, including the address of the interrupted instruction. This process is similar to the stack switch caused when the CPU handles a GPF.

Now we get to the heart of the matter. The V86 monitor transfers the instruction's address from the 32-bit stack to the V86 stack, simulating what the CPU would automatically do in real mode. It also extracts the 16-bit interrupt handler's address from the task's vector table and inserts it in the Ring-0 stack. An IRET instruction returns to V86 mode and starts the 16-bit interrupt handler.

Pay attention! There are two interrupt handlers in motion. The PM handler gains control through the IDT using the CPU's hardware. The V86

Q. *How do you know you're getting the most from your development tool purchase?*

A. *Compare Avocet Systems with the competition.*

- A Broad Line of High-Quality Products at Competitive Prices
- Free On-Line Technical Support for Registered Users. No Voicemail!
- Attractive Multi-User Discount Prices & Our "50%+" Educational Discount Plan
- Unconditional 30-Day Money-Back-Guarantee

Now call the obvious choice!

AVOCET
SYSTEMS; I NC.

The Best Source for Quality
Embedded System Tools

(800) 448-8500
(207) 236-9055 Fax (207) 236-6713

ANSI-C COMPILERS
8051 • 68xxx • 68HC11
6805 • Z80/180/64180
6801 • 6809
H8/300 & More

MACRO ASSEMBLERS
8051 • 68xxx • 68HC11 6805
• Z80/180 • Z8 • 8096/196
6800/6801 • 6809
H8/300 • 8048 & More

SIMULATORS/DEBUGGERS
8051 • 680xx • 68HC11 6805
• Z80/180 • 6800
6809 • 8048 • 6502
8085 & More

AND HARDWARE
EPROM Programmers &
Emulators by ETools, Tribal,
Softaid, Cactus Logic

#120

For Borland C/C++, Microsoft C/C++, Borland Pascal

RTKernel

Real-Time Multitasking for DOS

RTKernel is a professional, high-performance, real-time multitasking system for **MS-DOS** and **Embedded Systems**. It can use DOS device drivers and BIOS, and runs other DOS applications as a task - even **Windows!**

RTKernel is **loaded with features**: an unlimited number of tasks, excellent performance, a full set of inter-task communication functions (semaphores, mailboxes, synchronous **message-passing**), real and protected mode support, drivers for up to **38** COM ports and Novell's IPX services, and **lots** more...

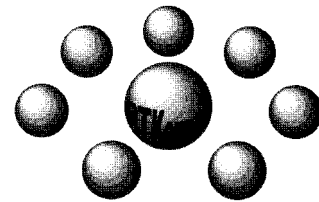
It's **ROMable** and very **compact** (about 16K code, 6K data), making it ideally suited for Embedded Systems.

RTKernel is well-documented and easy to use. All hardware drivers always come with source code: **kernel source code available** at extra charge. **No run-time royalties.**

Join thousands of satisfied customers!

In North America, please contact:
On Time Marketing
88 Christian Avenue Setauket, New York 11733 USA
Phone (516) 689-6654 Fax (516) 689-1172
BBS (516) 689-6285 · FaxFacts (516) 689-6315 · CIS 73313.3177

From other countries, please contact:
On Time Marketing
Karolinenstrasse 32 · 20357 Hamburg · GERMANY
Phone +49-40-437472 · Fax +49-40-435196
CompuServe 100140,633



Use **RTKernel** for:

- process control
- ▶ data acquisition
- ▶ real-time simulations
- ▶ background processing

Libraries: \$495
Source Code: add \$445

On Time
MARKETING
Professional Programming Tools

Free demo disk! Request Info Kit I
Internet: <http://www.on-time.com>
ftp.on-time.com
info@on-time.com

#121

handler gains control after the monitor twiddles its stack.

The next step in Figure 1 occurs when the 16-bit interrupt handler attempts to execute an I RET. In V86 mode, I RET is a privileged instruction that causes an immediate GPF and invokes the V86 monitor using the same mechanism we explored last month. Once again, the CPU switches to 32-bit, Ring-0 code and pushes the V86 state on the stack.

At this point, the V86 mode stack should hold the same return address that the monitor created when it handled the interrupt. The monitor recovers that address, inserts it in the Ring 0 stack, and executes another I RET to return to V86 mode, only this time it's in protected mode. Assuming the V86 code didn't change the return address [it can happen!], the CPU resumes execution at the interrupted instruction as it would in real mode.

As you trace through Figure 1, you can see why interrupt latency can be a big problem in V86-mode code. The V86 monitor gets involved in two places, simulating both the interrupt and the subsequent I RET. Although RISC proponents argue that simple instructions are faster than complex instructions, it remains true that executing a vast number of teeny instructions still takes more time than running a few husky ones.

What's in those mysterious gaps? Looks like PM code to me!

NUMERIC RELATIONS

The interrupt that starts this process could come from any of the usual sources. I used the parallel port with its access for a pulse generator or a push-button switch. This is one of the few cases where contact bounce isn't much of a problem. Even though the code is slow by previous standards, it's still faster than a button.

The LPT1 printer-port interrupt normally drives the IRQ 7 line. I'll leave the pathological case of LPT1 with IRQ 5 as an exercise for you. The system board's primary 8259 interrupt controller handles IRQ 0-IRQ 7, which normally invokes the handlers for Int 08-Int 0F. Unfortunately, those interrupts conflict with the CPU's method

of reporting protected-mode exceptions.

The FFTS start-up code reprograms the 8259 to generate Int 50-57, which are the more-or-less standard interrupts used by other protected-mode systems. IRQ 7 thus produces Int 57 rather than Int 0F. Keeping this straight can be a challenge, but the payoff is worth it.

Pop quiz: if the CPU automatically responds to Int 57, which interrupt does the V86 code see?

Answer: anything is possible when the V86 monitor shuffles the stack!

The structure in Listing 3 defines the relations built into FFTS. The first row represents IRQ 0 and the last is IRQ 15. As in real mode, IRQ 2 cannot occur because the secondary 8259 chains its output into the master 8259 through that signal. Including a row for IRQ 2 simplifies the table access code and costs a mere eight bytes.

The first two columns give the PM and V86 interrupt numbers which correspond to each IRQ line. The PM column uses the same constants that the FFTS set-up routine loads into the 8259 controller. The V86 column dis-

Anatomy of a Great Frame Grabber

Low Price
Forget high priced on-board processing. Our designs exploit the host CPU without sacrificing performance.

Small Board
Half slot ISA, PC/104 and STD boards for embedded and compact designs.

Precision Image
Industry leading sampling jitter less than $\pm 3nS$. Video noise less than 1 LSB.

Easy Interface
Image processing library, C library including source, Windows DLLs for C and Visual Basic.

Call 800-366-9131

Call and find out how ImageNation's quality products, responsive technical support and elegant software get your application quickly to market.

ImageNation™

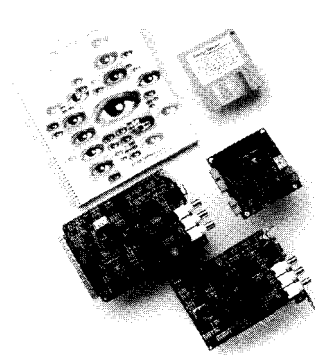
“Vision Requires Imagination”

1-800-366-9131 • Phone: 503-641-7408 • Fax: 503-643-2458

E-mail: info@ImageNation.com

Homepage: <http://www.ImageNation.com/-image>

P.O. Box 276, Beaverton, OR 97075-0276



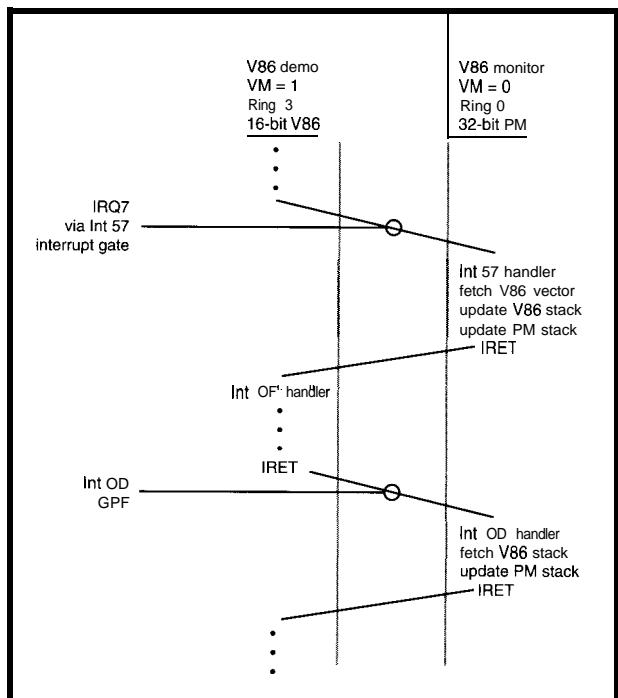


Figure 1-Responding to an interrupt in V86 mode requires firmware coordinated with the CPU's hardware. Here, the process begins when an IRQ 7 from LPT1 triggers Int 57. The V86 monitor's 32-bit interrupt handler simulates an Int 0F and activates the V86 task's 16-bit handler. When the 16-bit IRET triggers a GPF, the V86 monitor adjusts the stacks to simulate a normal IRET and returns control to the interrupted V86 instruction.

each interrupt. Each entry point is a short stub that executes the PUSHA described above, then loads AL with its IRQ number. The FFTS set-up code creates an interrupt gate for each interrupt

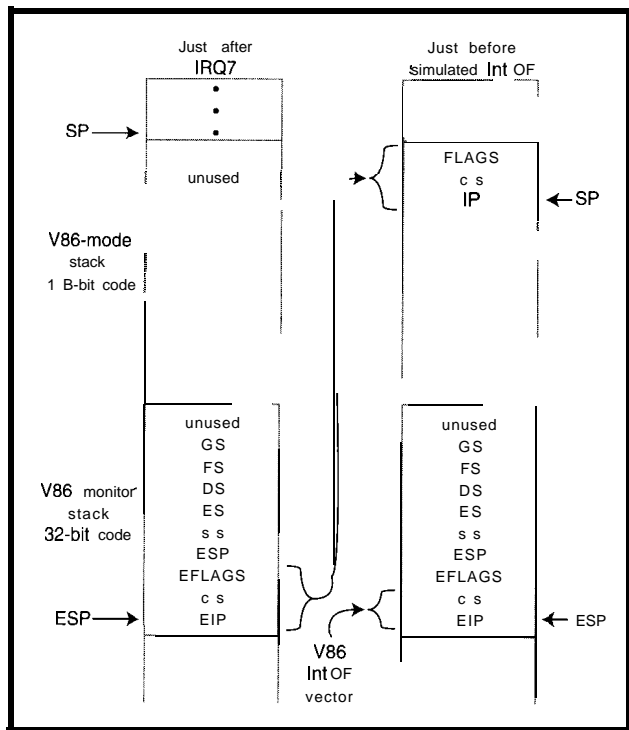
plays the familiar real-mode values. Unlike genuine real mode, there is no conflict with the CPU's reserved interrupts: the V86 monitor creates these.

The third column merely pads each table entry to exactly eight bytes. The '386 CPU's scaled addressing mode simplifies access to tables with entries that are 2, 4, or 8 bytes long. It's a CISC thing.

The fourth column holds the offset of the PM interrupt handler for

using the values in this column.

Obviously, the interrupt number pairs are arbitrary. Each pair represents several promises that you must fulfill while writing the rest of the code. The 8259 interrupt controllers must emit the PM interrupt number, the V86 code must prepare for interrupts using the V86-mode numbers, and the V86 monitor must match them up correctly. That's a lot of "must" with no automated checking.



You need not put this information in a table. Burying it in special-purpose handlers can produce somewhat faster code at the expense of easy maintenance. Once you see what's going on, just tune things to match your needs.

Figure 2-A V86-mode task has two stacks: one for 16-bit code and one for the 32-bit V86 monitor. This diagram shows the stack contents just after an interrupt occurs (i.e., before the V86 monitor activates the 16-bit interrupt handler). The monitor simulates real-mode CPU action by copying the interrupted instruction's address from the Ring-0 stack to the Ring-3 stack.

\$149 4A C Compiler?

You heard right! A quality K&R C compiler designed for the 8051 microcontroller family, just \$149, including the Intel compatible assembler and linker. A source level simulator is also available for just \$169. A great companion to our fine Single Board Computers, like those below. CALL NOW!

552SBC

- 80C552 a '5 1 Compatible Micro
- 40 Bits of Digital I/O
- 8 Channels of 10 Bit A/D
- 3 Serial Ports (RS-232 or 422/485)
- 2 Pulse Width Modulation Outputs
- 6 Capture/Compare Inputs
- 1 Real Time Clock
- 64K bytes Static RAM
- 1+ UVPR0M Socket
- 5 12 bytes of Serial EEPROM
- 1 Watchdog
- 1 Power Fail Interrupt
- 1 On-Board Power Regulation
- 1 Expansion Bus

Priced at just \$229 in 100 piece quantities. Call about our 552SBC C Development Kit, just \$499 qty 1.

Hyperactive '51!

Our popular 8031SBC can now be shipped with Dallas Semi's hyperactive DS80C320, an 8051 on steroids. Averaging 3x faster than the standard 51, your project can really scream! Call or ftp for pricing and brochures today!

Other versions of the 8031 SBC have processors with on-chip capture registers, EEPROM, IIC, A/D and more. Call or ftp for a list!

8031SBC as low as \$49

Call for your custom product needs. Quick Response.



Since 1983

(619) 566-1892



Internet e-mail: info@hte.com
Internet ftp: ftp.hte.com

Listing 3—FFTS maps external hardware interrupts into V86-mode interrupts using this array. Each row corresponds to an IRQ signal. The PM interrupt handler extracts the V86 interrupt number to locate the V86-mode vector in real-mode storage.

STRUC	HWGATEMAP		
PMIntNum	DB	?	; Int number in PM mode
V86IntNum	DB	?	; Int number in V86 mode
	DW	?	; pad to dword boundary
HandlerAddr	DD	?	; offset of handler stub
ENDS	HWGATEMAP		
LABEL	HWIntGates DWORD		
HWGATEMAP	<I8259A_VECTOR_PM+0,08h,0,OFFSET V86IRQ0Stub>		
HWGATEMAP	<I8259A_VECTOR_PM+1,09h,0,OFFSET V86IRQ01Stub>		
HWGATEMAP	<I8259A_VECTOR_PM+2,0Ah,0,OFFSET V86IRQ02Stub>		
HWGATEMAP	<I8259A_VECTOR_PM+3,0Bh,0,OFFSET V86IRQ03Stub>		
HWGATEMAP	<I8259A_VECTOR_PM+4,0Ch,0,OFFSET V86IRQ04Stub>		
HWGATEMAP	<I8259A_VECTOR_PM+5,0Dh,0,OFFSET V86IRQ05Stub>		
HWGATEMAP	<I8259A_VECTOR_PM+6,0Eh,0,OFFSET V86IRQ06Stub>		
HWGATEMAP	<I8259A_VECTOR_PM+7,0Fh,0,OFFSET V86IRQ07Stub>		
HWGATEMAP	<I8259B_VECTOR_PM+0,70h,0,OFFSET V86IRQ08Stub>		
HWGATEMAP	<I8259B_VECTOR_PM+1,71h,0,OFFSET V86IRQ09Stub>		
HWGATEMAP	<I8259B_VECTOR_PM+2,72h,0,OFFSET V86IRQ10Stub>		
HWGATEMAP	<I8259B_VECTOR_PM+3,73h,0,OFFSET V86IRQ11Stub>		
HWGATEMAP	<I8259B_VECTOR_PM+4,74h,0,OFFSET V86IRQ12Stub>		
HWGATEMAP	<I8259B_VECTOR_PM+5,75h,0,OFFSET V86IRQ13Stub>		
HWGATEMAP	<I8259B_VECTOR_PM+6,76h,0,OFFSET V86IRQ14Stub>		
HWGATEMAP	<I8259B_VECTOR_PM+7,77h,0,OFFSET V86IRQ15Stub>		

SAVING THE STATE

The CPU stores the information required to resume the interrupted instruction on a stack, regardless of whether the CPU was in real or protected mode. In V86 mode, however, the state information winds up on a Ring-0 stack that is inaccessible to Ring-3 code. The V86 monitor, running in Ring 0, adjusts both stacks before activating the 16-bit interrupt handler and readjusts them on return.

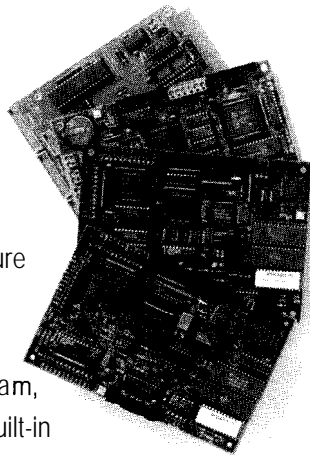
The left-hand diagrams in Figure 2 show the two stacks after the IRQ 7 occurs (just before the CPU enters the V86 monitor program). The right-hand diagrams show the adjusted layouts.

V86-mode tasks generate addresses using the familiar seg:off method, with segment registers holding the high-order 16 bits of the 20-bit address. The address bit patterns in the segment registers are generally not valid protected-mode selectors. As you should know by now, a protection exception occurs very quickly should the CPU enter protected mode with bad segment registers.

Once you have a CISC CPU, though, a little more complexity is no big deal. After the CPU acknowledges the external interrupt and switches stacks, it pushes the V86-mode contents of DS, ES, FS, and GS onto the Ring-0 stack and fills the registers with zeros. An all-zero selector corresponds to the null descriptor entry in

JAM
Packed

Embedded Controllers



- ◆ A/D inputs, 12-bit accuracy
- ◆ Analog outputs
- ◆ Relay control
- ◆ Counter/Quadrature encoder inputs
- ◆ Buffered RS-232/485 serial ports
- ◆ Operator interface via keypad and LCD display
- ◆ Program using a PC
- ◆ 512K program, 512K data memory
- ◆ 5V only operation
- ◆ Built-in BASIC supports all on-card hardware
- ◆ Floating point math
- ◆ From \$195 in 1's

REMOTE™
PROCESSING
The embedded control company

Call for more information and **FREE** catalog of embedded controllers!

Ph: 303-690-1588, Fax: 303-690-1875

CPL	Current Privilege Level
DPL	Descriptor Privilege Level
EOI	End Of Interrupt (command)
FDB	Firmware Development Board
FFTS	Firmware Furnace Task Switcher
GDT	Global Descriptor Table
GDTR	GDT Register
GPF	General Protection Fault
IBF	Input Buffer Full
IDT	Interrupt Descriptor Table
IDTR	Interrupt Descriptor Table Register
IF	Interrupt Flag
IOPL	I/O Privilege Level
LDT	Local Descriptor Table
LDTR	LDT Register
NT	Nested Task
OBF	Output Buffer Full
P bit	Present bit (in a PM descriptor)
RF	Resume Flag
RPL	Requestor Privilege Level
TF	Trap Flag
TR	Task Register
TSS	Task State Segment
VM	Virtual Machine (in EFLAGS)

GDT[O], which means the zeroed registers won't trigger any errors.

The CPU next pushes the V86 SS:ESP onto the stack. Obviously, there is something CISCy going on here because the CPU pushes these values onto the Ring-O stack using a selector before entering the protected-mode handler. These are complicated instructions, indeed. As you become more familiar with protected-mode programming, you begin to appreciate what CISC really means.

The next three stack entries are the EFLAGS and CS:EIP values. The VM bit in the EFLAGS stack entry is 1 because the CPU was in V86 mode when the interrupt occurred. The CPU clears VM in the EFLAGS register before starting the interrupt handler, which is why the handler is a 32-bit PM routine instead of 16-bit V86 code.

The CPU stores a 32-bit value for EIP. The high-order 16 bits of the 32-bit EIP register almost always is zero in a V86-mode task. The exception occurs when the program executes code in the 64 KB just beyond the 1-

MB line. I ignore that possibility—our code wouldn't *think* of doing such a thing. If you must run real DOS programs in a V86 box, this is just one of the headaches you'll encounter.

With that out of the way, the CPU fetches and executes the ISR's first instruction. Next month you'll see the FFTS handler begins with a PUSH A that saves the remaining CPU registers on the stack, producing the structure in Listing 3. Since the stack grows downward in memory, 01 dGS is the first value pushed on the stack and 0 1 d ED I is the last. Each segment register value has 16 high-order zeros to keep ESP aligned on four-byte addresses.

All the stack shuffling before the PUSH A happens *automagically* in the first few microseconds after the port's IRQ line goes active. What follows soaks up the remainder of that mysterious 50- μ s gap: faking a real-mode hardware interrupt.

of the video buffer. The first character changes after each 256 V86 task switches, the second increments on valid IRQ 7 interrupts, and the third tallies default interrupts caused by invalid timing. A signal applied to LPT1's -ACK (pin 10) triggers the interrupts.

You can drive pin 10 with a push button or signal generator. If you crank the frequency high enough, you'll see some default IRQ 7 interrupts.

Next month, we'll fill in those gaps with straightforward code. \square

Ed Nisley (KE4ZNU), as Nisley Micro Engineering, makes small computers do amazing things. He's also a member of Circuit Cellar INK's engineering staff. You may reach him at ed.nisley@circellar.com or 74065.1363@compuserve.com.

RELEASE NOTES

The code this month pokes three characters into the bottom-right corner

IRS

- 413 Very Useful
- 414 Moderately Useful
- 415 Not Useful

This Parts List Software is for Engineers / Designers

Not MRP: P&V actually assists development!

Windows™ program helps you stay organized



Keep track of:

- Drawings & Specs
- Suppliers & quotes
- Product material costs
- Engineering stock

Parts & Vendors™

...for independent and departmental projects. Use alone or in a workgroup.

\$99 + shpg Call **800.280.5176**

Requires 486, 8meg min. RAM, Windows™

Trilogy DESIGN voice 916.273.1985 fax 916.477.9106 P.O. Box 2270, Grass Valley, CA 95945

BLAZING SPEED AND SMART TOO!

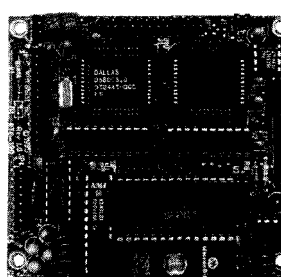
RTC320

One of Micromint's hottest-selling products for the past five years has been the RTC31/52 stackable controller. It has been a leading price/performance choice among our customers. With our new RTC320 board, we have expanded the value of that relationship even more.

Occupying the same small 3.5"x3.5" RTC footprint and using S-V-only power, the RTC320 uses the new Dallas Semiconductor 80C320, which is 8031 code compatible and 3-5 times faster. At 33 MHz, the RTC320 is an 8-MIPS controller! Along with the new powerful processor, the RTC320 board accommodates up to 192 KB of memory, two serial ports (RS-232 and RS-485), 24 bits of TTL parallel I/O, and a 2-channel, 12-bit ADC. The RTC320 puts some real firepower under the abundant variety of RTC I/O expansion boards. Plugging in your favorite ICE or EPROM emulator is the easiest way to develop code. For the diehards who like to twiddle the bits directly, we have a ROM monitor specifically designed for the Dallas '320.

RTC320-1 (22 MHz)
\$239 \$179/Qty.100

(Call for pricing on 33 MHz)



CALL 1-800-635-3355 TO ORDER

MICROMINT, INC.

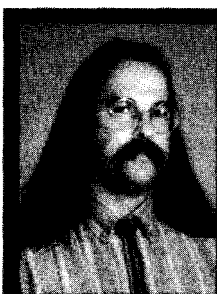
4 PARK STREET • VERNON, CT 06066 • (860) 871-6170 • FAX (860) 872-2204

#119

FROM THE BENCH

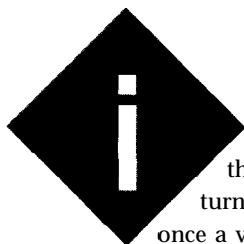
Jeff Bachiochi

Carrier Current Modem Part 1: Communicating at 1200 bps Around the House



Have you
wanted to
send data
through

your house without
running more wires?
SGS-Thomson's
ST7537 power-line
modem chip may be just
the thing for you. Jeff
shows how to use the
chip in a simple circuit.



If you count only the PCs that are turned on at least once a week at my house, you get five. I own the slowest, an 8088 portable which still gets used for writing this monthly column. Ryan, now a senior in high school, has the fastest machine. (He swaps motherboards the way I used to trade baseball cards. No telling what speed he's currently running!)

The machines have two things in common, though. They all have their own assortment of games (ugh) and a word processor. Although PCs abound, printers do not, and we all need to print. If we were tied together, print files could easily be moved to the print station without stringing more wires around. Wouldn't it be nice if we could use wiring already in place throughout the house? How about the power line?

Although this month's project starts out to solve this problem, it morphs into a different animal.

POWER-LINE STANDARDS

Ken has kept us up-to-date on the CEBus committee's work toward national standards for home automation. I think it's interesting to note that the old and outdated X-10 "standard" still used by many today remains a hurdle in gaining a new standard.

Those who have X-10 modules installed in their home know that what sounds like a great idea isn't perfect. X-10 is inherently a one-way system. Attempting to jump from one phase to another or noise on the line can sometimes keep transmissions from getting through. Without feedback, you have no way of knowing that the command was lost.

It's not that I hate X-10. I don't. In fact, if not for it, we may not have progressed to this stage of home automation. Although X-10 has put the facilities in their standard to allow bidirectional communication and data passing, very few devices currently support it.

A big break came for many manufacturers with the PL5 13 line interface which lets us safely interface our computers to the line and send X-10 commands. PL5 13 enabled third-party equipment to use X-10 modules.

When the TW523 module was released, we thought two-way communication was finally here. But, the command for sending extended data is not recognized by even the TW523, so using the power line as a data-passing medium was still not at hand.

X-10 PROTOCOL

First, let's have a quick overview of the X-10 protocol.

By sending three 1-ms bursts of a 120-kHz tone, a binary 1 bit is indicated. The bursts occur at the 0, 60, and 120° points of the 60-Hz wave. The transmission is followed by an absence of bursts in the second half cycle. Binary 0 bits send no bursts for the first half cycle and 1-ms bursts for the second half cycle at the 180, 240, and 300° points.

The preamble doesn't use quite the same format, so it isn't confused as data. However, for an overview, just remember data is transmitted in 1-ms bursts of 120-kHz carrier (or lack of).

CEBUS'S PLBUS

In *INK 15*, Ken's "CEBus Update" touched on most of the media proposed in that standard. Special attention was given to power-line proposals. The data protocol then looked an awful lot like the X-10 protocol.

hung low. It must be returned high for 2 μ s before reenabling the transmitter.

The modulated signal then passes through a switched-capacitor bandpass filter to remove much of the harmonic content. This step is important to the internal processing of the ST7537 since the final spectral output must pass FCC limits. Second-harmonic distortion is reduced by 50 dBm and third-harmonic distortion by 60 dBm.

Finally, an output amplifier drives external push-pull transistors from the AT0 [analog-transmit output]. The final output stage is fed back into the ST7537 PAFB (power-amplifier feedback) input to keep the output under control. The push-pull's output drives a winding of the tank circuit. The tank circuit is part of an isolation transformer which couples the carrier to the power line.

Bias to the push-pull transistors is controlled by the ST7537's comple-

mentary outputs PABC and \bullet PABC (power-amplifier bias control). When the ST7537 exits the transmit mode, these outputs remove the bias from the push-pull transistors, leaving them in a high-impedance state so they don't load down the tank.

FOLLOW THE RECEIVE PATH

The ST7537 spends most of its time in receive mode, looking for carriers to detect. The output transistors don't impose any significant load on the tank circuit. Therefore, the ST7537 can listen for a carrier coupled through the isolation transformer from the line. These signals enter the ST7537 through the RxAI (receive-analog input). They pass through a switched-capacitor bandpass filter (similar to the one in the transmit section) and a 20-dB amplification stage.

A local oscillator driven by the crystal then converts the amplified

carrier to 5.4 kHz. A second switched-capacitor bandpass filter (this time centered on 5.4 kHz) improves the signal-to-noise ratio, and the carrier is demodulated.

Provided the *CD (carrier-detect) output is low, data is available at the RxD output. While *CD is high, the RxD output is held high (idle). *CD proclaims the carrier status about 6 ms after the carrier has actually been detected or lost. This delay allows short carrier absences and noise to be overlooked.

Similar to a telco modem, the power-line modem connects to an available RS-232 serial port on your favorite PC (refer to Photo 1 and Figure 2). Four connections are made: TXD, RXD, DCD, and RTS. RTS is used by the PC to place the ST7537 in Rx/*Tx mode, which determines the direction of the data.

In converting TTL to RS-232, you can use a MAX232 with the ST7537. If the MAX232 is powered by an isolated source, it provides sufficient protection for the PC from the line. Unisolated power supplies referenced to one side of the line can damage your PC. The computer's interface must be isolated via optocouplers to prevent damage from occurring. Figure 3 shows how optocouplers can be used to establish an isolation barrier between the ST7537 and your PC.

It is necessary to use a communications program which lowers the RTS line when transmission begins and which raises it to disable the transmitter when listening. If yours doesn't do this, take a look at Listing 1. Here you can see how you might communicate between systems using this project's hardware. (The software also applies to those of you who wish to interface your PC to an RS-485 network. The RTS signal can control the DE pin on 75176 RS-485 bus drivers.)

Connected systems act as slaves, monitoring for messages until a key is pressed. A key press lets the program know you wish to send a message to another node. The program prompts for a message and node address and checks to see if the bus is in use by testing the carrier detect from the ST7537. If it's free, raising the RTS

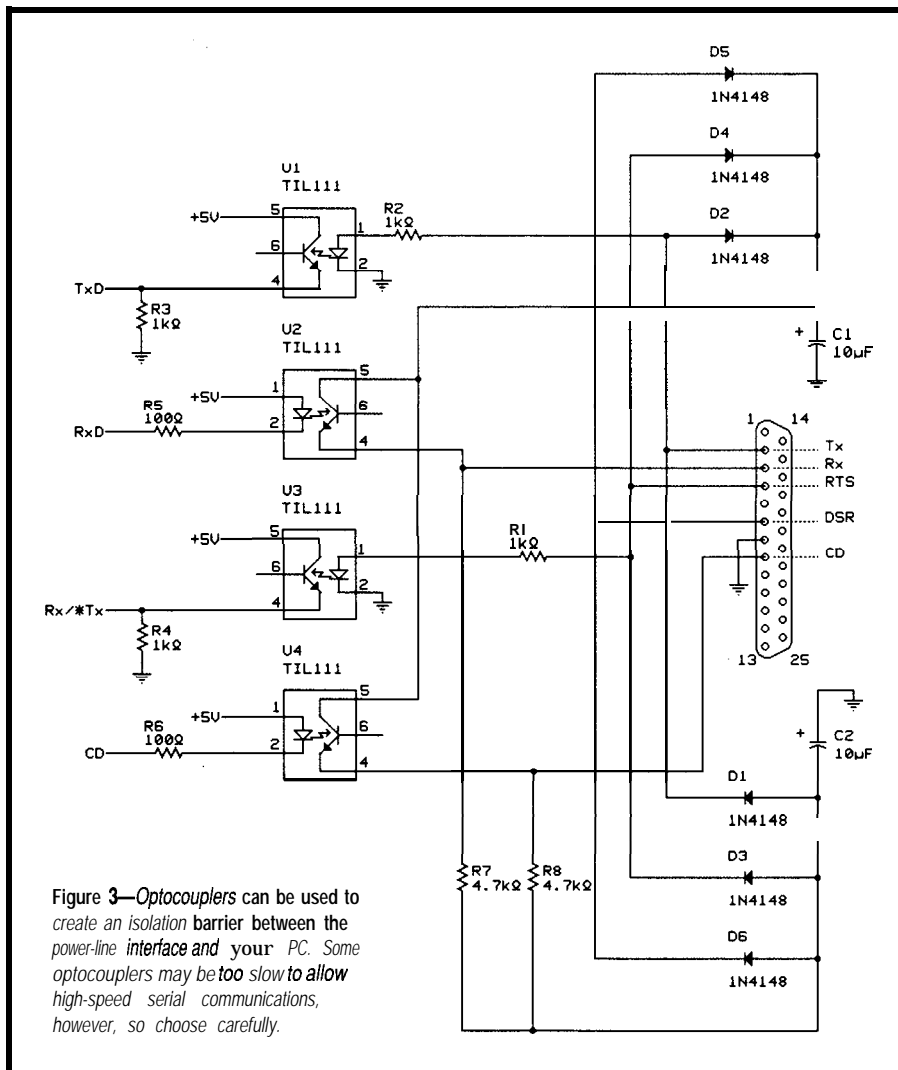


Figure 3—Optocouplers can be used to create an isolation barrier between the power-line interface and your PC. Some optocouplers may be too slow to allow high-speed serial communications, however, so choose carefully.

Listing 1--This BASIC demonstrates the framework for a multimaster system of communicating PCs.

```
10 SCREEN 0,0: WIDTH 80
20 KEY OFF: CLS: CLOSE
30 DEFINT A-Z
40 OUT BASE+4, INP(BASE+4 AND &HFD)
50 INPUT"Enter the address of this node (0-127)";HERE
60 IF (HERE<0 OR HERE>127) THEN GOTO 40
70 INPUT"Enter a 1=COMM1, 2=COMM2";BASE
80 IF (BASE<1 OR BASE>2) THEN GOTO 70
90 IF BASE = 1 GOTO 120
100 OPEN"COM2:1200,N,8,1,CS0,DS0" AS #1: BASE=&H3F8
110 GOTO 130
120 OPEN"COM1:1200,N,8,1,CS0,DS0" AS #1: BASE=&H2F8
130 OPEN"scrn:"FOR OUTPUT AS 2
140 LOCATE , ,1
150 ON ERROR GOTO 490
160 GOTO 250
170 INPUT"Type in a string to send";B$
180 INPUT"Enter the address to send it to (0-127)";THERE
190 IF (THERE<0 OR THERE>127) THEN GOTO 180
200 B$ = CHR$(&HFF) + CHR$(&HAA) + CHR$(THERE) + CHR$(HERE) + B$
210 IF (INP(BASE+6) AND &H80) = 0 THEN PRINT"Carrier detected
    waiting": GOTO 210

220 PRINT"No carrier--OK to transmit"
230 OUT BASE + 4, INP(BASE+4 OR 2)
240 PRINT #1, B$
250 OUT BASE + 4, INP(BASE+4 AND &HFD)
260 X = 0
270 N = LOC(1)
```

(continued)

line enables the ST7537's transmitter, and the message is sent. Lowering the RTS line disables the ST7537's transmitter (carrier), and the node returns to slave mode.

If a message is received while in slave mode, the preamble and destination bytes are checked. If the destination matches this node's address, the message is received, and an acknowledge message is returned to the sender. If the destination matches this node's address with the high bit set, this is an acknowledgment of a previously sent message. If the destination doesn't match the node's address, the message is discarded.

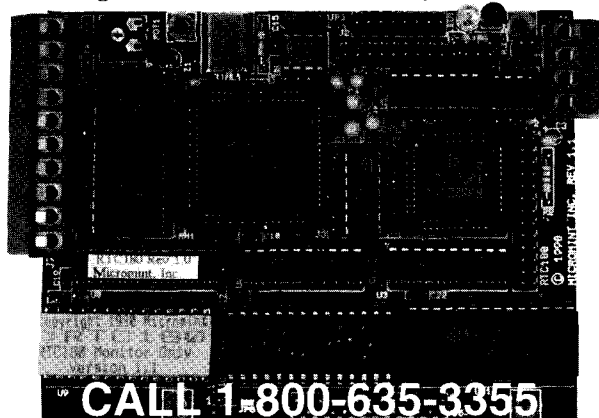
The program does not include any packet protection such as a checksum. This kind of error trapping is left up to the user.

NEXT MONTH

Sending messages between computers over the power line probably does not seem like a big deal. But, stay tuned. I'll put some smarts in the node and try to close the loop on X-10. ☐

**STOP
LOOK
LISTEN**

Odds are that some time during the day you will stop for a traffic signal, look at a message display or listen to a recorded announcement controlled by a Micromint RTC180. We've shipped thousands of RTC180s to OEMs. Check out why they chose the RTC180 by calling us for a data sheet and price list now.



CALL 1-800-635-3355

MICROMINT, INC.

4 Park Street, Vernon, CT 06066
(203) 871-6170 • Fax (203) 872-2204



in Europe: (44) 0285-658122 • in Canada: (514) 36-9426 • in Australia: (3) 467-7194 • Distributor Inquiries Welcome

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on Circuit Cellar INK's engineering staff. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circellar.com.

REFERENCE

[1] Joel Huloux and Laurent Hanus, "ST7537: Power Line Modem Application," *Power Line Modems and Applications*, SGS-Thomson Microelectronics, Application Note AN655/0394, 1994.

CONTACT

SGS-Thomson
55 Old Bedford Rd.
Lincoln, MA 01773
(617) 259-0300

IRS

416 Very Useful
417 Moderately Useful
418 Not Useful

Listing 1-continued

```

280 N = NN: NN = LOC(1): IF INKEY$<>" THEN PRINT "Operator
      interrupt": GOTO 170

290 IF N = NN THEN X = X + 1 ELSE X = 0
300 IF (X>255 AND LOC(1) = 0) THEN PRINT"waiting...":X=0: GOTO 280
310 IF X<256 THEN GOTO 280
320 A$ = INPUT$(LOC(1),#1)
330 FOR X = 1 TO LEN(A$)
340 IF MID$(A$,X,1) = CHR$(&HAA) THEN AS = MID$(A$,X+1): GOTO 370
350 NEXT X
360 PRINT"Preamble not found--Canceling": GOTO 260
370 IF (MID$(A$,1,1) = CHR$(A)) THEN PRINT"It's my address": GOTO
      400
380 IF (MID$(A$,1,1) = CHR$(A+&H80)) THEN PRINT"It's an ACK to
      me": GOTO 450

390 PRINT"It's not to me Canceling": GOTO 260
400 PRINT"A message to me has been received"
410 PRINT"It is...":MID$(A$,3)
420 PRINT"1 will send an acknowledge"
430 B$ = CHR$(&HFF) + CHR$(&HAA) + CHR$(THERE+&H80) + CHR$(HERE)
      + MID$(A$,3)

440 GOTO 210
450 PRINT"An acknowledgement to my message has been received"
460 PRINT"It is...":MID$(A$,3)
470 PRINT"Now let's wait for more messages"
480 GOTO 250
490 PRINT"error #":ERR: RESUME

```

The Next Generation in Lighting Control

Introducing

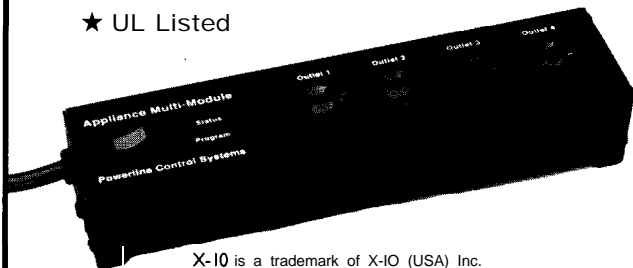
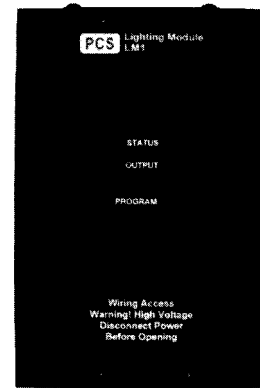
PCS Lighting Modules

Features include:

- ★ Brighten From Off
 - ★ Pre-Set Dim
 - * Soft-Start
- ★ 200 Dim Levels
- ★ Dim Level Retained
 - ★ Advanced Programming
 - ★ UL Listed

The Complete **X-10** Compatible lighting Control Solution is Here!

Powerline Control Systems has combined the reliability of hard-wired systems with the ease of X-10 installation. Our full line of 1000, 1500, and 2000 Watt Lighting Control Modules cover all of your high power lighting needs at a fraction of the cost of other alternatives. These products are available now through all major Home Automation distributors.



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

- ★ Inductive Load Control
- ★ Improved Receiving
- ★ Remote Operation with **Leviton** Slave Switches
- ★ **EEPROM** Memory
- ★ Solid State
- ★ 100% Money Back Guarantee
- ★ Made in the U.S.A.

For more information about PCS's lighting control products call:

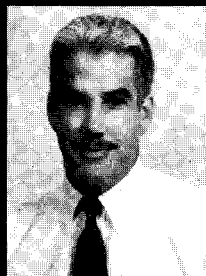
1-818-701-9831

Powerline Control Systems ★ 9031 Rathburn Avenue ★ Northridge, CA 91325
FAX: (818) 701-1506 ★ E-MAIL: PCS@infodial.net

ADSP4ME

SILICON UPDATE

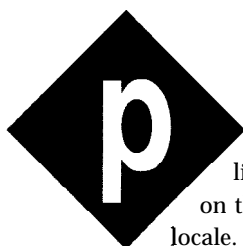
Audio Processor Chips for the Masses



In *INK 63*, Tom showed how the latest

digital video chips take much of the mystery, cost, and headaches out of tough designs. Here, he looks at the AD1847 and ADSP-2181 that handle audio with the same panache.

Tom Cantrell



ersonalized license plates take on the flavor of the locale. Here, in Silicon Valley, if you've got a vexing design problem, keep an eye out for "PAL GURU."

Critical market intelligence can be gathered with an eagle eye. Remember back when conjecture raged about Apple's beyond-68k plans? A cruise through their lot would have revealed "RISC MAC."

My favorite is the regally restored Rolls Royce that commands attention, not to mention about four of those ever-shrinking excuses for parking spaces, wherever it goes. Believe me, it's a big deal when Silicon Valley's

candy man, the "DRAM REP," deigns to visit.

One reason the DRAM rep rides in style is that all the new-fangled multimedia stuff sucks bits-DRAM, CD, hard disk, and CPU-like that Rolls sucks gas. Around here, when folks in the PC food chain hear "multimedia," it sounds like "mucho muy dinero."

Last month, I covered the basics of digital video and how the latest chips take much of the mystery, not to mention cost and design headaches, out of what were traditionally tough designs. Since it's called "multimedia," not "unimedia," now's a good time to look at some no-muss-no-fuss wonder chips that handle the audio side of the multimedia equation with similar panache.

HEAVY METTLE

Analog Devices splits the audio work into analog and digital parts, the AD1847 stereo CODEC and the ADSP-2181 DSP, respectively.

The '1847, responsible for analog I/O, mixing, and code/decode, conveys the digital audio to and from the DSP via a specialized serial bus. Let's start with the input jacks and follow a signal through the '1847 to see (er, hear) each function in action.

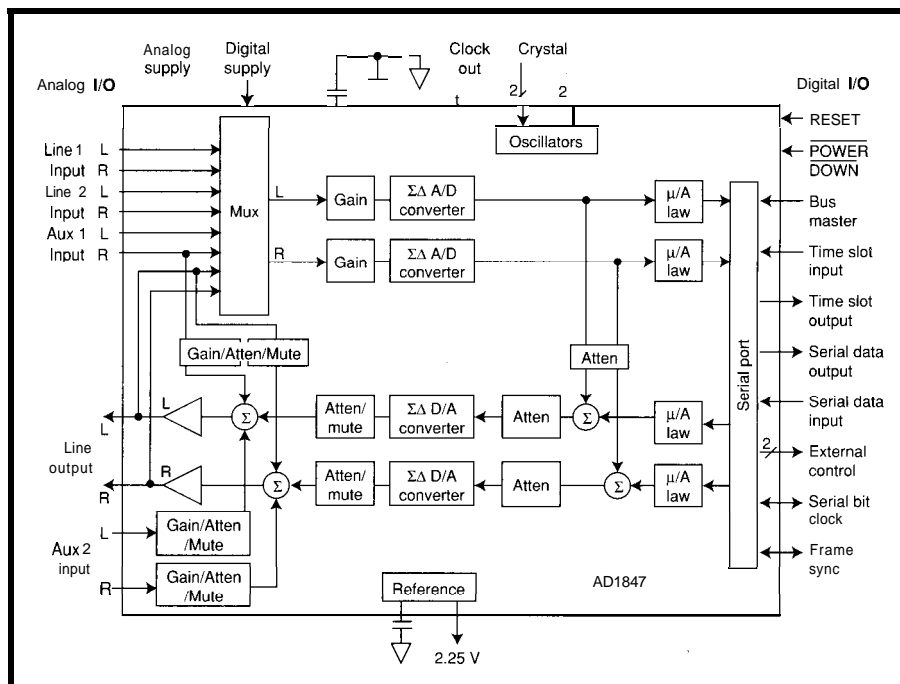


Figure 1—The AD1847 CODEC integrates the entire analog portion of a high-performance audio design onto a single chip. Features include multiple stereo inputs, 16-bit A/D and D/A converters, 5.5–48-kHz sample rate, analog and digital mixing, programmable gain and interpolation, decimation, and low-pass filters.

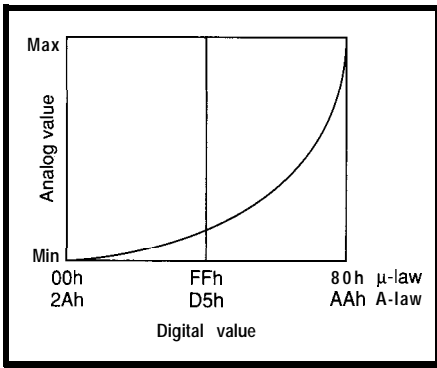


Figure 2-A primary function of a CODEC is to *compand* (compress and expand) the digitized audio. The principle is simple-give away high-magnitude input resolution to increase dynamic range.

Up to four stereo inputs (Line 1, Line 2, Aux 1, and Aux 2) can be connected to the '1847. As shown in Figure 1, the Line and Aux 1 inputs feed a multiplexer for selectable presentation to the A/D converters. Aux 1 can also be mixed in the analog domain (i.e., post-D/A converter) with the output, a role Aux 2 is limited to. Even if the D/A converters aren't outputting anything (i.e., they're muted), the analog mixing function works.

Once past the mux, a pair of amps applies independent 32-level gain for the left and right channels which each proceed to 16-bit (i.e., CD quality) A/D converters.

Forget hassling with any troublesome front-end filter. Thanks to a combination of 64-times oversampling followed by low-pass decimation (to

0.4 times the sampling frequency), the only external filter components are a 1.0- μ F capacitor for each channel. However, to deal with rather loose definitions of line level (2 V p-p?) in the audio world, it is recommended to AC couple the '1847 inputs (1 V p-p) and outputs (0.7 V p-p), calling for a few more Rs and Cs.

Speaking of the all-important sampling frequency, the question is which one? Featuring a pair of crystals (typically 24.576 and 16.9344 MHz) and a programmable divider, the '1847 handles more than a dozen sample rates that cover the range from sub-phone (5.5 kHz) to better than CD (48 kHz).

The output of the A/D converters is 16-bit signed PCM (same as CDs) and offers a whopping 96-dB dynamic range. Note that the 16-bit PCM data is also made available for on-chip mixing, this time in the digital realm.

Onto the "CO" part of "CODEC," where the linear 16-bit PCM is compressed to 8 bits using the telecommunications-inspired A-law (Europe) and u-law (USA, Japan) standards. As you can see in Figure 2, these so-called *companding* (i.e., compress and expand) schemes effectively nonlinearize the A/D conversion by devoting more codes to low-amplitude inputs.

The result is more dynamic range with the same number of bits. For example, simple 8-bit unsigned PCM

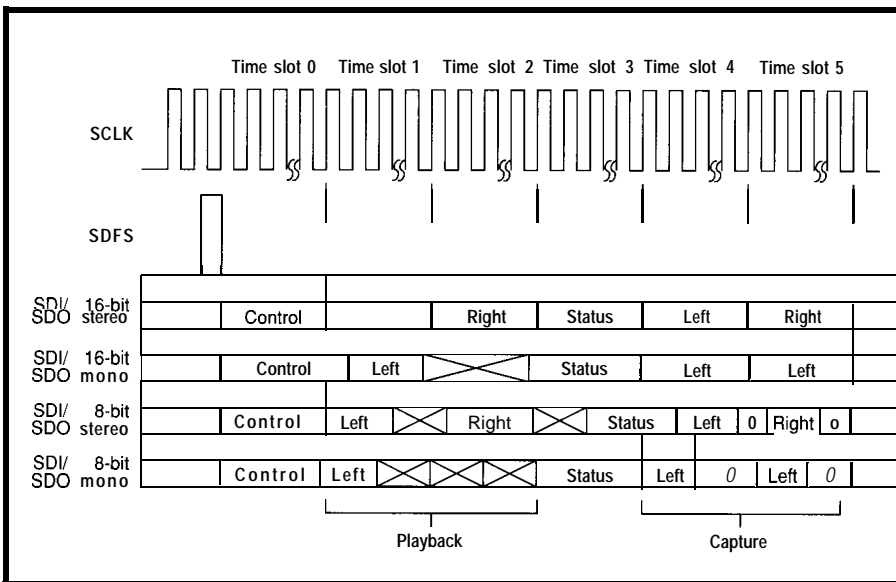


Figure 4-The AD1847 transfers a frame (the SDFS pin signals frame sync) containing six 16-bit (clocked by SCLK MSB first) slots, three in each direction.

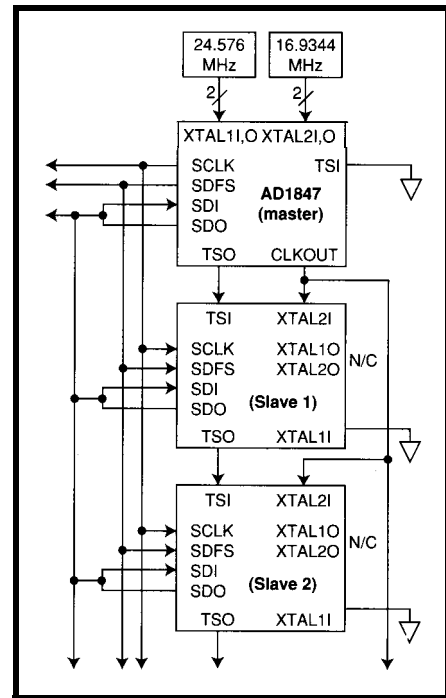


Figure 3-The AD1847 serial port is an on-ramp to an audio highway that supports multiCODEC designs. A single-master CODEC generates clocks for all slaves and the TSI/TSO (Time Slot In/Out) daisy-chain controls access. In this so-called one-wire configuration, the input and output (SDI and SDO) pins are tied together.

(a format the '1847 also supports) has a 48-dB range while A-law and u-law achieve 64 and 72 db, respectively (the equivalent of 13- or 14-bit PCM).

The coded digital data is passed on to the serial port, which is a rather innocuous name for what's actually an elegant time-division-multiplexing (TDM) packet LAN for audio.

Multiple CODECs can be daisy-chained (see Figure 3) with the master (BM pin high) responsible for generating the serial clock (SCLK) and frame sync (SDFS) for the slave(s). The token-passing-like arbitration scheme hands control from the master to each slave via daisy-chained TSI/TSO (Time Slot In/Out) pins.

As you can see in Figure 4, this is a so-called *single-wire* setup, in which the data in and out pins (SDI, SDO) are tied together. This configuration assigns six 16-bit slots-three in each direction-to each CODEC, while a two-wire variant that doubles bandwidth (i.e., three slots) is also supported.

Incoming digital data goes through more or less the reverse process-decoding (the "DEC" part of "CODEC")

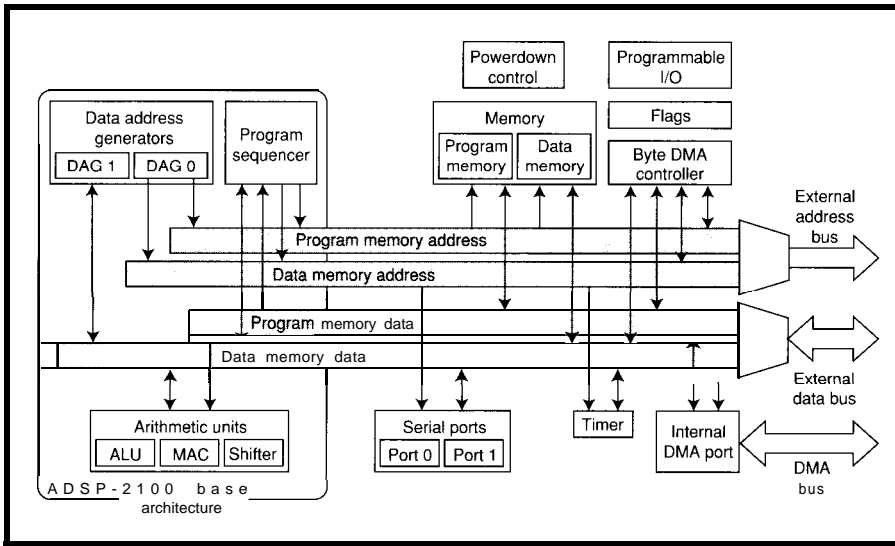


Figure 5—The ADSP-2181 combines a high-speed (33 MIPS) DSP core with CODEC-compatible serial ports (SPORT) and a whopping 80 KB of on-chip RAM.

to 16-bit PCM and D/A conversion with some extra filtering and programmable (64-level) attenuation stages along the way.

Consuming little board space, power (5 V @ 140 mA, less than 1 mA in power-down mode), and bucks (\$12 in low volume), the AD 1847 easily dispatches with all the analog hassles. Now, it's simply a matter of figuring out what to do (and how to do it) with all those sound bytes flying to and fro.

RISC MEETS 8051

When it comes to the never-ending architecture wars, DSPs don't get the attention of their more glamorous desktop RISC and CISC brethren. But, look under the hood, and you'll find that DSPs are just as whizzy in their own purposeful way.

As shown in Figure 5, the ADSP-2181 (\$54 in low volume) is the latest member in the Analog Devices 21xx family, whose members combine a

common DSP core with chip-specific memory and I/O.

The '2181 does exhibit a number of RISCy characteristics including pipelining, high-clock rate (33 MHz), single-cycle execution of most instructions, and copious (80 KB!) on-chip 1-clock memory.

A key difference with desktop CPUs is that the on-chip memory is configured as simple RAM rather than as cache. From Figure 5 it's clear that the '2181 (like most DSPs) adopts a Harvard approach with a separate internal bus and memory for instructions (16 KB x 24) and data (16 KB x 16).

The special-purpose nature of the architecture is certainly reflected in the '2181 programmer's model (see Figure 6). Gone are those boring R0-Rn general-purpose register files. Instead, the registers are split across functional units including the ALU, MAC (Multiply-Accumulator), barrel shifter, and two data-address generators (DAGs).

The ALU, MAC, and barrel shifter each have dedicated input and result registers. For instance, a typical ALU instruction has the form AR = AX op AY, which places the result in AR and sets flags in AF. Similarly, a MAC

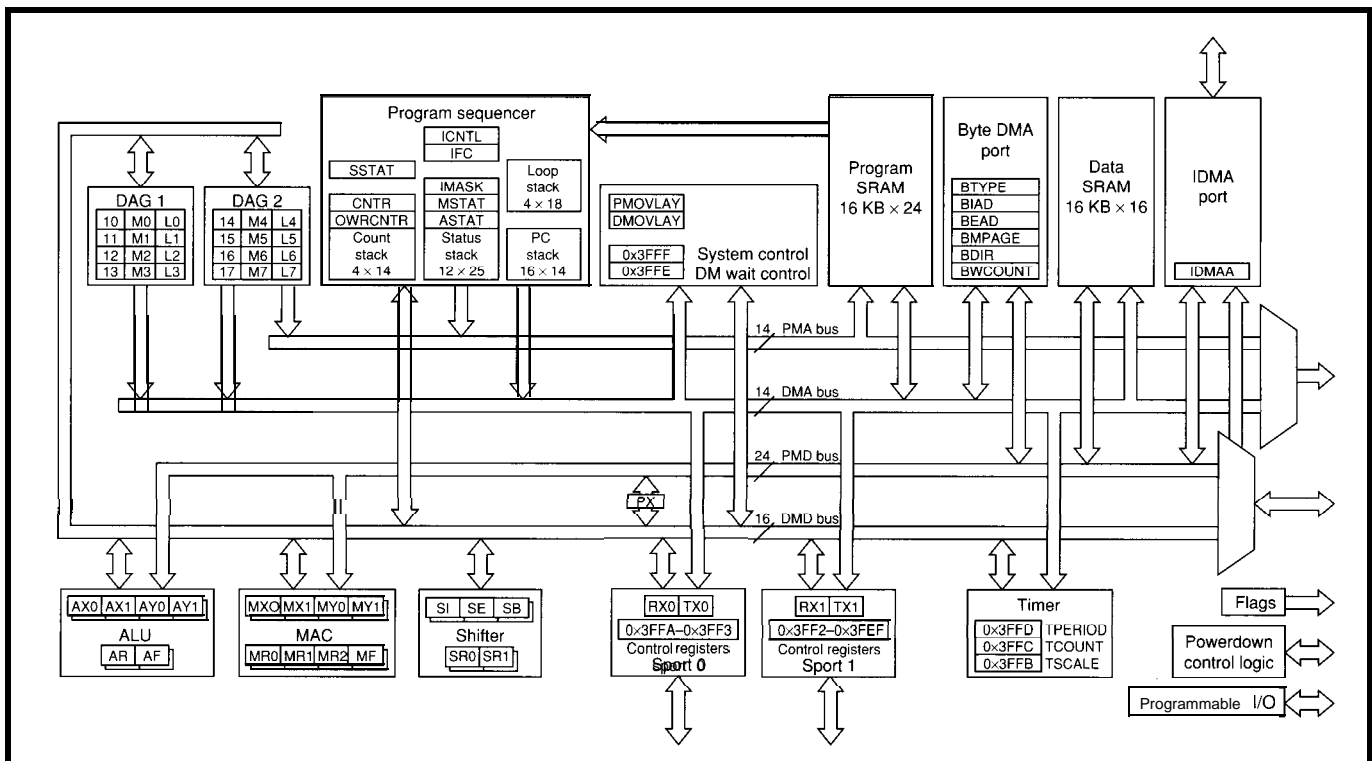


Figure 6—For the ADSP-2181 register set, form follows function and the function is to blast through DSP loops at warp speed. General-purpose registers need not apply.

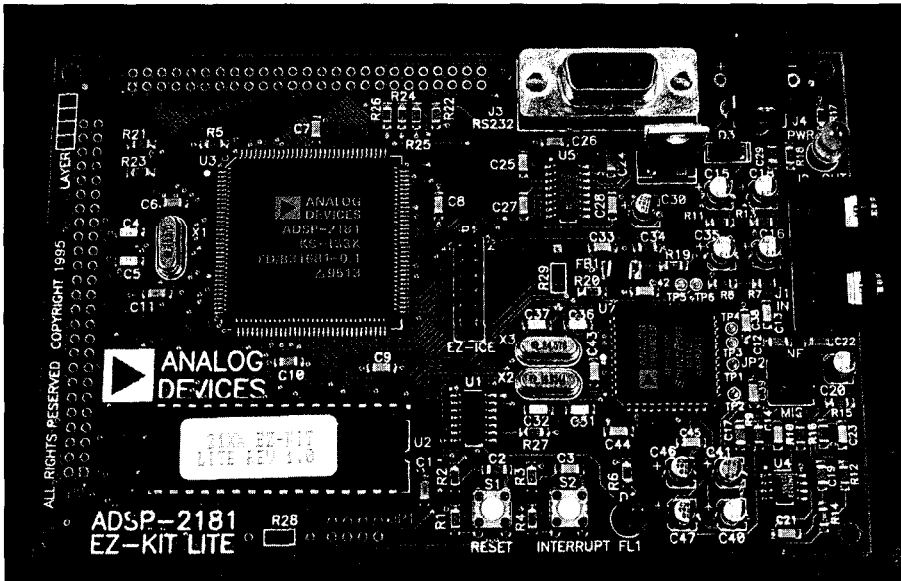


Photo 1-Despite its **small size**, the **EZ-Kit Lite**, combining a **DSP (ADSP-2181)** and **16-M stereo CODEC (AD1847)**, is a complete sonic subsystem that communicates with a PC via **RS-232 (9-pin)** connector. The EPROM contains monitor and demo **software** that is automatically **bootstrapped** into the DSP's on-chip RAM at reset.

operation like $MR = MR + (MX \cdot MY)$ puts the result in MR and sets flags in MF.

Shifter inputs include an operand (SI), exponent (SE), and block exponent (SB). As these registers' names imply, besides the usual bit spinning, the shifter supports operations (such as **NO RMalize**) that speed software floating point.

The DAGs are responsible for generating the operand addresses and consist of **I** (index), **M** (modify), and **L** (length) registers. Naturally, the **I** register holds the operand address with **M** automatically added (postmodify) after each access. A load instruction takes the form $reg = DM/PM (In, Mn)$ where **DM/PM** refers to the data or program memory, respectively. The **L** register supports modulo addressing (i.e., it defines the length of a circular buffer and automatically corrects **I** for wrap-around). **DAG1** can also be configured to automatically generate bit-reversed addresses, a technique commonly used in FFT computations.

The program sequencer contains a variety of status registers including **ASTAT**, **MSTAT**, and **SSTAT** for the **ALU**, **MAC**, and **shifter**, respectively. Notably, most instructions feature conditional execution based on a status (**Z**, **C**, **V**, etc.) flag.

Particularly important for DSP applications is a dedicated **CNTR**

register and specialized stack, which work together to implement zero-overhead looping. Once the count, loop, and fall-through addresses are initialized, the loop automatically runs to completion without the need for the

typical termination test and conditional branch.

Listing 1 gives you a taste of '2181 programming with a loop (**DO 1 a be 1 UNT I L C E**), conditional execution (**I F M V SAT MR**), and program and data memory addressing (**M X 0 = DM (I 0 , M)**).

With shades of superscalar, you'll even notice that the '2181 can execute multiple instructions at once, typically combining operand load/stores and an **ALU** or **MAC** operation. In this case, the inner loop (**S o p :**) loads both operands (**MXO**, **MYO**) and **MACs** them in a single cycle.

The '2181's specific add-ons include two serial ports (**SPORTs**) that work just like the port on the '1847 (i.e., timeslots, companding, etc.), a high-speed 16-bit timer, and 13 parallel I/O lines (two are shared with **SPORT 1** and three are output only).

As shown in Figure 7, from the outside, the '2181 looks more like a simple single-chipper than a zillion-pin RISC. The bus interface consists of a 24-bit data and 14-bit address bus

HOT SWAPPING ELECTRONIC EXTENDERS

For PCI, ISA, VXI, EISA, VESA, Micro Channel and NuBus



Electronic PCI Extender /
PCI Mini Extender



Electronic ISA Extender /
PC Mini Extender

- Insert/Remove Cards With PC Power On!
- Save Time Testing And Developing Cards
- Save Wear On Your PC From Rebooting
- Adjustable Overcurrent Sensing Circuitry
- NO Fuses, All Electronic For Reliability
- Single Switch Operation W/ Auto RESET
- Optional Software Control Of All Features
- Breadboard Area For Custom Circuitry

Full line of passive extenders: PCI, ISA, VXI, EISA, VESA, MC and more



Passive PCI Extender



Passive EISA Extender



Passive MC32 Extender

**1 Year Warranty
and 30 Day
Money Back
Guarantee**

Call our 24 Hour Fax-On-Demand System for FREE Information.

AZ-COM

3343 Vincent Road, Suite D, Pleasant Hill, CA 94523

TEL: 800-209-2418 / FAX: 510-947-1900

strobed with separate data, program, common, I/O, and byte selects.

Recognizing that most applications execute on-chip, the Byte DMA controller (BDMA) reconfigures the bus by assigning eight of the data lines as addresses, boosting addressability to 4 MB. In particular, the BDMA solves the bootstrap problem since it can automatically load the on-chip RAM at reset.

Another way to boot and otherwise talk to the '2181 is via the Internal DMA (IDMA) port. The IDMA port consists of a dedicated 16-bit multiplexed address and data bus with the requisite read, write, strobe, and acknowledge control lines. Access is a two-step process that sets the address and then accesses the data.

Note that the address is automatically incremented, so a sequential

access doesn't need another address cycle. In this way, all the '2181s internal RAM (but not control registers) is made directly accessible via the IDMA port.

THE EZ WAY OUT

You can see that combining an ADSP-2181, AD1847, EPROM, and a few discretes should be easy. In fact, the only thing easier is picking up the phone and ordering the EZ-Kit Lite, shown in Photo 1.

I have to admit I'm a sucker for cute little demo boards, especially when they cost less than \$100 and do something useful. The EZ-Kit Lite easily fills both bills since \$89 not only includes the board, but also complete (not restricted demo versions) copies of the PC-based assembler, linker, and simulator.

Though the tightest loops will likely remain the province of assembly language, the DSP world is headed (like it or not) toward C along with everyone else. Fortunately, exploiting the evermore popular GNU C, a complete compiler, library, and debugger suite only costs \$395.

For instant gratification, the board comes with short-and-sweet demos (bandpass filter, ADPCM, DTMF, echo-cancel, etc.). Well, more like instant gratification in my case since shoving the disk in and typing `SETUP` got me a disturbing message: `This program requires Windows to install.`

It's time to 'fess up and admit I don't have Windows running. I do most of my office work on a Mac and have been able to get by, until now, using DOS for engineering. Sure, I've been planning to upgrade someday, but figured I'd hold off until about Windows 97.

Wanting to get on with it, I briefly considered installing Windows 3.1 on my brand X clone, but a quick glance under the hood (386/33, 1-MB RAM, 80-MB hard disk) brought me to my senses.

Instead, I simply dragged the demo setup (board, speaker, microphone, cables, disks, etc.-you know the drill) over to a Windows-enabled friend's house and checked it out there. I can attest that the demos work as advertised, but didn't really wring the board out since we spent most of the time fiddling with AUTOEXEC. BAT. I left my friend with a cheery "Thanks, bro", and don't forget to uninstall."

Funny thing is, once you get past the Windows-Or-Else install procedure, you'll find that most of the software is (ho-ho) DOS stuff anyway! In fact, some programs (such as the simulator) only work with plain-old DOS and won't run in a Windows DOS-box.

I don't mean to get on Analog Devices' case, though I wish they'd made the demo DOS friendly. Instead, just consider this a cautionary harbinger of things to come as the machinations of a desktop-driven Microsoft collide with the realities of embedded systems. ☹

Listing 1-- This simple FIR filter program illustrates a number of ADSP-2181 programming features including looping, simultaneous data transfer and computation, and conditional execution.

```
.MODULE fir-sub;

FIR transversal filter subroutine

Calling Parameters
  IO -> Oldest input data value in delay line
  LO = Filter length (N)
  I4 -> Beginning of filter coefficient table
  L4 = Filter length (N)
  M1, M5 = 1
  CNTR = Filter length - 1 (N - 1)

Return Values
  MR1 = Sum of products (rounded and saturated)
  IO -> Oldest input data value in delay line
  I4 -> Beginning of filter coefficient table

Altered Registers
  MK0, MY0, MR

Computation Time
  N 1 + 5 + 2 cycles

All coeff and data values are assumed to be in 1.15 format.

.ENTRY fir;

fir:  MR = 0, MK0 = DM (IO, M1), MY0 = PM (I4, M5);
      DO sop UNTIL CE;
sop:  MR = MR + MK0 * MY0 (SS), MK0 = DM (IO, M1),
      MY0 = PM (I4, M5);
      MR = MR + MK0 * MY0 (RND);
      IF MW SAT MR;
      RTS;

.ENDMOD;
```

Figure 7—Since most of the action is on-chip, the ADSP-2181 external interface can be kept simple. Notice how byte mode extends the address space to 4 MB. It's especially useful for storing various routines (including boot code) in a single EPROM and moving them on-chip as required. Meanwhile, the IDMA port provides a looks-like-a-RAM interface to the DSP's internal memory.

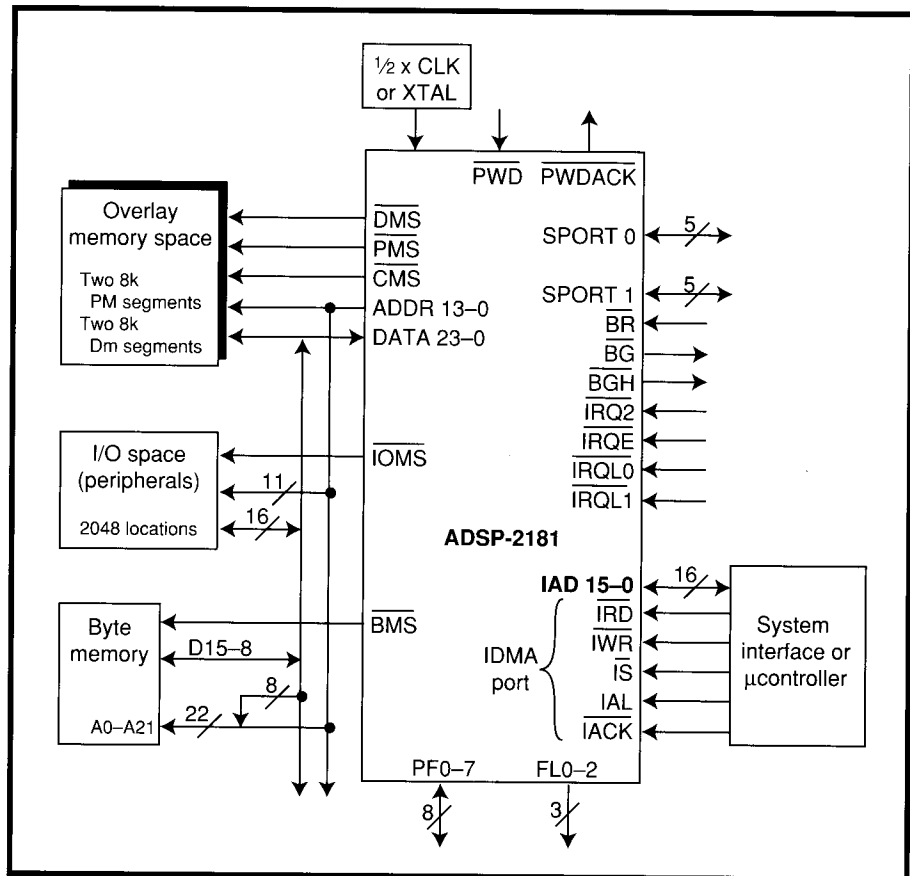
Tom Cantrell has been working on chip, board, and systems design and marketing in Silicon Valley for more than ten years. He may be reached at (510) 657-0264 or by fax at (510) 657-5441.

CONTACT

Analog Devices
 One Technology Way
 Norwood, MA 02062
 (617) 461-3881
 Fax: (617) 821-4273

IRS

419 Very Useful
 420 Moderately Useful
 421 Not Useful



Home Control is as simple as 1, 2, 3...

Energy Management



Access Control



Coordinated Home Theater



Coordinated Lighting



Monitoring & Data Collection

Get all these capabilities and more with the Circuit Cellar HCS2-DX. Call, write, or fax us for a brochure. Available assembled or as a kit.

1) The Circuit Cellar HCS2-DX board is the brains of an expandable, network-based control system which incorporates direct and remote analog and digital I/O, real-time event triggering, and X-10 and infrared remote control. Control programs and event sequences are written on a PC in a unique user-friendly control language called XPRESS and stored on the HCS2-DX in nonvolatile memory.

2) The Relay BUF-Term provides hardwire interface protection to the HCS2-DX. Its 16 inputs accommodate both contact-closure and voltage inputs within the range of ± 30 V. Its eight relay outputs easily handle 3 A AC or DC, to directly control solenoids, motors, lamps, alarm horns, or actuators.

3) The PL-Link provides wireless X-10 power-line control to the HCS II system. The PL-Link is an intelligent interface that sends, receives, and automatically refreshes X-10 commands. It works with all available X-10 power-line modules.

Of course, there's a lot more! The HCS II system has phone and modem interfaces, infrared remote control, voice synthesizers, and much more. Whatever your application, there's a configuration to accommodate it.

GET STARTED TODAY WITH AN HCS2 "123-PAK"

The 123-PAK consists of an HCS2-DX board, Relay BUF-Term board, PL-Link board, TW523 power-line interface, PS12-1 power supply, serial cable, and V3.0 HCS XPRESS software.

Assembled \$651

Kit \$461

CIRCUIT CELLAR, INC.

4 Park Street, Vernon, CT 06066
 (860) 875-2751 • Fax (860) 872-2204

CONNECT TIME

conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(860) 871-1 988-Four incoming lines
Internet E-mail: sysop@circellar.com

If you haven't already read my editorial, go check it out if you're one of the many who've been asking us to put together a Web page.

While it's far from complete, it's a *start*.

In this month's BBS threads, we start by looking at an elegant way of packing more into a limited space without resorting to complicated compression algorithms. In the other discussion, many people immediately blame equipment failures on heat without taking into account another likely factor: humidity.

Information Theory

Msg#:39105

From: Dave Ewen To: All Users

I have what I think is a simple information theory problem: If I have, say, 37 symbols available (not a power of 2), and want to transmit a bit stream with them, how could I map these symbols to make best use of them?

Msg#:39201

From: Russ Reiss To: Dave Ewen

If all symbols occur with equal likelihood, about all you can do is use 6 bits to encode the symbols, since it's the smallest integer number of bits that can represent your 37 symbols. But, since you mention information theory and efficiency of transmission, let's presume that your symbols don't occur with equal probability (for example, alphanumeric spelling out words in some language).

It can be more efficient in this case to design a code in which the more-likely-to-occur symbols use shorter code sequences, whereas less-likely symbols have longer codes. On the average (simply the weighted sum of the code length times the probability of occurrence of the symbol, summed over all symbols), your code could be shorter than the 6-bit "brute force" code approach.

It's been said that some of this technique was applied to Morse Code. For example, the letter E, which has high probability of occurrence, is encoded as a single dot, whereas the letter Q, with low probability, is a longer dash-dash-dot-dash.

Even with numbers thrown in, there are no symbols longer than five code elements (dots and dashes), except when you get into punctuation, most of which is six elements long. In fact, all the letters are four or fewer elements long. All the numbers are exactly five elements long. But

it's not really a binary code, since the spaces between code elements are as important as the elements. That's the only way letters (symbols) are separated from one another. So, don't be too impressed by the seeming efficiency.

You will find techniques for generating the most efficient codes possible based on whatever symbol probability you have in classical books on information theory.

Msg#:39542

From: Dave Ewen To: Russ Reiss

Well, it finally dawned on me that the simple solution was that some symbols would be worth 6 bits and some 5 bits. But I'm not sure that is the optimum solution; as you say, frequency of expected usage could be important..

Msg#:40448

From: Russ Reiss To: Dave Ewen

It's all up to frequency of occurrence! Consider the extreme, that one symbol occurred with 99.99% frequency (I never said it was realistic!). You'd want that symbol to be coded with a single bit if possible. You really don't care how long the codes for other symbols would be. Your average message length (over a long sequence of symbols) would come very close to 1 bit per symbol in this case.

Now, I'm not saying this is typical at all (I often use extreme cases to illuminate a point), but if there is any reasonable order to the symbol probabilities (i.e., not purely random), then you can shorten the average message length by using the right code.

Msg#:40814

From: Lee Stoller To: Dave Ewen

Another possibility that avoids the use of variable-bit-length coding would be to use your 64 6-bit numbers to encode your 37 symbols directly, and use the 27 numbers left over to encode common symbol combinations. (If we were encoding English text, some of the single symbols would be used to encode common words like "the," and others might encode common letter combinations.)

Msg#:41909

From: Dave Ewen To: Russ Reiss

This stuff about probabilities works fine with ASCII text, but my interest is something similar to UUencoding.

CONNECTIME

The question I'm pondering now is how I might make use of unused range. For example if I have a variable that ranges over integer values 0-87, I need 7 bits to encode it, but the possible values 88-127 are (seemingly) wasted.

Msg#:41953

From: Russ Reiss To: Dave Ewen

Well, it works with any symbols which do not have equal probability of occurrence (i.e., completely random]. About the only thing you might do with unused symbols is what was suggested by another: use them to encode pairs or triples of symbols, especially those which might occur with higher probability (if there are any). Even if all equally likely, you'd gain a little this way, and not waste the codes.

Msg#:42099

From: Robert Lunn To: Dave Ewen

Gee, among all this discussion of probabilities, variable bitlength hamming codes, and so on, I'm almost afraid to suggest it. Why don't you just store your codes as base-40 numbers?

This is an ancient technique that was used for storing text strings in compilers (error messages, etc.) back when a minicomputer with 16 KB of memory was a large machine.

A base-40 number obviously lets you encode 40 symbols, which nicely covers the letters A-Z, the digits 0-9, and space, comma, full stop, and "?" for punctuation.

Now $40 \times 40 \times 40 = 64,000$. So three base-40 numbers fit neatly into one 16-bit word. Storage is reduced by 30%, and the encode and decode functions are trivial.

Msg#:43821

From: James Meyer To: Robert Lunn

Three numbers in one 16-bit word, huh? Sixteen bits divided by three is five and one third bits. If you throw away the one third bit, you have five bits per word left. That's only 32 different characters. If you can provide further illustration, I'd be grateful.

Msg#:46429

From: Robert Lunn To: James Meyer

Of course, Jim. I will even "prove" it.

Consider an alphabet. Indeed, consider the 26-letter Roman alphabet. Now, I'm thinking of a letter between A and Z. How can I tell you which one? Clearly, I have to give you sufficient information to distinguish between 26 different possibilities or states.

If the symbols I am using to send information to you are each capable of this many states, then one symbol will suffice. If my symbols are more restricted (say, only a dot and a dash) then I can concatenate symbols to create the number of states that I need.

Now I want to send you two letters from the Roman alphabet. How much information must I send?

Assume the first letter is A. It can be followed by any of A-Z, and so this two pair combination takes 26 states.

Assume the first letter is B. It also can be followed by any of A-Z, and so this takes "another" 26 states.

You will see that to represent all possible two-letter combinations drawn from a 26-letter alphabet I will need 26×26 distinguishable states.

By similar extension you will see that to represent all possible three-letter combinations drawn from a 26-letter alphabet I will need $26 \times 26 \times 26$ distinguishable states.

Now let's expand the alphabet to include more characters. How about 40 characters? To represent all possible three-letter combinations drawn from a 40 character alphabet I will need $40 \times 40 \times 40$ distinguishable states.

So to send to you a specific three-letter sequence from my 40-character alphabet, I must give you sufficient information to pick one state from a possible $40 \times 40 \times 40 = 64,000$ states.

A concatenation of sixteen symbols drawn from a two-symbol alphabet can represent $2^{16} = 65,536$ states. Such a concatenation therefore provides more than enough states for my purpose.

Such a concatenation is conveniently represented in a computer by 16 binary digits. Thus, a 16-bit word is able to contain enough information to specify every possible sequence of three letters drawn from a 40-character alphabet.

Now let's show a specific implementation.

I suggested our 40-character alphabet comprise 0-9, A-Z, space, comma, period, and question mark. A convenient coding is:

0-9 becomes 00-09	"," becomes 37
A-Z becomes 1035	." becomes 38
"" becomes 36	?" becomes 39

Having coded our characters thus, we can store three characters, $c1, c2, c3$, in a 16-bit word j :

$$j = (((c1 \times 40) + c2) \times 40) + c3$$

The number j will never exceed 63,999 and so cannot overflow 16 bits.

Given j , we can retrieve the three characters:

$$\begin{aligned}c3 &= j \bmod 40 \\c2 &= (j \div 40) \bmod 40 \\c1 &= j \div 1600\end{aligned}$$

The singular advantage of this scheme is the simplicity of encoding and decoding the characters.

CONNECTIME

Msg#:47423

From: James Meyer To: Robert Lunn

I completely misinterpreted your original message and was trying to divide the 16-bit word into three pieces on an integer bit boundary. It's clear to me now what you were trying to show me and I'll never forget it. It's truly elegant!

Question: I can see that the base-40 scheme works well for optimizing storage, but how about transmission?

The base-40 method requires that the length of the message is either fixed and known in advance or is some multiple of three characters. For example, if I encoded a message by the base-40 method and had to send it 8 bits at a time, the receiver would have a hard time with the decoding if he missed the very first 8 bits.

Might that be one reason that many encoding schemes include a "null" character? I noticed that you didn't include one in your example.

Msg#:47428

From: Robert Lunn To: James Meyer

Well, actually I looked up your posts over the last few months. From these I could see that your background was

primarily in hardware (yes?) and in that area you obviously knew what you were doing. Therefore, you were simply misunderstanding me and it was worth my time to explain.

It is my experience that hardware people tend to have a more "nuts and bolts" approach to software, and can fail to see some of the subtleties involved. In the same way, of course, the many subtleties of solid-state electronics go right over *my* head.

The coding scheme was originally used (to my knowledge) for storing static character strings back when memory was much more limited than today. For this purpose it works OK.

However, it falls way short of "optimizing storage."

Even as a first approximation you can see that only 64000165536 = 97.7% of the "channel" capacity is used.

Transmission systems use much more complicated methods (which also address the issues of link establishment, flow control, error detection and correction, etc.) because the cost of implementing these methods is outweighed by the savings in reduced transmission times.

There are no control characters in the example I gave. It would, of course, be easy to define <39> as an escape code

WE'RE NOW #1

80C52-BASIC CHIPS IN VOLUME

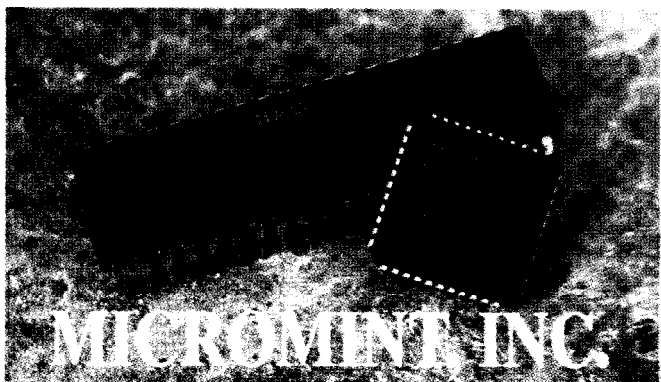
Micromint's 80C52-BASIC chip is an upgraded replacement for the venerable Intel 8052AH-BASIC chip. Ours is designed for industrial use and operates over the entire industrial temperature range (-40°C to +85°C). Available in 40-pin DIP or PLCC.

80C52-BASIC
chip **\$19.00**

OEM 100 qty. **\$12.00**

BASIC-52

prog. manual **\$15.00**



Call (860) 871-6170 or 1-800-635-3355 to order

MICROMINT, INC. 4 PARK STREET, VERNON, CT 06066

Microcontroller Networks (μ LANs)

With Cimetrics' 9-Bit μ LAN you can link together up to 250 of the most popular 8- and M-bit microcontrollers (8051, 80C196, 80C186EB/EC, 68HC11, 68HC16, PIC16C74).

The 9-Bit μ LAN is:

- Fast-A high speed (62.5k baud) multidrop master/slave RS-485 network
- Flexible-compatible with your microcontrollers
- Reliable-robust 16-bit CRC and sequence number error checking
- Efficient-low microcontroller resource requirements (uses your chip's built-in serial port)
- Friendly**- Simple to use C and assembly language software libraries, with demonstration programs
- Complete-includes network software, network monitor and RS-485 hardware
- Practical**— applications include data acquisition and distributed control

Microcontroller, PC Master, or Laptop

Intel 8051 HVAC/R Lighting Control

Intel 80C186EB/EC Point-of-Sale

Motorola 68HC16 Process Control Transportation

Microchip PIC16C74 Fire Detection Security

9-Bit RS-485 Multidrop Network

Cimetrics
TECHNOLOGY

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

CONNECTIME

and then interpret the following code as having some additional meaning.

However, note that all of the 16-bit numbers 64,000 through 65,535 are unused. All of these can be used to implement such flow control functions as <start of block>, <end of block>, <acknowledge>, <idle>, and so on.

Of course, the 16-bit words used for such a purpose cannot then also encode data.

Having reached consensus, I can make the following points:

You will now see that the statement in my first post that $40 \times 40 \times 40 = 64,000$ does in fact constitute proof of my original assertion.

You will also have realized when reading my last message that base-40 numbers (or any other type of number) are themselves simply letters drawn from some alphabet, or, more generally, simply symbols. Thus the whole exercise is just a conversion of one set of symbols into another.

Thus the whole argument becomes wonderfully circular. This is the region where number theory, complexity theory, information theory, graph theory, and so on all reveal themselves to be different ways of looking at the same thing. Many people claim that this says something profound about the universe. . .

Msg#:51294

From: James Meyer To: Robert Lunn

This BBS isn't called the *Circuit . Cellar for nothing. Yep, although I have done some programming, I'm better at circuit design.

Perhaps your original claim was proof, but it certainly wasn't obvious to this solder-slinger until you showed me the algorithm, though. An example or two makes it much easier for me. I guess that's just the way my mind works.

Running PC in warm environment

Msg#:17049

From: Lee Leduc To: All Users

I have a question about running a PC clone at a room temperature of 85°F. Is it safe to operate a PC at this temperature. I'm setting up a BBS and the building has no air conditioning. In ancient times, 10-15 years ago, I had problems with memory chips running in an ambient temperature above 90°. The specs I can find on the memory, CPU, and other chips in the system claim a commercial temperature range (0-70°C) for these components. I'm using a CPU fan and a temp. probe placed inside the case of the machine shows a temperature of <100°F. Any opinions would be welcomed.

Msg#:19083

From: Ken Simmons To: Lee Leduc

FWIW, I've had my system in a 90° environment and it hasn't had any problems.

As long as the internal temperature remains below 95°, you should be all right.

Oh, don't forget to regularly vacuum all the accumulated dust and such from inside your system. If any of the fan vents in the power supply get clogged, air flow is reduced and efficient cooling doesn't happen.

Msg#:19490

From: Janice Marinelli To: Ken Simmons

Sometimes humidity is the factor that pushes it over the top. Like humans, they seem to be able to take more heat as long as the humidity is not through the roof too.

Msg#:21196

From: Ken Simmons To: Janice Marinelli

I remember a situation where I installed an automated test system controlled by an H/P 1000 computer and a 7906 cartridge drive. The customer started having disk-access problems and data errors. I eventually traced the problem to too much humidity supplied by the room's air conditioner/humidifier-the thick air (I think it was 60%, noncondensing) was causing the disk read errors. As soon as the humidity was reduced to a more manageable 40%, the disk errors went away and never returned!

I sometimes ponder how PCs in "tropical" locales (i.e., East Coast in summer) manage to keep functioning with the high temps and humidities? I'm thankful I live in the Puget Sound area where the humidity rarely breaks 50% without condensation.

Msg#:23233

From: George Novacek To: Ken Simmons

If the computer is running, its internal temperature is above ambient and, therefore, the relative humidity inside the box is lower. What you must be wary of is water ingress due to high humidity, which happens when the machine is turned off and allowed to cool down to the ambient temperature.

This is one argument for leaving the computer on all the time (this is too controversial and a completely different subject). When you run temperature and humidity test cycles on electronic equipment, it is much more likely to fail after you have soaked it in a low temperature or high temp/high humidity environment and then powered up, as opposed to keeping it energized.

Any properly made electronic equipment should take 60% RH indefinitely, especially if we are talking an air-conditioned environment. The ambient temperature is

CONNECT TIME

maintained fairly low and, consequently, the interior of the equipment will work at significantly lower RH without exceeding its maximum temperature rating. The component aging and stress come primarily from the elevated temperature. You do not want to operate close to the maximum rating.

I would be a little apprehensive using a computer built with parts rated for 50°C max. operating temperature in Saudi Arabia. But we have had equipment built with 70°C rated parts running nonstop for years in Singapore, where humidity gets to 95% or higher without a problem.

Msg#:25152

From: Ken Simmons To: George Novacek

To a point, of course. If the ambient humidity is relatively high to begin with, the air pulled into the machine via the fan will also have that humidity. Once the air warms up, the internal humidity will rise above the external, just because of the nature of water vapor.

Having worked in environmental stress-screening for the past 9 years, I've witnessed this phenomenon and.. it's not pretty sometimes.

>This is one argument for leaving the computer on
>all the time.

True. However, it's best if you can control the humidity somewhat inside the machine. One way is desiccant packs, available from packing-supply companies. The 1" x 1" packs are relatively cheap and you can toss a few into your machine for a month (or so?) of dehumidification.

Any properly made circuit board that expects to operate on 60% RH *should* be conformal coated to minimize moisture infiltration, especially if that humidity should turn condensing. Even more so if the temperature should suddenly rise (i.e., heat wave).

Finally, you need to remember what high ambient humidity does to humans: it causes them to sweat profusely and that same sweat drips into places you don't want it to go. Add body salt (and other electrolytes) that are mixed with that sweat and you have the makings of electronic disaster via unwanted conductive paths, etc. Another argument for dehumidification or conformal-coating of electronic gear (especially consumer-oriented gear).

Otherwise, what you state is quite valid.

>we have had equipment built with 70°C rated parts
>running nonstop for years in Singapore, where humidity
>gets to 95% or higher without a problem.

Sounds like you've found winners there in that equipment. One caveat, though: you should do periodic checks

for corrosion, especially on all nongold-plated connections (e.g., printer and modem cables). You'd be amazed at what humidity will do to mere cadmium-plated, nickel-plated, and chrome-plated surfaces after awhile.

Msg#:28126

From: George Novacek To: Ken Simmons

>Once the air warms up, the internal humidity will rise
>above the external, just because of the nature of water
>vapor.

Quite the opposite. Once the air has warmed up, its relative humidity drops. This is precisely why you have static electricity problems in the wintertime. The following is an excerpt from an encyclopedia:

"...Humidity, the water-vapor content of air, is expressed as a fraction of the mass of water vapor in a given mass of air-usually taken as 1 kg (2.2 lb.)-it is called specific humidity. A variable quantity, it ranges from 0% in very dry air to as much as 4% in very humid air.

"The upper limit of the amount of water that air can hold in vapor form is called the saturation specific humidity. At surface atmospheric pressure (1,000 millibars) the maximum is 0.1 g of water vapor per kilogram of air at a temperature of -40°C (-40°F); at the freezing point the ratio is 3.8 g/kg; at 20°C (68°F) it is 15 g/kg, and at 40°C (104°F), it is 50 g/kg, or 5%.

"Relative humidity (RH) is actual specific humidity measured as a percent of saturation specific humidity. For example, if relative humidity is 90% at the freezing point (a very high value), then specific humidity is 3.4 g/kg. Relative humidity is the most widely used humidity indicator.. ."

What the above boils down to is that if you had 95% RH at 68°F ambient and warmed up the air inside the computer to 104°F, the RH *would* *drop* from 95% to a mere 28%. Then, let the equipment cool down and if its temperature drops below 68°F where the RH hits 100%, or the saturation point, you'll get condensation, soaking the electronics in water.

Where I come from, a properly made PCB *is* conformally coated. I'm not familiar with PC manufacturers' standards. Again, it's not the heat wave you should worry about, but a sudden temperature drop-that is when you get condensation (e.g., fog).

>Sounds like you've found winners there in that equipment.

We did not find the winners, we built them. However, if you want to swap stories of what corrosion can do to

CONNECT TIME

equipment, count me out. It's just too depressing. Seeing equipment after only a couple of days of salt fog test makes me cry. :-)

Msg#:32012

From: Ken Simmons To: George Novacek

Given that information, if the starting air doesn't have much humidity (absorbed water vapor) to begin with, I have to agree that warming said low-moisture air, without providing more moisture to balance it, will result in a humidity decrease. Am I right in concluding that's what you're trying to say here?

However, if you do provide more humidity to that warmed air, the RH will rise again, regardless of the target temperature. That's what I see happening in the PC environment with a continuous flow of air into and out of the machine. There's always a little bit of stagnant air inside that'll tend (as I see it) to absorb any moisture and keep it there, especially considering the rather poor air flow from the typical case vent design.

Now, I agree that if you have an initially warm, humid atmosphere and you cool it, you'll get condensation/precipitation (a common occurrence here in the Puget Sound area). However, with the cooling mechanism used for the PC innards, the temperature differential between interior and exterior will rarely be more than 10-20°F. Let me explain.

I have a thermometer on my home system monitoring the PC fan exhaust (I have a '286 system). Even when the ambient temperature is over 80°F, the exhaust temperature rarely breaks 90°F! Even the normal exhaust temperature, with 70-75°F external temperature, only hovers around 85°F.

Additionally, I have what's called a silencer, which is a temperature-controlled fan-speed regulator attached to my PC's fan. This results in having the cooling fan operating at half speed (i.e., approx. 6 VDC applied to the 12-VDC fan), which not only reduces the noise, it reduces the internal dust buildup due to a lower flow rate. I've had it in my machine for over 7 years and it rarely (if ever) speeds up due to over-temperature problems, even in summer.

Now, when you talk about your typical tropical environments, where the RH is 80+% for 90+°F temperature, the humidity change will not be that dramatic, if at all, when that hot, humid air is drawn into the (perhaps) warmer PC inside. Furthermore (and I think you'll agree with me here), humid air is a poorer heat conductor than dry air (relatively speaking). Therefore, PC innards in a tropical (or desert) environment will not be cooled as well as one in a more temperate environment.

It's the nature of water vapor, once heated, to resist further attempts at absorbing more heat as the vapor concentration increases. That's why there's very little percep-

tual difference between 70% and 90% RH compared to between 40% and 70% RH. Add dissolved salt to that humid, tropical air (i.e., coastal environments), and you'll agree you have the makings of a very hostile environment for modern electronics (which you mention later).

>Where I come from, a properly made PCB *is* conformally >coated. I'm not familiar with PC manufacturers' standards.

True. If you look at PC motherboards, they do have a very minimal coating that, to my observation at least, appears to be nothing more than the solder mask. The chips and sockets are bare to the environment, so they're gonna get the brunt of punishment.

As to the cooling factor, that'll happen when you turn it off, regardless of the ambient environment. It is that power cycling which, I think you'll agree, is what causes the damage to occur with corrosion.

Where I work (Boeing Military), our PWAs are encapsulated with either silicone or urethane resin (Hysol, PC-18M) so they can handle salt-fog, dust, and humidity without failing.

I used to clean equipment that had been through salt-fog qualification testing. Aside from crusting, our stuff washed clean (with lots of warm, deionized water, that is!) and operated to spec after being fogged and summarily cleaned.

And I second your grief regarding corrosion damage on consumer- and industrial-grade stuff. All the military-grade stuff I've worked on seemed to be relatively immune from any type of corrosion.

Msg#:36240

From: Pellervo Kaskinen To: Ken Simmons

> However, if you do provide more humidity to that >warmed air, the RH will rise again, regardless of the target >temperature. That's what I see happening in the PC.

I fail to see it happening that way at all! True, *if* you introduce more moisture, the RH of course starts increasing, but where does that additional moisture come from?

Surely you are not suggesting there are some famous Demons, selectively kicking each water molecule from the dryer air into the more humid pocket you advocate.

The only pockets approaching such behavior are found where pressurized air expands and cools, not when the temperature is increasing, with pockets or not.

The reason why you might see more failures when the humidity increases is not related to the temperature increasing locally. It is due to the higher relative humidity in the general atmosphere, which means that even after the

CONNECT TIME

RH is reduced by heating, it is not reduced to as low a value as it would be when the ambient RH is lower.

What the higher temperature does in addition to the higher RH is promote the migration of water through the polymeric materials, such as plastic packages of the semi-conductors. Once the water reaches the chip, all kinds of problems may arise. (Yes, Jim, I know the chips have a protective passive layer :-)) Corrosion of the very thin bonding wires is just one of the things.

Now, although we have been talking about increased RH causing problems to electronics, the opposite is much more true in my books. The low RH gives rise to some pesky static discharge damages. I may not have seen the high humidity as a real problem in the equipment we build, but I sure have seen the dead chips due to static electricity every year after the beginning of the heating season.

We invite you to 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 (860) 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.

Software for the articles in this and past issues of *Circuit Cellar INK* 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.

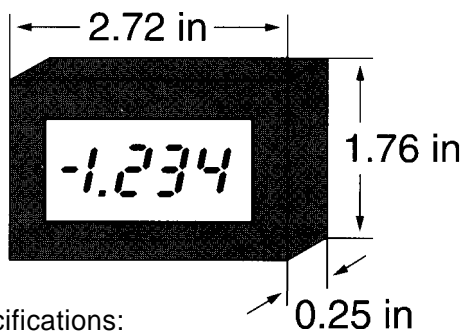
To order Software on Disk, send check or money order to: Circuit Cellar INK, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your Visa or MasterCard and call (860) 8752199. Be sure to specify the issue number of each disk you order. Please add \$3 for shipping outside the U.S.

IRS

422 Very Useful 423 Moderately Useful 424 Not Useful

3½-DIGIT LCD PANEL METER

-Available now at an unheard of price of **\$15** plus s & h
New! Not surplus!



Specifications:

- Maximum input: k199.9 mV
additional ranges provided through external resistor dividers
- Display: 3½-digit LCD, 0.5 in. figure height, jumper-selectable decimal point
- Conversion: Dual slope conversion, 2-3 readings per sec.
- Input Impedance: >100M ohm
- Power: 9-12 VDC @ 1 mA DC

Circuit Cellar, Inc.

4 Park Street, Suite 12, Vernon, CT 06066
Tel: (860) 875-2751 Fax: (860) 872-2204

The only
8051/52
BASIC
compiler
that is
100 %
BASIC 52
Compatible
and

has full
floating
point,
integer,
byte & bit
variables.

- Memory mapped variables
- In-line assembly language option
- Compile time switch to select 8051/8031 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 compatible
- Compatible with all 8051 variants

■ **BXC51 \$ 295.**

508-369-9556

FAX 508-369-9549



Binary Technology, Inc.

P.O. Box 541 • Carlisle, MA 01741



STEVE'S OWN INK

They Still Flip Hamburgers, Don't They?



The Circuit Cellar *INK* crew had a great time at the Embedded Systems Conference in San Jose. Not only was there no question about Circuit Cellar's dominant niche in the publishing market, embedded-control manufacturers have come to truly recognize the caliber and excellence of our readers. By the end of the show, we had spoken with all the major embedded-system players, committed to some really neat projects for next year, and scheduled a couple of ambitious industry-sponsored design contests.

As you might expect, everyone leaves this kind of convention thinking that every discipline is ultimately a candidate for embedded control. Implementation is just a matter of timing and price.

To physically test this concept, I stopped at the Restaurant Equipment Manufacturer's convention in Las Vegas before returning. Given all the compufunctions, LCDs, bells and whistles jammed into the average consumer toaster, I figured they'd probably stuffed a **miniCray** into a **Hobart** commercial dishwasher by now. I expected you could measure water temperature, **pH**, dissolved solids, viscosity, soap-film density, residue reflective indices, motor currents, sprinkler volume, droplet velocity, and on and on. Surely even washing a dish would now be a whole new experience.

Well... talk about an undiscovered country. The kitchen side of the food industry must have "If it ain't broke, don't fix it!" chiseled on their butcher block. With the exception of a few OSHA-mandated safety interlocks, today's dishwasher and practically everything else in the restaurant is the same as what I used during high school.

Unless the computer controls are so embedded that they're invisible, I saw very little in the way of mechanical-control replacement or innovative new features requiring microcontroller technology. The \$20,000 gas ovens use piezoelectric mechanical starters and bimetallic temperature regulators. Refrigerators? Well, they're still plain old refrigerators. No built-in bar-code inventory systems yet.

Except for a couple of electronic temperature-probe manufacturers ardently searching for customers, there was a phenomenal lack of sophistication in food-preparation equipment. The one light on the horizon, brightly shining with a plethora of formula and concoction alternatives, was the abundance of automatic cappuccino and espresso machines. The literature from Acorto proudly describes how its powerful microprocessor monitors and controls all the major functions to produce one of any 26 drinks "at the touch of a button."

While not completely discouraged, it was obvious that my search for automatic electronic hamburger flippers was useless. In fact, I wondered if there wasn't even a bit of regression going on.

In one booth, I smelled hamburgers cooking. The tray of samples attracted my attention that much more. I listened to the sales pitch about it being the fastest hamburger cooker available and how by using "direct energy transfer," it could cook two burgers in 25 seconds with no flipping. To everyone else, "direct energy transfer" was obviously something wonderful and worth every penny. To an electrical engineer, it was a provocation: "Say what?"

The salesman didn't really want to explain direct energy transfer. After all, a heating element under a pan transfers heat to the food, doesn't it? The key word is "direct." Take a couple %-lb. chunks of ground beef, squash them between two stainless steel electrodes, apply 2000+ watts of energy for 25 seconds, and I'd say you could call that direct energy transfer! In nonfood-speak, it's still electrocuted beef.

Deep under the covers, I suspect a micro modulates the current to keep the burger from becoming a charcoal briquette, but the salesman didn't know. The ultimate shocker wasn't the technique, but the price. At \$1900, I decided to cook hamburgers the old-fashioned way with catsup and with or without embedded control.

So much for introducing new technology to restaurant kitchens.